Часть первая
Общие принципы построения и арифметические основы ЭВМ

За последнее десятилетие компьютер стал не только помощником, но незаменимым инструментом практически в любой области человеческой деятельности. Компьютер позволяет в считанные доли секунды точно и красиво выполнять операции, требовавшие от человека многих часов напряженной и кропотливой работы, открывает новые перспективы для творчества, дает возможность объять то, что прежде считалось необъятным. Развитие глобальных компьютерных сетей позволяет быстро и качественно обмениваться информацией, даже если людей разделяют многие тысячи километров.

Компьютер вошел в обиход многих людей, но для того чтобы в полной мере использовать богатейшие его возможности, мало освоить тот или иной пакет прикладных программ. Чем лучше человек знает устройство и принципы работы ЭВМ, тем больше может дать ему компьютер. Для того, кто умеет программировать, открываются просто необозримые горизонты. Для того, чтобы получить представление об общих принципах работы электронно-вычислительных машин, необходимо познакомиться не только с общей схемой компьютера (ЭВМ), составом устройств, их назначением, арифметическими и логическими основами ЭВМ, но прежде всего понять последовательность этапов решения задач, требующих использования компьютера.

В последовательности этапов и в их содержании

заключается то важное, что позволит решать не только учебные задачи, но и сложные практические, с которыми сталкиваются специалисты в любой сфере профессиональной деятельности человека.

Общие принципы построения и работы ЭВМ

Для понимания принципов построения и работы ЭВМ необходимо понять и усвоить последовательность решения задач. В общем случае эта последовательность складывается из следующих этапов:

1) постановка задачи и разработка ее математической модели в виде формульных зависимостей;

2) выбор метода численного решения (если необходимо);

3) разработка алгоритма решения задачи, т. е. определение последовательности элементарных действий, приводящих к искомому результату;

4) написание программы на одном из языков программирования;

5) реализация вычислительного процесса с помощью компьютера.

Если первые четыре этапа всегда реализуются человеком, то выполнение последнего этапа зависит от компьютера-исполнителя программы. Для создания эффективной и рациональной программы разработчик должен не только знать язык программирования, но и представлять себе, как осуществляется вычислительный процесс в недрах компьютера.

Реализация вычислительного процесса с помощью ЭВМ

Рассмотрим простейший пример. Пусть необходимо производить вычисления в соответствии с выраже-

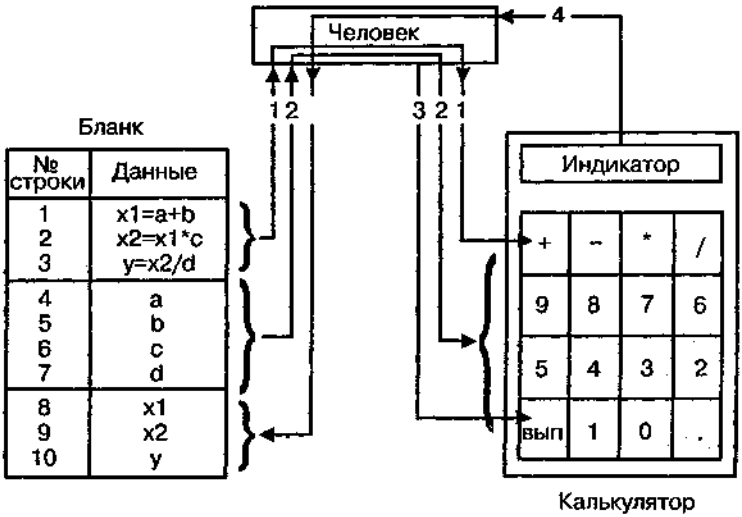


Схема 1. Реализация вычислительного процесса

нием

$$y = (a + b) \cdot c/d. \qquad (1.1)$$

Тип и диапазоны изменения данных заданы. Это выражение можно представить в виде следующей последовательности элементарных действий:

$$x_1 = a + b, \quad x_2 = x_1 \cdot c, \quad y = x_2/d. \qquad (1.2)$$

Схема реализации вычислительного процесса по алгоритму (1.2) в случае, когда исполнителем является человек, представлен на схеме 1.

Процесс вычислений складывается из последовательной реализации каждой формулы (1.2) путем выполнения следующих действий:

{1} — ввод в программируемый калькулятор последовательности действий;

{2} — чтение человеком из бланка данных (a, b, ...) и ввод их в калькулятор;

{3} — исполнение калькулятором заданной программы (ВЫП);

{4} — чтение человеком результата с индикатора (x_1, x_2, y) и запись его в бланк.

Номера указанных действий приведены на схеме 1. В подавляющем большинстве практических случаев решение задачи требуется получить за приемлемое время, т. е. исполнитель алгоритма должен обладать высоким быстродействием, намного превышающим возможности человека. Например, процессоры современных компьютеров выполняют более 1 миллиарда операций в секунду. Необходимость создания таких быстродействующих исполнителей и привела к появлению автоматических исполнителей алгоритмов — электронных вычислительных машин (ЭВМ) или компьютеров.

Состав устройств ЭВМ и их назначение

Состав основных устройств ЭВМ и их назначение определяются схемой реализации вычислительного процесса, пример которой приведен на схеме 1. В соответствии с ней в состав ЭВМ должны входить следующие устройства:

- память (ее называют оперативной памятью — ОП), которая предназначена для хранения соответствующей алгоритму программы действий и данных (выполняет функции бланка);
- центральный процессор (ЦП), предназначенный для управления выполнением операций в соответствии с заданным алгоритмом (выполняет функции калькулятора по реализации действий {1}...{4});
- устройство ввода (УВв), обеспечивающее запись программы и данных в ОП;

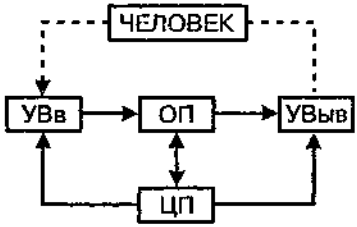


Схема 2. Общая схема ЭВМ

- устройство вывода (УВыв) для вывода результатов решения задачи из памяти в виде, удобном для человека.

Эти устройства и образуют общую схему ЭВМ, приведенную на схеме 2.

Принцип программного управления

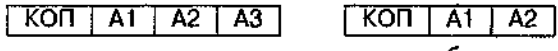
Работа современных ЭВМ осуществляется в соответствии с принципом программного управления, сформулированным американским ученым Дж. фон Нейманом. Сущность его заключается в следующем.

Алгоритм решения задачи в виде последовательности управляющих слов-команд и данных кодируется внутри ЭВМ в виде двоичных кодов. Двоичный код обеспечивает наиболее удобную реализацию функций устройств компьютера и их высокую надежность.

Команды и данные размещаются в оперативной памяти (ОП), состоящей из ячеек памяти, имеющих свои номера — адреса.

Для удобства работы со схемами общеприняты следующие сокращения:

ЦП — центральный процессор
ОП — оперативная память
КОП — поле кода операции
УУ — устройство управления
АЛУ — арифметико-логическое устройство
СчК — счетчик команд



а *б*

Схема 3. Поля машинных команд
(а — трехадресная команда;
б — двухадресная команда)

РК — регистр команд

ПЗУ — постоянные запоминающие устройства

ВЗУ — внешние запоминающие устройства

НМД — накопители на жестких магнитных дисках (в обиходе их называют винчестерами)

НГМД — накопители на гибких магнитных дисках (дискеты)

ПУ — периферийные устройства

КПУ — контроллеры для подключения и управления взаимодействием периферийных устройств с другими устройствами ЭВМ

МК — мультиплексный канал

СК — селекторный канал

ПО — программное обеспечение

СП — системы программирования

КПТО — комплект программ технического обеспечения

ППП — пакеты прикладных программ

ЯВУ — язык программирования высокого уровня

Команды представляют собой закодированные управляющие слова, в которых указывается какое выполнить действие, а также из каких ячеек памяти считывать операнды (данные, участвующие в операции) и в какую ячейку записать результат. В соответствии с этим каждая команда имеет поля: поле кода операции (КОП) и адресное поле (см. схему 3).

В адресном поле указываются адреса операндов (А1, А2, А3). Команды могут иметь три или два адреса (иногда даже один адрес). Адреса А1 и А2 определя-

Таблица 1

	Адреса команд и данных	Коды команд и данных КОП	A1	A2	A3	Комментарии
Программа	001	01	004	005	008	$x_1 = a + b \Rightarrow 008$
	002	03	008	006	008	$x_2 = x_1 \cdot c \Rightarrow 008$
	003	04	008	007	008	$y = x_2/d \Rightarrow 008$
Данные	004	$< a >$				
	005	$< b >$				
	006	$< c >$				
	007	$< d >$				
	008	$< x_1 > < x_2 > < y >$				

ют номера ячеек памяти, в которых хранятся 1-й и 2-й операнды соответственно. Адрес А3 определяет номер ячейки для записи результата. В двухадресной команде результат записывается по адресу А1 (1-й операнд при этом стирается в памяти).

Последовательность команд, реализующих заданный алгоритм, образует машинную программу решения задачи. В таблице 1 приведена простейшая машинная программа, реализующая алгоритм (1.2).

Все адреса и команды для наглядности представлены в таблице в десятичной системе счисления.

При этом использованы следующие коды операций (КОП):

01 — сложить; 03 — умножить; 04 — разделить.

Программа содержит три команды, размещаемые в ячейках 001, 002 и 003. При последовательном выполнении этих команд промежуточные значения переменных (x_1, x_2) и окончательный результат (y) записываются в одну и ту же ячейку с адресом 008. Это возможно потому, что при чтении из оперативной памяти (ОП) содержимое ячейки сохраняется, а при записи —

«старое» стирается, «новое» записывается.

Каждая команда программы имеет свой адрес, который автоматически формируется в центральном процессоре (ЦП).

Запись программ в виде, приведенном в таблице 1, использовалась в ЭВМ первого и второго поколений. В современных ЭВМ алгоритмы решения задач описываются на одном из языков программирования высокого уровня (ЯВУ), а затем с помощью специальных программных средств переводятся (транслируются) на машинный язык для их выполнения компьютером (ЭВМ).

Таким образом, чтобы решить задачу с помощью компьютера, необходимо «записать» в его память машинную программу, соответствующую заданному алгоритму.

Работа устройств ЭВМ при автоматическом выполнении команд программы

Для автоматического выполнения последовательности команд (программы) в состав ЦП входят два основных устройства (см. схему 4):

- арифметико-логическое устройство (АЛУ);
- устройство управления (УУ).

АЛУ предназначено для приема из ОП кодов операндов и для выполнения над ними операций в соответствии с кодами операций команд.

УУ предназначено для приема кода очередной команды, считываемой из ОП по ее адресу, хранения этого кода в течение времени выполнения команды в ЦП, для формирования адреса следующей команды и для синхронизации работы всех других устройств при выполнении команд.

В состав УУ входит счетчик команд (СчК), в котором формируется адрес следующей команды. В простейшем случае это производится путем добавления

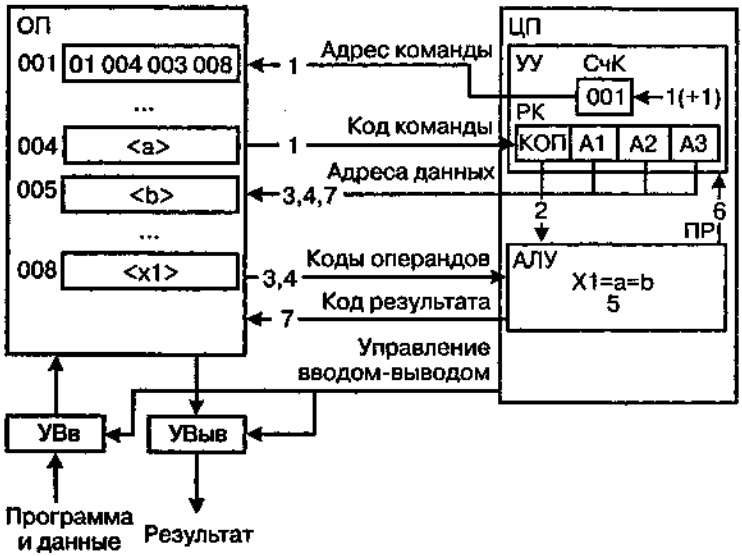


Схема 4. Работа ЭВМ при выполнении команд

единицы (+1) к адресу текущей команды. Для приема из ОП кода команды, его хранения и выдачи содержимого его полей в другие устройства используется регистр команд (РК). Другие узлы УУ, обеспечивающие управление выполнением команд, на схеме 4 не приведены.

После записи команд машинной программы и данных в ячейки ОП в СчК автоматически формируется адрес первой команды программы (в нашем примере — 001). С этого момента компьютер готов к автоматическому выполнению команд программы.

Программа реализуется компьютером путем последовательного исполнения ее команд в ЦП. При этом каждая команда выполняется в течение стандартного цикла работы УУ, состоящего из следующих этапов (см. схему 4):

{1} — выборка очередной команды из ОП в регистр

команд (РК) УУ по ее адресу в СчК и формирование адреса следующей команды в СчК (+1);

{2} — настройка АЛУ на операцию, код которой записан в поле КОП выбранной команды;

{3} — выборка в АЛУ 1-го операнда из ОП по адресу А1;

{4} — выборка в АЛУ 2-го операнда из ОП по адресу А2;

{5} — выполнение операции в АЛУ;

{6} — анализ результата в АЛУ и запись признака результата (ПР) в УУ;

{7} — запись в ОП по адресу А3 результата из АЛУ.

На этапе {6} при выполнении арифметических операций на основании анализа результата (он может быть = 0, < 0, > 0) вырабатывается соответствующий признак результата (ПР), двоичный код которого запоминается в УУ. Этот признак в дальнейшем может использоваться для организации в программах переходов с помощью специальных команд управления.

Выполнение программы обычно завершается выводом результатов из памяти, для чего используются специальные команды вывода, включаемые в программу.

Основные характеристики и классификация ЭВМ

Свойства ЭВМ любого типа оцениваются с помощью их технических характеристик, к основным из которых относятся: операционные ресурсы, емкость памяти, быстродействие, надежность, стоимость.

Операционные ресурсы — это множество реализуемых в ней операций, формы представления данных, их форматы, а также используемые способы адресации данных в памяти. Чем шире операционные ресурсы, тем больше аппаратурных затрат при построении ЭВМ.

Емкость оперативной памяти определяет общее количество ячеек памяти для хранения информации. Основной единицей хранения является байт (8 двоичных разрядов — бит), каждый из которых имеет свой номер (адрес). Емкость памяти измеряется в байтах, килобайтах — Кб ($K = 2^{10}$) или мегабайтах — Мб ($M = 2^{20}$).

Быстродействие ЭВМ определяется числом коротких операций типа сложения, выполняемых за 1 сек. Эта характеристика позволяет оценить в основном быстродействие процессора при выполнении определенного вида операций. Для более объективной оценки быстродействия ЭВМ при решении задач различных классов (научно-технических, экономических и т. п.) используют производительность ЭВМ, которая оценивается статистически на основе среднего числа задач определенного класса, решаемых на конкретной ЭВМ за единицу времени (час; сутки).

Надежность характеризует свойство ЭВМ выполнять свои функции в течение заданного времени без ошибок. В качестве показателя надежности обычно используется среднее время работы машины между двумя отказами в часах, которое определяется статистическим путем.

Стоимость ЭВМ определяется суммарными затратами на приобретение аппаратных и программных средств, а также на их эксплуатацию за определенный период времени (обычно за год).

Многообразие свойств и характеристик порождает и многообразие классификаций ЭВМ. В качестве основного признака часто используют размеры системы. По этому признаку различают: сверхбольшие, большие, малые и микроЭВМ.

Бурное развитие технологии и успехи в разработке программных средств приводят к сглаживанию различий между этими классами ЭВМ. Поэтому наиболее существенным признаком при классификации совре-

менных компьютеров является область их применения. По этому признаку различают: компьютеры общего назначения, проблемно-ориентированные и специализированные.

Компьютеры общего назначения отличаются большими операционными ресурсами, обладают памятью большой емкости и комплектуются широкой номенклатурой ПУ. Компьютеры этого класса эксплуатируются в вычислительных центрах и предназначены для решения широкого круга задач в различных сферах деятельности человека. Поэтому их часто называют **универсальными.**

Проблемно-ориентированные компьютеры используются для решения ограниченного круга задач, имеющих проблемное применение. Они сравнительно дешевы, просты в эксплуатации и обслуживании и рассчитаны на массовое применение. Наиболее часто подобные компьютеры используются в качестве управляющих в составе автоматизированных систем управления технологическими процессами (АСУ ТП), в системах автоматизированного проектирования (САПР) и т. п. Снижение стоимости ЭВМ данного класса достигается за счет разумного уменьшения операционных ресурсов применительно к конкретным проблемным задачам. В качестве машин этого класса используются мини- и микроЭВМ, в том числе и ПЭВМ (персональные компьютеры).

Специализированные ЭВМ используются для решения узкого круга задач с фиксированными алгоритмами, что позволяет наиболее рационально организовать вычислительный процесс, «приспособив» ЭВМ к выполнению указанных задач. Такая специализация позволяет увеличить их быстродействие, что весьма важно при управлении объектами в реальном масштабе времени. Обычно компьютеры данного класса используются в качестве бортовых (на самоле-

тах, ракетах, космических аппаратах, в автомобилях и т. п.).

Типовые структуры ЭВМ

Структура ЭВМ. Под структурой ЭВМ понимают совокупность ее устройств и связей между ними.

При построении архитектуры компьютера используется модульный принцип, согласно которому ЭВМ строится из набора устройств и блоков-модулей. Каждый модуль реализует законченные функции и обладает свойством независимости от других модулей. Модули объединяются в необходимую конфигурацию-архитектуру.

Соединение модулей производится с помощью шин (электрических цепей) для передачи по ним сигналов. Совокупность шин, связывающих два модуля, и алгоритм, определяющий порядок обмена информацией между ними, называется интерфейсом (сопряжением). Число шин в интерфейсе определяется разрядностью передаваемой по нему информации. Обычно это 8, 16 или 32 разряда (бита).

В зависимости от состава устройств и способа их соединения с помощью интерфейсов различают различные структуры ЭВМ.

Структура ЭВМ на основе единого интерфейса. Типовая структура (архитектура, конфигурация) ПЭВМ на основе единого интерфейса (общей шины) приведена на схеме 5.

Подобная структура является наиболее простой и широко используется для построения мини- и микроЭВМ (в том числе и ПЭВМ).

Все устройства компьютера подключаются к единому интерфейсу (общей шине). Периферийные устройства (дисплей, клавиатура, накопители на жестких (НМД) и гибких (НГМД) магнитных дисках, прин-

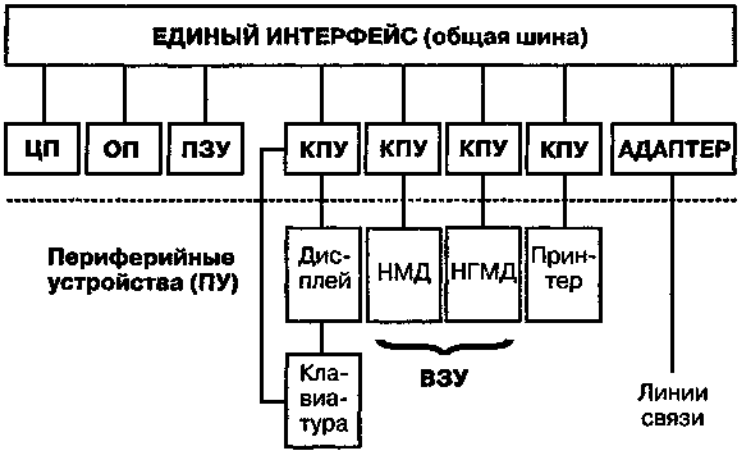


Схема 5. Структура ПЭВМ на основе единого интерфейса

тер) подключаются к интерфейсу через свои контроллеры ПУ (КПУ — управление периферийным устройством). Контроллеры обеспечивают сопряжение ПУ с другими устройствами ПЭВМ, а также синхронизацию обмена между ОП и ПУ.

В едином интерфейсе предусмотрены следующие основные правила обмена (протокол):

- передача производится «словами», длина которых обычно равна 16 бит (32 бит);
- одновременно обменивается информацией одна пара модулей (источник, приемник);
- непосредственный обмен между ПУ недопустим (источником или приемником всегда является или ЦП, или ОП).

Необходимая эффективность обмена в такой структуре достигается при небольшом числе ПУ. Очевидно, что использование единого интерфейса исключает возможность совмещения действий по обмену информацией между разными модулями, что приводит к снижению общей производительности ЭВМ.

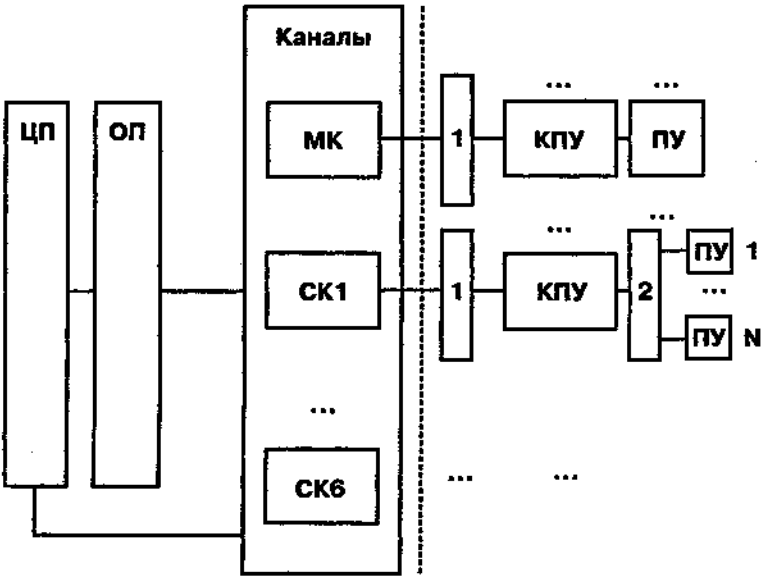


Схема 6. Структура ЭВМ на основе канала ввода-вывода (1 — интерфейсы ввода-вывода каналов, 2 — интерфейсы ПУ)

Структура ЭВМ на основе канала ввода-вывода. Данная структура отличается тем, что к ОП подключены с одной стороны ЦП, а с другой каналы ввода-вывода (схема 6). Периферийные устройства (ПУ) подключаются через свои контроллеры (КПУ) к интерфейсу ввода-вывода (В-В) каждого из каналов шириной 8 бит (байт).

При такой организации соединения в конфигурацию ЭВМ может включаться большое число периферийных устройств различного назначения. Каналы представляют собой специализированные процессоры, работающие под управлением своих (канальных) программ. В них используются специальные команды управления обменом. Включение каналов в структуру ЭВМ позволяет полностью освободить ЦП от обслуживания операций обмена. Следовательно, возможны

одновременное выполнение какой-либо программы в ЦП и обмен между периферийными устройствами (ПУ) и оперативной памятью (ОП) по другой программе. Это приводит к повышению производительности ЭВМ и возможности реализации многопрограммного режима (в ОП может находиться несколько выполняемых программ).

Обычно в состав каналов ввода-вывода (В-В) входит один мультиплексный канал (МК) и несколько (до 6) селекторных каналов (СК). К мультиплексному каналу (МК) подключаются относительно медленнодействующие периферийные устройства (принтеры, дисплеи и т. п.), которые в период обмена могут обслуживаться каналом одновременно. К каждому селекторному каналу (СК) подключаются до 8 относительно быстродействующих ПУ (ВЗУ на дисках и лентах). Селекторный канал обслуживает при обмене лишь одно периферийное устройство.

Такая структура ЭВМ характерна для машин общего назначения.

Общие сведения о программном обеспечении ЭВМ

Совокупность аппаратных и программных средств ЭВМ образует вычислительную систему. Программные средства ЭВМ, в свою очередь, образуют программное обеспечение, стоимость которого составляет до 75% стоимости системы.

Программное обеспечение компьютера предназначено для выполнения следующих функций:

1. Управление вычислительным процессом.
2. Организация управления вводом-выводом.
3. Организация диалога пользователя с машиной.
4. Предоставление пользователю сервисных услуг.
5. Автоматизация процесса подготовки программ и их выполнение в различных режимах работы ЭВМ.

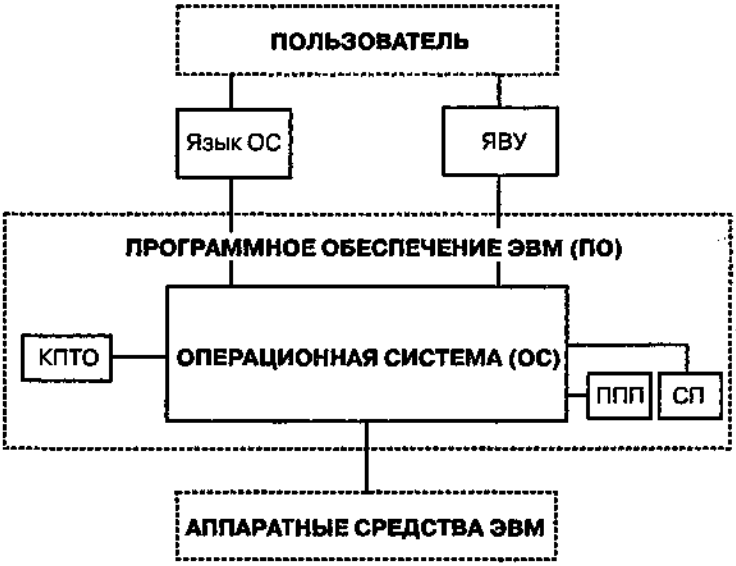


Схема 7. Структура программного обеспечения ЭВМ

6. Снижение трудоемкости работ по эксплуатации ЭВМ.

Состав программного обеспечения. В состав программного обеспечения ЭВМ (ПО) входят следующие компоненты (см. схему 7):

- общее (системное, базовое) ПО, к которому относят: операционную систему (ОС), системы программирования (СП) и комплект программ технического обслуживания (КПТО);
- специальное (прикладное) ПО — пакеты прикладных программ (ППП).

Общее программное обеспечение. Основным компонентом общего программного обеспечения (ПО) является операционная система (ОС), которая предназначена для выполнения с первой по четвертую функции программного обеспечения, указанные выше.

Основу операционной системы составляют управляющие программы, обеспечивающие:

- управление вычислительным процессом;
- планирование заданий пользователя;
- организацию диалога с пользователем;
- распределение ресурсов между программами пользователей;
- управление обменом.

Эти программы после загрузки системы постоянно хранятся в оперативной памяти (ОП) (являются резидентными). Помимо управляющих в состав операционной системы входят обрабатывающие программы, выполняющие следующие функции:

- сервис по компоновке и отладке программных модулей;
- хранение и настройка программных модулей;
- загрузка и исполнение программ.

Для разных классов ЭВМ используются различные виды операционных систем. Для каждой из них существует язык операционной системы — системные команды, с помощью которых пользователь общается с аппаратными средствами ЭВМ.

Системы программирования представляют собой комплекс программ, обеспечивающих выполнение 5-й функции программного обеспечения. Обычно с оперативной системой связан стандартный набор языков программирования высокого уровня (ЯВУ), к которым относят языки BASIC, PASCAL, C, FORTRAN и т. п.

Для каждого из языков программирования создается система программирования (СП), в состав которой входят следующие компоненты:

- описание ЯВУ;
- трансляторы различного вида;
- редакторы;
- компоновщики;
- библиотекари и т. п.

Основу системы программирования составляют программы-редакторы и программы-трансляторы.

Программы-редакторы предназначены для создания и редактирования программ на ЯВУ (исходных программных модулей).

Программы-трансляторы (или просто трансляторы) используются для преобразования (трансляции) исходных программных модулей в программы на машинном языке (объектные программные модули).

Различают трансляторы компилирующего типа (компиляторы) и интерпретирующего типа (интерпретаторы).

Компиляторы производят трансляцию всего текста программы на ЯВУ на машинный язык за один непрерывный проход (процесс) и создают программу на машинном языке, готовую к загрузке в ОП и исполнению. Для большинства ЯВУ используются компиляторы. Их основное достоинство — большая скорость исполнения готовой программы. Основной недостаток — сложный процесс отладки программы.

Интерпретаторы последовательно анализируют операторы программы на ЯВУ и осуществляют их преобразование на язык машинных команд с одновременным исполнением этих команд. Таким образом, интерпретатор должен постоянно находиться в оперативной памяти для выполнения программ, т. е. требует большей емкости оперативной памяти. Основное достоинство — простота отладки программы.

Комплект программ технического обслуживания (КПТО) включает в свой состав комплект контролирующих и диагностирующих тест-программ, которые используются при техническом обслуживании ЭВМ.

Пакеты прикладных программ (ППП). Пакеты прикладных программ — специально организованные программные комплексы для учета потребностей возможно большего числа пользователей. Каждый пакет

прикладных программ должен обладать своим входным языком для настройки конкретных программ из этого пакета, обеспечивающих решение задачи пользователя. Важной характеристикой пакета является уровень его интеграции. Уровень интеграции ППП определяется числом различных решаемых им задач и разнообразием сервисных услуг.

Различают методо-ориентированные ППП и проблемно-ориентированные ППП. Первые ППП используются для решения задач одного и того же класса различными методами (обычно это задачи вычислительного характера). Вторые ППП используются для решения задач, отличающихся по постановке и методам решения, но составляющих в совокупности одну большую задачу (проблему).

В ПЭВМ используются интегрированные пакеты для решения проблемно-ориентированных задач в области экономики, статистики, обработки текстов, обработки баз данных и т. п. Интегрированные пакеты — это программы, которые объединяются (интегрируются) в единый пакет.

Арифметические основы ЭВМ

ЭВМ являются арифметическими машинами, реализующими алгоритмы путем выполнения последовательных арифметических действий. Арифметические действия производятся над числами, представленными в принятой для ЭВМ системе счисления, в заданных формах и форматах с использованием специальных машинных кодов.

Системы счисления и кодирования информации

Под системой счисления понимается способ изображения чисел с помощью ограниченного набора

символов (цифр), имеющих определенное количественное значение. Системы счисления делятся на непозиционные и позиционные.

В непозиционных системах количественное значение символа определяется только его изображением и не зависит от его места (позиции) в числе. Например, в известной римской системе, использующей набор символов I, V, X, L, C, D, ..., десятичное число 38 представляется XXXVIII = 10 + 10 + 10 + 5 + 1 + 1 + 1.

Количественное значение числа определяется суммой (XXI) или разностью (IV) значений символов. Действие и значение символа зависят от места символа по отношению к другому символу, т. е. значение символа неоднозначно. Так, число 99 в римской системе изображается XCIC. Символ X на любом месте равен 10, но в сочетании «слева от старшего» (XC) X = −10, в сочетании после младшего (IX) X = +10. В непозиционных системах счисления не принято представлять дробные и отрицательные числа, действия над числами связаны с большими трудностями, поэтому используются только для наименования веков, знаменательных дат, томов, разделов и глав в книгах.

В позиционных системах счисления количественное значение символа (цифры) в числе зависит от его места (позиции или разряда). Для позиционных систем счисления характерным и определяющим является наличие основания системы, которое показывает, во-первых, во сколько раз изменяется количественное значение цифры при перемещении ее на соседнюю позицию и, во-вторых, какое число различных цифр входит в ограниченный набор, называемый алфавитом системы счисления.

Основанием системы счисления может быть любое целое число не менее 2. Наименование системы счисления соответствует ее основанию (десятичная, двоичная и т. д.) (таблица 2).

Таблица 2

Основание системы счисления (q)	Алфавит системы счисления	Число, записанное в десятичной системе счисления	То же число, представленное в другой системе счисления	Наименование системы счисления
2	0, 1	10	1010	Двоичная
3	0,1,2	10	101	Троичная
4	0,1,2,3	10	22	Четверичная
5	0,1,2,3,4	10	20	Пятеричная
10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10	10	Десятичная

В позиционных системах счисления значение одной и той же цифры зависит как от позиции, которую цифра занимает в числе, так и от системы счисления, т. е. ее основания. Например, цифра 1 в числе может иметь следующие значения (см. таблицу 3).

В десятичном числе $A_{10} = 552{,}25 = 5 \cdot 10^2 + 5 \cdot 10^1 + 2 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2}$ цифры 5 и 2, находящиеся на разных позициях, имеют различные количественные значения, при перемещении цифры на следующую позицию ее величина изменяется в 10 раз. Алфавит включает 10 цифр от 0 до 9, т. е. основание системы равно 10.

Любое число в любой позиционной системе счисления можно записать в общем виде:

$$A_q = a_{m-1}a_{m-2}\ldots a_1a_0,a_{-1}a_{-2}\ldots a_{-l},$$

или представить степенным рядом

$$A_q = a_{m-1}q^{m-1} + a_{m-2}q^{m-2} + \ldots + a_1q^1 + a_0q^0 + a_{-1}q^{-1} + a_{-2}q^{-2} + \ldots a_{-l}q^{-l}$$

Таблица 3

Название системы счисления и ее основания	Примеры чисел в разных системах счисления	Значение цифры 1 в числе, представленное в десятичной системе счисления
Двоичная	1 10 100 1000 10000 100000000	$1 \cdot 2^0 = 1$ $1 \cdot 2^1 = 2$ $1 \cdot 2^2 = 4$ $1 \cdot 2^3 = 8$ $1 \cdot 2^4 = 16$ $1 \cdot 2^8 = 256$
Троичная	1 10 100 1000	$1 \cdot 3^0 = 1$ $1 \cdot 3^1 = 3$ $1 \cdot 3^2 = 9$ $1 \cdot 3^3 = 27$
Восьмеричная	1 10 100 1000	$1 \cdot 8^0 = 1$ $1 \cdot 8^1 = 8$ $1 \cdot 8^2 = 64$ $1 \cdot 8^3 = 512$
Десятичная	1 1000	$1 \cdot 10^0 = 1$ $1 \cdot 10^3 = 1000$
Шестнадцатиричная	1 10 100	$1 \cdot 16^0 = 1$ $1 \cdot 16^1 = 16$ $1 \cdot 16^2 = 256$

или

$$A_q = \sum_{k=-l}^{m-1} a_k \cdot q^k, \qquad (1.3)$$

где q — основание системы счисления;

a_k — любая цифра из алфавита системы основания q;

m, l — число позиций (разрядов) соответственно для целой (m) и дробной частей числа (l).

Для представления чисел используется также схема Горнера:

$$A_q = A_q^{\text{ц}} + A_q^{\text{др}},$$

где

$$A_q^{\text{ц}} = \sum_{k=0}^{m-1} a_k \cdot q^k =$$

$$= (\ldots((a_{m-1}q + a_{m-2})q + a_{m-3})q + \cdots + a_1)q + a_0 \quad (1.4)$$

— целая часть числа A_q,

$$A_q^{\text{др}} = \sum_{k=-l}^{-1} a_k \cdot q^k =$$

$$= q^{-1}(a_{-1} + q^{-1}(a_{-2} + \cdots + q^{-1}(a_{-l+1} + a_{-l} \cdot q^{-1})\ldots)) \quad (1.5)$$

— дробная часть числа A_q.

В современных ЭВМ используются позиционные системы счисления с основаниями 10, 2, 8 и 16. В таблице 4 приведено соответствие чисел в этих системах. При этом круглыми скобками выделены алфавиты систем, а квадратными — их основания. Основание в любой системе изображается 10, но имеет разное количественное значение (см. также таблицу 3).

Наименьшее число цифр имеет алфавит двоичной системы (0 и 1), и она является самой простой для выполнения действий. Например:

- при сложении чисел:

$$\begin{array}{rrrrr} +0 & +0 & +1 & +1 & +11 \\ 0 & 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 1 & 10 & 100 \end{array}$$

- при вычитании чисел:

$$\begin{array}{rrrrr} -1 & -1 & -10 & -11 & -101 \\ 0 & 1 & 1 & 10 & 11 \\ \hline 1 & 0 & 1 & 1 & 10 \end{array}$$

Таблица 4

q=10	(0	1	2	3	4	5	6	7	8	9)	[10]
q=2	(0	1)	[10]	11	100	101	110	111	1000	1001	1010
q=8	(0	1	2	3	4	5	6	7)	[10]	11	12
q=16	(0	1	2	3	4	5	6	7	8	9	A

q=10	11	12	13	14	15	16
q=2	1011	1100	1101	1110	1111	10000
q=8	13	14	15	16	17	20
q=16	B	C	D	E	F)	[10]

- при умножении чисел:

$$\begin{array}{r} 1 \\ \times\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ \times\ 1 \\ \hline 0 \end{array} \qquad \begin{array}{r} 11 \\ \times\ 1 \\ \hline 11 \end{array} \qquad \begin{array}{r} 101 \\ \times\ 1 \\ \hline 101 \end{array} \qquad \begin{array}{r} 101 \\ \times\ 10 \\ \hline 1010 \end{array} \qquad \begin{array}{r} 101 \\ \times\ 0 \\ \hline 0 \end{array}$$

При умножении двоичных чисел частичные произведения множимого числа на один разряд множителя равны либо множимому числу, либо нулю.

Действия над числами с основаниями 8 и 16 непривычны и поэтому вызывают некоторые сложности. При их выполнении как и в десятичной системе счисления, при сложении чисел может образоваться единица переноса в старший разряд, если сумма цифр равна или больше основания (8 или 16). При вычитании чисел, если цифра уменьшаемого меньше цифры вычитаемого, из старшего разряда занимается одна «единица», значение которой равно основанию.

Примеры:

1) $A_8 = 750$, $B_8 = 236$. Найти $A_8 + B_8$ и $A_8 - B_8$.

$$\begin{array}{r} A_8 = 750 \\ +\ B_8 = 236 \\ \hline 1206 \end{array} \qquad \begin{array}{r} A_8 = 750 \\ -\ B_8 = 236 \\ \hline 512 \end{array}$$

2) A_{16} = B09, B_{16} = 7FA. Найти $A_{16} + B_{16}$ и $A_{16} - B_{16}$.

$$\begin{array}{r} +\ A_{16} = \text{B09} \\ \underline{B_{16} = \text{7FA}} \\ 1303 \end{array} \qquad \begin{array}{r} -\ A_{16} = \text{B09} \\ \underline{B_{16} = \text{7FA}} \\ 30\text{F} \end{array}$$

Перевод чисел из одной системы счисления в другую

При преобразовании числа из одной системы счисления в другую его количественное значение остается прежним, изменяется лишь набор символов (цифр), с помощью которых записывается число в новой системе счисления.

Перевод чисел с основаниями, являющимися степенью цифры 2. Если между основаниями p и q соблюдается связь $p^l = q^k$, где k — целое, то каждая цифра числа с основанием p представляется k цифрами алфавита основания q.

Так, если $p^1 = 8^1 = q^k = 2^3$, то каждая цифра восьмеричного числа (ее количественное значение) представляется тремя двоичными цифрами — триадами и наоборот — каждая триада (от запятой влево и вправо) двоичного числа заменяется восьмеричной цифрой. Например, цифра «1» представляется триадой «001», цифра «2» — триадой «010», цифра «7» — триадой «111».

Примеры:

1) A_8 = 4 0 7 , 1 5 A_2 = 100000111,001101

100 000 111 001 101

2) A_2 = 010 011 011 , 010 110 A_8 = 233,26

2 3 3 2 6

Если $p^l = 16^l = q^k = 2^4$, то каждая шестнадцатиричная цифра заменяется четырьмя двоичными цифрами — тетрадами и при обратном преобразовании — каждой двоичной тетраде, отсчитанной от запятой вправо и влево, ставится в соответствие одна шестнадцатиричная цифра.

Примеры:

1) $A_{16} =$ 4 F 0 E , 1 C

$0100_{1111}{}_{0000}{}_{1110}\quad 0001_{1100}$

$A_{16} = A_2 = 0100111100001110,00011100$

2) $A_2 = \underbrace{110}\,\underbrace{01101011}\,,\underbrace{111101}$

$\underbrace{0110}_{6}\ 6\ \underbrace{13}_{B}\ F\ \underbrace{0100}_{4}$

$A_{16} = 66B,F4.$

Нули перед старшим разрядом целой части и после младшего разряда дробной части можно не записывать.

Такие преобразования используются для сокращения записи двоичных чисел, при переводе чисел из десятичной системы счисления в двоичную, а также при выполнении некоторых операций в ЭВМ.

Перевод целых чисел из одной системы счисления в другую. Целое число в системе счисления q может быть представлено эквивалентным числом в системе счисления p по формуле (1.4):

$$A_q^{\text{ц}} = A_p^{\text{ц}} = (\ldots((b_{m-1}p + b_{m-2})p + b_{m-3})p + \cdots + b_1)p + b_0. \tag{1.6}$$

Задача перевода числа из одной системы счисления (q) в другую (p) заключается в отыскании значений цифр b_k числа в новой системе счисления.

Разделив обе части равенства (1.6) на основание новой системы (p), выраженное цифрами системы q,

получим:

$$\frac{A_q^{\text{ц}}}{p} = \frac{A_p^{\text{ц}}}{p} = (\ldots(b_{m-1}p + b_{m-2})p + b_{m-3} + \ldots + b_2)p + b_1 + \frac{b_0}{p}$$

или

$$A_q^{\text{ц}} = A_p^{\text{ц}} = (A_q^{\text{ц}})_1 p + b_0,$$

где $(A_q^{\text{ц}})_1$ — целое частное; b_0 — остаток, являющийся первой младшей цифрой числа в новой системе счисления, выраженный цифрами исходной системы счисления.

При следующем делении частного на основание p будут получены новое целое частное и новый остаток. $(A_q^{\text{ц}})_1 = (A_q^{\text{ц}})_2 p + b_1$, где b_1 — вторая младшая цифра числа. Продолжая деление целых частных до нулевого значения, можно найти все цифры числа в новой системе.

Правило перевода целого числа из одной системы счисления в другую:

1) последовательно делить данное число и получаемые целые частные на основание новой системы счисления, выраженные цифрами исходной системы, до тех пор, пока частное не станет равным нулю;

2) полученные остатки, являющиеся цифрами числа в новой системе счисления, выразить цифрами алфавита этой системы;

3) составить число в новой системе, начиная с последнего остатка.

Пример. Перевести числа $A_{10} = 173$ в A_2 различными способами: через восьмеричную, через шестнадцатиричную и непосредственно.

1) 173	8	5	$A_8 = 255 \rightarrow A_2 = 10101101$
21	8	5	
2	→	2↑	

2) 173	16	13 (D)	$A_{16} = AD \rightarrow A_2 =$ 10101101
10	→	10 (A) ↑	

3) 173	2	1	$A_2 = 10101101$
86	2	0	
43	2	1	
21	2	1	
10	2	0	
5	2	1	
2	2	0	
1	→	1 ↑	

Перевод десятичного числа в двоичное, если десятичное число является многозначным, целесообразно проводить через шестнадцатиричную систему счисления.

Например: $A_{10} = 175834$

175834	16	10 (A)	$A_{16} = 2AEDA$
10989	16	13 (D)	$A_2 = 101010111011011010$
686	16	14 (E)	
42	16	10 (A)	
2	→	2 ↑	

При вводе информации в ЭВМ каждая десятичная цифра заменяется ее двоичным эквивалентом в виде тетрады. Такая форма представления десятичных чисел называется двоично-десятичной.

Например: $A_{10} = 173 \rightarrow A_{2-10} = 000101110011 \neq A_2$.

В ЭВМ преобразование десятичных чисел в двоичные проводится по схеме Горнера (1.4):

$$A_2 = (0001 \cdot 1010 + 0111)1010 + 0011 = 10101101.$$

Перевод дробных чисел из одной системы счисления в другую. Дробная часть числа по схеме Горнера представляется в виде (1.5). Рассуждая по аналогии с переводом целых чисел из одной системы счисления в другую, но используя операцию умножения, формулируем правило перевода дробных чисел.

Правило перевода дробных чисел из одной системы счисления в другую:

1) последовательно умножать данное число и получаемые дробные части произведений на основание новой системы, выраженное цифрами исходной системы, до тех пор, пока дробная часть произведения не станет равной нулю;

2) полученные целые части произведений, являющиеся цифрами числа в новой системе, выразить цифрами алфавита этой системы;

3) составить дробную часть числа в новой системе, начиная с целой части первого произведения.

Пример. Перевести число $A_{10} = 0{,}65625$ в A_2 различными способами.

1) $A_{10} \to A_2$

целая часть	дробная часть	множитель
0	65625	2
1	31250	2
0	62500	2
1	25000	2
0	50000	2
1	00000	

$A_2 = 0{,}10101$;

2) $A_{10} \to A_8 \to A_2$

целая часть	дробная часть	множитель
0	65625	8
5	25000	8
2	00000	

$A_8 = 0{,}52 \to A_2 = 0{,}10101$;

3) $A_{10} \to A_{16} \to A_2$

целая часть	дробная часть	множитель
0	65625	16
(A) 10	50000	16
8	00000	

$A_{16} = 0{,}A8 \to A_2 = 0{,}10101$.

В частном случае, если знаменатель дробной части представляет целую степень цифры 2, т. е.

$$A = a/2^k,$$

то числитель «a» переводится как целое, которое записывается в «k» разрядах после запятой.

Пример.

$$A_{10} = \frac{5}{16} = \frac{5}{2^4} \rightarrow A_2 = 0{,}0101.$$

Действительно,

$$A_{10} = \frac{5}{16} = \frac{4}{16} + \frac{1}{16} = \frac{1}{4} + \frac{1}{16} = 1 \cdot 2^{-2} + 1 \cdot 2^{-4} \rightarrow$$
$$\rightarrow A_2 = 0{,}0101.$$

При переводе смешанных дробей отдельно по своим правилам переводятся целая и дробная части, результаты перевода разделяются запятой.

Перевод чисел из любой системы счисления в десятичную систему. Перевод чисел в десятичную систему может выполняться либо по правилам перевода, либо по формулам степенного ряда (1.3), либо по формулам разложения по схеме Горнера (1.4), (1.5).

1) Перевод целых чисел из одной системы счисления в другую (по правилам).

а) $A_2 = 10101101 \rightarrow A_{10}$

10101101	1010	0011(3)
10001	1010	0111(7)
1	→	1(1) ↑

$A_{10} = 173$;

б) $A_8 = 255 \rightarrow A_{10}$

255	12	3
21	12	7
1	→	1 ↑

$A_{10} = 173$;

в) $A_{16} = \mathrm{AD} \rightarrow A_{10}$

AD	A	3
11	A	7
1	→	1 ↑

$A_{10} = 173$.

Перевод дробной части числа (по правилам).

а) $A_2 = 0{,}10101 \rightarrow A_{10}$

целая часть	дробная часть	множитель
0	10101	1010
(6) 110	1001	1010
(5) 101	1010	1010
(6) 110	0100	1010
(2) 10	1000	1010
(5) 101	0000	

$A_{10} = 0{,}65625$;

б) $A_8 = 0{,}52 \rightarrow A_{10}$

целая часть	дробная часть	множитель
0	52	12
(6)	44	12
(5)	50	12
(6)	20	12
(2)	40	12
(5)	00	

$A_{10} = 0{,}65625$;

в) $A_{16} = 0{,}A8 \rightarrow A_{10}$

целая часть	дробная часть	множитель
0	A8	A
(6)	90	A
(5)	A0	A
(6)	40	A
(2)	80	A
(5)	00	

$A_{10} = 0{,}65625$.

2) Перевод целых чисел по степенному ряду.

$A_2 = 10101101 \rightarrow A_{10} = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 128 + 32 + 8 + 4 + 1 = 173$;

$A_8 = 255 \rightarrow A_{10} = 2 \cdot 8^2 + 5 \cdot 8^1 + 5 \cdot 8^0 = 128 + 40 + 5 = 173$;

$A_{16} = AD \rightarrow A_{10} = 10 \cdot 16^1 + 13 \cdot 16^0 = 173$.

Перевод дробной части числа по степенному ряду.

$A_2 = 0{,}10101 \to A_{10} = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0\ 2^{-4} + 1 \cdot 2^{-5} = 0{,}5 + 0{,}125 + 0{,}03125 = 0{,}65625;$

$A_8 = 0{,}52 \to A_{10} = 5 \cdot 8^{-1} + 2 \cdot 8^{-2} = 0{,}625 + 0{,}03125 = 0{,}65625.$

$A_{16} = 0{,}A8 \to A_{10} = 10 \cdot 16^{-1} + 8 \cdot 16^{-2} = 0{,}625 + 0{,}03125 = 0{,}65625.$

3) Перевод целых чисел по схеме Горнера.

$A_2 = 10101101 \to A_{10} = ((((((1 \cdot 2 + 0)2 + 1)2 + 0)2 + 1)2 + 1)2 + 0)2 + 1 = 173;$

$A_8 = 255 \to A_{10} = (2 \cdot 8 + 5)8 + 5 = 168 + 5 = 173;$

$A_{16} = AD \to A_{10} = 10 \cdot 16 + 13 = 173.$

Перевод дробной части числа по схеме Горнера.

$$A_2 = 0{,}10101 \to A_{10} = \frac{1}{2}\left(1 + \frac{1}{2}\left(0 + \frac{1}{2}\left(1 + \frac{1}{2}\left(0 + \frac{1}{2}\right)\right)\right)\right) = \frac{21}{32} = 0{,}65625;$$

$$A_8 = 0{,}52 \to A_{10} = \frac{1}{8}\left(5 + 2 \cdot \frac{1}{8}\right) = \frac{21}{32} = 0{,}65625;$$

$$A_{16} = 0{,}A8 \to A_{10} = \frac{1}{16}\left(10 + 8 \cdot \frac{1}{16}\right) = \frac{21}{32} = 0{,}65625.$$

Из анализа способов перевода чисел из разных систем счисления в десятичную следует, что перевод чисел с использованием рассмотренных методов из восьмеричной и шестнадцатиричной систем счисления в десятичную затруднителен, так как требует знания восьмеричной и шестнадцатиричной таблиц умножения.

Перевод по степенному ряду многоразрядных чисел требует знания или вычислений степеней основания, а для перевода по форме Горнера умножение производится только на первую или минус первую степени основания.

Кодирование информации в ЭВМ

Компьютеры могут обрабатывать информацию, представленную только в числовой форме. При вводе информации в память компьютера каждый символ-буква русского или латинского алфавита, цифра, знак пунктуации или знак действия — кодируется определенной последовательностью двоичных цифр. Это происходит в соответствии с таблицами кодирования. Существует несколько разных таблиц кодирования.

Для персональных компьютеров, совместимых с ЭВМ IBM PC и работающих под управлением операционной системы MS DOS, применяется русифицированная альтернативная таблица кодирования ASCII (таблица 5), а для среды Windows — русифицированная таблица ANSI.

Каждая таблица включает 16 строк и 16 столбцов с шестнадцатиричными номерами от 0 до F (двоичные от 0000 до 1111). Таблицы позволяют закодировать до 256 (16 × 16) символов. Код символа составляется из номера строки, к которому приписывается номер столбца, на пересечении которых записан символ. Например, латинская буква L имеет код 4C или 01001100, а русская буква Л — код 8B или 10001011, т. е. каждый символ, вводимый в компьютер и хранящийся в памяти, представляется байтом (слогом).

Каждая таблица разделена на две части по 128 символов. Первая часть (основная) с номерами строк от 0 до 7 включает символы латинского алфавита, десятичных цифр, знаков пунктуации, арифметических действий и др., имеющихся на клавиатуре с латинским шрифтом. Эта часть в основном одинакова для разных таблиц кодирования. Вторая часть представляет собой таблицу расширения (дополнения) с номерами строк от 8 до F и предназначена главным образом для кодирования национальных алфавитов. Эти части имеют различия. Так, во второй части таблицы ASCII симво-

лы русского алфавита располагают в строках с номерами 8, 9, A, E, а в таблице ANSI — в строках от C до F. Русская буква А в этой таблице имеет другой код — CB или 11001011. Поэтому в среде Windows имеются средства перекодировки текстов, закодированных в ASCII, в систему ANSI.

Десятичные цифры в этих таблицах кодирования находятся в третьей строке, при этом значение каждой цифры соответствует номеру столбца (см. таблицу 5). Например, число $A_{10} = 173$ при вводе в ЭВМ будет закодировано тремя байтами

00110001 00110111 00110011.

Для хранения в памяти и выполнения действий этот код сначала «упаковывается» — отбрасываются первые тетрады из каждого байта и образуется число в двоично-десятичной форме — $A_{2-10} = 000101110011$, которое далее по формуле Горнера (1.4) преобразуется в двоичный код $A_2 = 10101101$ (см. стр. 33).

Замечание. Существуют и другие таблицы кодирования с более широкими информационными возможностями, например, использующими двухбайтовую кодировку на 65536 символов.

Формы и форматы представления числовых данных в ЭВМ

Каждый разряд двоичного числа (бит) представляется в ЭВМ физическим элементом, обладающим двумя устойчивыми состояниями, одному из которых приписывается значение 0, а другому 1. Совокупность определенного количества этих элементов служит для представления многоразрядных двоичных чисел и составляет разрядную сетку или формат представления числовых данных.

Таблица 5. Стандартная для России таблица кодов ASCII (позиции с 00 по 1F заняты управляющими символами)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
6	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	127

Продолжение таблицы 5

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	А 128	Б 129	В 130	Г 131	Д 132	Е 133	Ж 134	З 135	И 136	Й 137	К 138	Л 139	М 140	Н 141	О 142	П 143
9	Р 144	С 145	Т 146	У 147	Ф 148	Х 149	Ц 150	Ч 151	Ш 152	Щ 153	Ъ 154	Ы 155	Ь 156	Э 157	Ю 158	Я 159
A	а 160	б 161	в 162	г 163	д 164	е 165	ж 166	з 167	и 168	й 169	к 170	л 171	м 172	н 173	о 174	п 175
B	▒ 176	▒ 177	▓ 178	┃ 179	┤ 180	╡ 181	╢ 182	╖ 183	╕ 184	╣ 185	║ 186	╗ 187	╝ 188	╜ 189	╛ 190	┐ 191
C	└ 192	┴ 193	┬ 194	├ 195	─ 196	┼ 197	╞ 198	╟ 199	╚ 200	╔ 201	╩ 202	╦ 203	╠ 204	═ 205	╬ 206	╧ 207
D	╨ 208	╤ 209	╥ 210	╙ 211	╘ 212	╒ 213	╓ 214	╫ 215	╪ 216	┘ 217	┌ 218	▉ 219	▄ 220	▌ 221	▐ 222	▀ 223
E	р 224	с 225	т 226	у 227	ф 228	х 229	ц 230	ч 231	ш 232	щ 233	ъ 234	ы 235	ь 236	э 237	ю 238	я 239
F	Ё 240	ё 241	≥ 242	≤ 243	⌠ 244	⌡ 245	÷ 246	≈ 247	° 248	• 249	· 250	√ 251	n 252	^{2} 253	■ 254	255

В ЭВМ, как и в математике, используются как «естественная», так и «нормальная» формы записи чисел.

Каждая из форм имеет разные форматы для каждого типа ЭВМ, составленные из целого количества «байт». Длину формата данных измеряют в машинных словах или в количестве двоичных разрядов (бит). Например, в ЭВМ ЕС используются такие форматы:

полуслово — 2 байта (16 бит),

слово — 4 байта (32 бит),

двойное слово — 8 байт (64 бит).

В ПЭВМ:

слово — 2 байта (4 байта),

двойное слово — 4 байта (8 байт).

Естественная форма представления числа в памяти ЭВМ. Естественную форму обычно называют представлением данных (чисел) с фиксированной запятой или точкой, положение которой строго устанавливается для правильных дробей — перед старшим разрядом, для смешанных дробей — в определенном месте, отделяющим целую и дробные части числа, для целых чисел — после младшего разряда. В современных ЭВМ естественная форма используется в основном для представления целых чисел.

Во всех форматах знак числа занимает одно место перед старшим разрядом и кодируется: 0 — знак «плюс» и 1 — знак «минус».

Для удобства описания форматы представления числа в памяти ЭВМ обозначают латинскими буквами: «*H*», «*F*», «*E*».

Рассмотрим диапазон представления чисел в коротком формате — $H = 2$ байта и в длинном — $F = 4$ байта. В разрядных сетках указаны коды наименьшего и наибольшего значения чисел.

Числа в формате H могут иметь значения:

а) Представление наименьшего целого числа в естественной форме (формат H)

Знак	2^{14}	2^{13}		2^1	2^0
0	0	0	...	0	1
0	1	2	...	14	15

$|A|_{min} = 1;$

б) Представление наибольшего целого числа в естественной форме (формат H)

Знак	2^{14}	2^{13}		2^1	2^0
0	1	1	...	1	1
0	1	2	...	14	15

$|A|_{max} = 2^{15} - 1 \approx 2^{15} \approx 2^{10} \cdot 2^5 \approx 10^3 \cdot 2^5 = 32 \cdot 10^3.$

Числа в формате F имеют значения:

а) Представление наименьшего целого числа в естественной форме (формат F)

Знак	2^{30}	2^{29}		2^1	2^0
0	0	0	...	0	1
0	1	2	...	30	31

$|A|_{min} = 1;$

б) Представление наибольшего целого числа в естественной форме (формат F)

Знак	2^{30}	2^{29}		2^1	2^0
0	1	1	...	1	1
0	1	2	...	30	31

$|A|_{max} = 2^{31} - 1 \approx 2^{31} \approx (2^{10})^3 \cdot 2 \approx 2 \cdot 10^9.$

При представлении правильных дробей, например, в формате H, наименьшее и наибольшее значения определяются

$|A|_{min} = 0{,}00\ldots01 = 1 \cdot 2^{-15};$

$|A|_{max} = 0{,}11\ldots1 = 1 - 1 \cdot 2^{-15}.$

Для сокращения записи двоичных чисел можно использовать шестнадцатиричную систему. Так, в формате H:

$A_{min} = 0001;$

$A_{max} = 7FFF;$

в формате F:

$A_{min} = 00000001$;

$A_{max} = 7FFFFFFF$.

Каждая шестнадцатиричная цифра представляет собой двоичную тетраду.

Пример. Числа $A = 173$, $B = -173$ в форматах H и F:

$A_2^H = 0000000010101101$;

$B_2^H = 1000000010101101$;

$A_{16}^H = 00AD$; $A_{16}^F = 000000AD$;

$B_{16}^H = 80AD$; $B_{16}^F = 800000AD$.

По первой шестнадцатиричной цифре можно определить знак числа: если первая цифра меньше 8, то число положительное, если ее значение от 8 до F, то отрицательное.

Пример. $A_{16}^H = 510E > 0$;

$A_{16}^H = A74E < 0$.

Достоинствами естественной формы являются простота и наглядность представления чисел, простота алгоритмов реализации операций, а следовательно, простота устройств и высокая скорость выполнения операций. Существенным недостатком является ограниченный диапазон значений величин. Если результаты вычислений выходят за допустимые пределы значений, то наступает переполнение разрядной сетки и результат искажается. В больших ЭВМ при этом вырабатывается запрос на прерывание программы, а в персональных производится автоматический переход к представлению чисел в нормальной форме.

Нормальная форма представления чисел в памяти ЭВМ. Число можно представить в различной форме записи, например:

$$A = 55{,}25 = 5525 \cdot 10^{-2} = 0{,}5525 \cdot 10^{2} = 0{,}005525 \cdot 10^{4}.$$

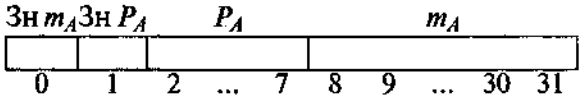


Схема 8. Формат числа в нормальной форме

Любое число в нормальной форме представляется в виде:

$$A = \pm m_A \cdot q^{\pm P_A}, \qquad (1.7)$$

где m_A — мантисса числа A,

q — основание системы счисления,

P_A — порядок.

Для однозначности представления чисел используется нормализованная форма, при которой мантисса должна отвечать условию:

$$\frac{1}{q} \leqslant |m| < 1.$$

Ограничение справа требует, чтобы мантисса представлялась правильной дробью, ограничение слева — чтобы после запятой присутствовала значащая цифра (не 0).

Нормальную форму называют также полулогарифмической или с плавающей запятой, положение которой определяется порядком, а также экспоненциальной.

Для представления чисел в нормальной форме используются фиксированные форматы разной длины. В разрядной сетке форматов отводятся места для знака мантиссы (нулевой разряд), знака порядка (первый разряд), значения порядка (6 разрядов, со 2-го по 7-й), в остальные разряды записывается мантисса числа. В других форматах первый байт не изменяется, а увеличивается область под мантиссу. На схеме 8 представлена разрядная сетка в формате 4 байта.

Диапазон представления чисел можно оценить по максимальному значению:

$$A_{max} = m_{max} \cdot q^{P_{max}},$$

где $m_{max} = 1-2^{-24}$,

$P_{max} = 2^6-1 = 63.$

Например, при $q = 2$:

$$A_{max} = (1-2^{-24}) \cdot 2^{63} \approx 1 \cdot 2^{63} =$$

$$= (2^{10})^6 \cdot 2^3 \approx (10^3)^6 \cdot 2^3 \approx 10^{19}.$$

По сравнению с естественной формой, диапазон представления чисел в нормальной форме при той же разрядной сетке увеличился на 10 порядков. Для примера рассмотрим форматы представления числа в ЭВМ ЕС: короткий E (4 байта), длинный D (8 байт) и повышенной точности (16 байт).

Особенностями нормальной формы в ЭВМ ЕС являются следующие:

1. Смещение числовой оси порядков в область положительных значений для облегчения действий над порядками, не имеющими знака. Обычно 7 разрядов (схема 8) отводится под значение порядка и его знак.

Следовательно, числовая ось порядков находится в диапазоне $-2^6 \leqslant P \leqslant 2^6-1$ или $-64 \leqslant P \leqslant 63$ (P — порядок числа).

Смещенный порядок (называемый характеристикой) определяется его смещением на $+2^6 = 64_{10} = 40_{16}$. В этом случае характеристика $P_x = P + 40$ не имеет знака.

Теперь характеристика может принимать значения в диапазоне

$$0 \leqslant P_x \leqslant 127 = 7F_{16}$$

и под ее значение, как уже было сказано, отводятся 7 разрядов (максимальное значение порядка $2^7-1 = 127$). Очевидно, если $P_x = 40$, то $P = 0$, если $P_x < 40$,

то порядок отрицательный $P < 0$, при $P_x > 40$ — порядок положительный $P > 0$. Если $P_x < 0$ или $P_x > 7F$, то значение характеристики пропадает и результаты искажаются.

2. Мантиссы и порядки чисел выражаются в шестнадцатиричной системе счисления в двоичном виде, что обеспечивает увеличение диапазона представления чисел, так как изменение характеристики на 1 приводит к сдвигу мантиссы на одну шестнадцатиричную цифру, т. е. сразу на одну двоичную тетраду. Действительно, если в формулу (1.7) подставить $q = 16$, то

$$A_{max} = (1-16^{-6}) \cdot 16^{63} \approx 1 \cdot (2^4)^{63} \approx (10^{19})^4 = 10^{76}.$$

Таким образом, значение порядка увеличилось в 4 раза.

Представим в разрядной сетке формата E два числа (знак $\sim$ обозначает равенство чисел в разных системах счисления):

$$A_{10} = 32008{,}5 \sim A_{16} = 7D08{,}8$$

и

$$B_{10} = -32008{,}5 \sim B_{16} = -7D08{,}8.$$

Для этого найдем нормализованные мантиссы и характеристики:

$m_A = 0{,}7D08{,}8$, $P_{XA} = 40 + 4 = 44$;

$m_B = -0{,}7D08{,}8$, $P_{XB} = 40 + 4 = 44$ (схема 9).

Здесь так же, как и при естественной форме хранения числа, его знак определяется по первой шестнадцатиричной цифре кода.

Таким образом две первые шестнадцатиричные цифры кода числа с плавающей запятой определяют характеристику P_x с учетом знака числа (0 или 1 в старшем разряде первой двоичной тетрады). Для положительных чисел две первые шестнадцатиричные цифры

Знак	Характеристика	Дробная часть мантиссы			
0	100 0100	0111 1101	0000 1000	1000 0000	
	44	7D	08	80	
1	100 0100	0111 1101	0000 1000	1000 0000	
	C4	7D	08	80	

Схема 9. Представление чисел A и B в формате E и их коды

Знак	Характеристика	Дробная часть мантиссы			
0	011 1111	0100 0000	0100 0000	0100 0000	
	3F	40	00	00	
1	011 1111	0100 0000	0100 0000	0100 0000	
	BF	40	00	00	

Схема 10. Представление чисел C и D в разрядной сетке и их коды

кода числа образуются как сумма $P_x + 00_{16}$, а для отрицательных — $P_x + 80_{16}$.

Представим в разрядной сетке (см. схему 10) два других числа: $C_{10} = 0{,}015625 \sim C_{16} = 0{,}04$

и $D_{10} = -0{,}015625 \sim D_{16} = -0{,}04$.

Для этого найдем нормализованные мантиссы, порядки и характеристики этих чисел:

$m_C = 0{,}4$; $P_C = -1$; $P_{XC} = 40-1 = 3F$; с учетом знака $P_{XC} + 00_{16} = 3F$;

$m_D = -0{,}4$; $P_D = -1$; $P_{XD} = 40-1 = 3F$; с учетом знака $P_{XD} + 80_{16} = BF$.

По шестнадцатиричному коду числа с плавающей запятой нетрудно определить и само десятичное число. Рассмотрим примеры кодов чисел, представленных на схемах 9 и 10.

Код числа $A_{16} = 447D0880$.

Характеристика с учетом знака = 44, первая цифра 4 < 8. Поэтому, число положительное, характеристика $P_{XA} = 44_{16}-00_{16} = 44_{16}$, порядок числа $P_A =$

$= 44_{16} - 40_{16} = 4_{16}$, а мантисса числа $m_A = 0{,}7D088_{16}$. Таким образом, число $A_{16} = m_A \cdot 16^{P_A} = 0{,}7D088 \cdot 16^4 = 7D08{,}8$. Следовательно,

$$A_{10} = 7 \cdot 16^3 + 13 \cdot 16^2 + 8 \cdot 16^0 + 8 \cdot 16^{-1} = 28672 + 3328 + 8 + 0{,}5 = 32008{,}5.$$

Код числа $B_{16} = C47D0880$.

Характеристика с учетом знака $= C4$, первая цифра $C > 7$. Поэтому, число отрицательное, характеристика $P_{XB} = C4_{16} - 80_{16} = 44_{16}$, порядок числа $P_B = 44_{16} - 40_{16} = 4_{16}$, а мантисса числа $m_B = -0{,}7D088_{16}$. Вычисления аналогичны, следовательно, $B_{16} = -7D08{,}8$, а $B_{10} = -32008{,}5$.

Код числа $C_{16} = 3F4400000$.

Характеристика с учетом знака $= 3F$, первая цифра $3 < 8$. Поэтому, число положительное, характеристика $P_{XC} = 3F_{16} - 00_{16} = 3F_{16}$, порядок числа $P_C = 3F_{16} - 40_{16} = -1_{16}$, а мантисса числа $m_C = 0{,}4_{16}$. Таким образом, число $C_{16} = m_C \cdot 16^{P_C} = 0{,}4 \cdot 16^{-1} = 0{,}04_{16}$. Следовательно, $C_{10} = 4 \cdot 16^{-2} = \frac{4}{256} \frac{1}{64} = 0{,}015625$.

Код числа $D_{16} = BF4400000$.

Характеристика с учетом знака $= BF$, первая цифра $B > 7$. Поэтому, число отрицательное, характеристика $P_{XD} = BF_{16} - 80_{16} = 3F_{16}$, порядок числа $P_D = 3F_{16} - 40_{16} = -1_{16}$, а мантисса числа $m_D = -0{,}4_{16}$. Таким образом, число $D_{16} = m_D \cdot 16^{P_D} = -0{,}4 \cdot 16^{-1} = -0{,}04_{16}$. Следовательно, $D_{10} = -4 \cdot 16^{-2} = -\frac{4}{256} = -\frac{1}{64} = -0{,}015625$.

Особенности представления чисел с плавающей запятой в ПЭВМ. В некоторых мини- и микроЭВМ также используется беззнаковый порядок, смещенный на $2^7 = 128 = 80_{16}$, который меняется в диапазоне $0 \leqslant P_X \leqslant 255 = FF$. В разрядной сетке под порядок отводится 8 двоичных разрядов, под мантиссу — 23.

Мантисса представляется в двоичной системе, изменение порядка на 1 приводит к смещению мантиссы влево или вправо на один двоичный разряд.

Поэтому числа в этих машинах при $P_{max} = 127$ представляются в диапазоне

$$A_{max} \leqslant \left(1-2^{-23}\right) 2^{127} \approx \left(2^{10}\right)^{12} \cdot 2^{7} \approx \left(10^{3}\right)^{12} \cdot 10^{2} = 10^{38}.$$

Отметим, что точность представления чисел определяется количеством двоичных разрядов, отводимых под мантиссу числа. Может сложиться мнение, что по сравнению с шестнадцатиричной системой нормализации тетрадами, когда под мантиссу в коротком формате отводится 24 разряда, использование двоичной системы нормализации с 23 разрядами для мантиссы не только сокращает диапазон представления чисел, но и уменьшает точность их представления. Однако это не так. Дело в том, что при двоичной системе в старшем разряде дробной части нормализованной мантиссы всегда должна быть 1. Эта единица в разрядную сетку не заносится, но она подразумевается. При выполнении действий над числами в сумматоре или при выводе результатов она автоматически учитывается.

Рассмотрим представление в разрядной сетке (см. схему 11) чисел A и D из предыдущих примеров для случая использования двоичной системы нормализации со смещением на 128.

Число $A = 32008{,}5_{10} = 111110100001000{,}1_2$;

$m_A = 0{,}1111101000010001_2$;

$P_{XA} = 128_{10} + 15_{16} = 143_{16} = 10001111_2$.

Число $D = -0{,}015625_{10} = -0{,}000001_2$;

$m_D = -0{,}1_2$;

$P_{XD} = 128_{10} - 5_{10} = 123_{10} = 01111011_2$.

Следует отметить, что шестнадцатиричные коды чисел при использовании двоичной системы норма-

Знак	Характеристика	Дробная часть мантиссы		
0	1000 1111	111 1010	0001 0001	0000 0000
	47FA		11	00
1	0111 1011	000 0000	0000 0000	0000 0000
	BD80		00	00

Схема 11. Представление чисел A и D в разрядной сетке при использовании двоичной системы нормализации со смещением на 128

лизации со смещением на 128 отличны от кодов этих же чисел, представленных на схемах 9 и 10.

Машинные коды чисел и действия над ними

Сущность и назначение машинных кодов. В ЭВМ посредством применения специальных машинных кодов все арифметические операции над числами сводятся к выполнению операции арифметического сложения и сдвигу их кодов вправо или влево. При этом учитываются знаки чисел, автоматически определяются знак результата и признаки возможного переполнения разрядной сетки заданных форматов. Применяются прямой, обратный и дополнительный коды. Замена операции вычитания на сложение может осуществляться с помощью обратного и дополнительного кодов. Сущность этого процесса заключается в том, что вычитаемое B, как отрицательное число, представляется в виде дополнения до некоторой константы K, при которой выполняется условие $K-B>0$. Обратный и дополнительный коды отличаются выбором значения константы K. Следовательно, операцию $C=A-B$, где A и B целые положительные числа в любой системе счисления, можно представить в виде:

$$C=A-B=A+(-B)=A+(10^n-B)-10^n=$$
$$=A+(10^n-1-B)-10^n+1, \quad (1.8)$$

где 10 — основание любой системы счисления.

$K = 10^n$ — константа образования дополнительного кода; $K = 10^n - 1$ — константа образования обратного кода; n — количество разрядов представления целых чисел в выбранной системе счисления (для дробных чисел $n = 0$).

Из выражения (1.8) следует, что из полученной суммы нужно исключить добавленную к вычитаемому константу.

Рассмотрим примеры действий над числами.

Пример 1. $A = 59$, $B = 34$.

Найти $C = A - B$.

Пусть константа для дополнительного кода $10^2 = 100$, для обратного кода $10^2 - 1 = 99$.

Тогда $(-B)_{доп} = 100 - 34 = 66$,

$(-B)_{обр} = 99 - 34 = 65$.

$$\begin{array}{r} +A = 59 \\ \underline{(-B)_{доп} = 66} \\ 125 - 10^2 = 25 = C; \end{array}$$

$$\begin{array}{r} A = 59 \\ \underline{(-B)_{обр} = 65} \\ 124 - 10^2 = 24 \\ \underline{+\quad\quad 1} \\ 25 = C. \end{array}$$

Из примера 1 следует, что операция вычитания заменяется операцией сложения с дополнениями, при этом из полученной суммы константа дополнительного кода компенсируется просто ликвидацией 1 переноса из старшего разряда суммы, а константа обратного кода компенсируется путем исключения единицы переноса и добавления ее к младшему разряду суммы, т. е. требуется дополнитсльная операция сложения. Поэтому в ЭВМ для выполнения действий используется дополнительный код, а обратный код используется для образования дополнительного кода.

По результатам действий под числами с дополнениями легко определить знак суммы и наличие переполнения разрядной сетки (сумма равна или больше по модулю константы образования дополнительного кода).

Пример 2. $A = 59$, $B = 34$. Найти:

а) $C_1 = B - A$, $\quad (-A)_{доп} = 10^2 - 59 = 41$

б) $C_2 = A + B$, $\quad (-B)_{доп} = 10^2 - 34 = 66$

в) $C_3 = -A - B$.

а) $\begin{array}{r} B = 34 \\ +\ (-A)_{доп} = 41 \\ \hline (C_1)_{доп} = 75 \end{array}$ $\quad C_1 = 75 - 10^2 = -25;$

б) $\begin{array}{r} A = 59 \\ +\ B = 34 \\ \hline C_2 = 93 \end{array}$ $\quad C_2 = 93;$

в) $\begin{array}{r} (-A)_{доп} = 41 \\ +\ (-B)_{доп} = 66 \\ \hline 107 \end{array}$ $\quad C_3 = 07 - 10^2 = -93$

$107 - 10^2 = 07 = (C_3)_{доп}.$

Из примеров 1 и 2 следует, что при сложении чисел с разными знаками (C и C_1) единица переноса из старшего разряда суммы является признаком положительного результата (C), отсутствие переноса (C_1) — признаком отрицательного результата, при этом константа образования дополнительного кода нескомпенсирована и осталась в сумме.

При сложении чисел с одинаковыми знаками признаки противоположны: отсутствие переноса единицы из старшего разряда при положительных слагаемых (C_2) является признаком положительного результата, наличие переноса при отрицательных слагаемых (C_3) — признаком отрицательного, при этом одна константа компенсируется переносом, а вторая сохраняется в сумме. Эти же признаки в суммах (C_2 и C_3) указывают на отсутствие переполнения разрядной сетки

для записи результатов (результаты имеют 2 десятичных разряда).

Пример 3. $A = 59$, $B = 65$. Найти $C_4 = A + B$, $C_5 = -A - B$.

$(-A)_{доп} = 10^2 - 59 = 41$, $(-B)_{доп} = 10^2 - 65 = 35$.

$$\begin{array}{rr} A = 59 & (-A)_{доп} = 41 \\ +\,B = 65 & +\,(-B)_{доп} = 35 \\ \hline C_4 = 124 > 10^2 & C_5 = 76 > |10^2|. \end{array}$$

Здесь в обоих случаях при сложении чисел с одинаковыми знаками происходит переполнение разрядной сетки, признаками которого являются: наличие переноса из старшего разряда при положительных слагаемых (C_4) и отсутствие переноса при отрицательных (C_5). При получении суммы C_5 перенос отсутствует, т. е. обе константы 10^2 остались в полученном результате. Следовательно, результат $76 - 10^2 = -24$, $-24 - 10^2 = -124$ по модулю больше константы и в отведенные 2 разряда не умещается.

Рассмотрим образование кодов в двоичной системе счисления на примере $A = \pm 34$. Знак числа, как было указано выше, кодируется 0 или 1, записывается перед старшим разрядом и отделяется для наглядности точкой, которая не является частью кода и в разрядной сетке не отражается. Для простоты примем, что задана разрядная сетка в один байт, т. е. 8 двоичных разрядов, из которых один отводится под знак, а 7 для записи числа:

$$A_{10} = \pm 34 = A_{16} = \pm 22 = A_2 = \pm 100010.$$

Примем константу для дополнительного кода: $K_{10} = 2^8 = K_2 = 10^{1000} = 100000000$

и константу для обратного кода: $K_{10} = 2^8 - 1 = K_2 = 10^{1000} - 1 = 11111111$.

В таблице 6 приведены прямой, обратный и дополнительный коды для чисел $A = 34$ и $A = -34$.

Таблица 6

Число	[A]пк	[A]ок	[A]дк
100010	00100010	00100010	00100010
		11111111	100000000
		⁻00100010	⁻ 00100010
−100010	10100010	11011101	11011110

Правила образования машинных кодов. *1) прямой код положительного и отрицательного чисел отличается только знаковыми разрядами, модуль числа не изменяется;*

2) положительное число в прямом, обратном и дополнительном кодах имеет одинаковое изображение;

3) обратный код отрицательного двоичного числа образуется из прямого кода положительного числа путем замены всех единиц на нули, а нулей на единицы, включая знаковый разряд;

4) дополнительный код отрицательного числа образуется путем добавления единицы к младшему разряду обратного кода этого же числа или заменой в коде положительного числа всех нулей на единицы, а единиц на нули, исключая последнюю единицу и следующие за ней нули.

Определение прямого кода отрицательного числа по его обратному и дополнительному коду производится по тем же правилам.

Числа, представленные в естественной форме, в памяти ЭВМ представляются в дополнительном коде, числа в нормальной форме хранятся в прямом коде. Действия в ЭВМ выполняются в прямом и дополнительном кодах, обратный код используется для получения дополнительного кода.

Действия над машинными кодами чисел. *А. Действия над числами, представленными в естественной форме*

При сложении кодов чисел в естественной форме следует учитывать следующие положения:

1) числа хранятся в памяти в дополнительном коде;

2) в сумматоре числа складываются вместе со знаками, при этом образуется знак результата;

3) при сложении чисел с разными знаками единица переноса из знакового разряда стирается, т. е. компенсируется одна константа образования дополнительного кода;

4) признаками переполнения разрядной сетки при сложении кодов чисел с одинаковыми знаками могут служить:

а) знак суммы не соответствует знакам слагаемых;

б) переносы из старшего разряда суммы в знаковый и из знакового не согласуются, т. е. один из них присутствует, а другой отсутствует.

Очевидно, что если переносы согласуются, т. е. оба отсутствуют или оба присутствуют, то переполнения разрядной сетки не происходит.

Пример 4. Даны два числа: $A = 254$, $B = 175$.

Найти сумму чисел при разных знаках слагаемых в 16-ти разрядном формате.

Решение.

а) Представим исходные числа в двоичной системе счисления:

$A_{10} = 254 \sim A_{16} = \text{FE} \sim A_2 = 11111110$;

$B_{10} = 175 \sim B_{16} = \text{AF} \sim B_2 = 10101111$.

б) Составим машинные коды этих чисел с разными знаками:

$[\text{A}]_{\text{пк}} = 0.000000011111110$;

$[\text{B}]_{\text{пк}} = 0.000000010101111$.

$[-\text{A}]_{\text{дк}} = 1.111111100000010$;

$[-\text{B}]_{\text{дк}} = 1.111111101010001$.

в) Выполним действия:

$$C_1 = A + B$$

$$\begin{array}{l} [A]_{пк} = 0.0000000111111110 \\ +\ [B]_{пк} = 0.0000000101011111 \\ \hline [C_1]_{пк} = 0.0000001101011101 > 0; \end{array}$$

$$C_2 = A - B$$

$$\begin{array}{l} [A]_{пк} = 0.0000000111111110 \\ +\ [-B]_{дк} = 1.1111111010100001 \\ \hline [C_2]_{пк} = 10.0000000100111111 > 0; \end{array}$$

$$C_3 = B - A$$

$$\begin{array}{l} [B]_{пк} = 0.0000000101011111 \\ +\ [-A]_{дк} = 1.1111111000000010 \\ \hline [C_3]_{пк} = 1.1111111101100001 < 0; \end{array}$$

$$C_4 = -A - B$$

$$\begin{array}{l} [-A]_{дк} = 1.1111111000000010 \\ +\ [-B]_{дк} = 1.1111111010100001 \\ \hline [C_4]_{пк} = 11.1111110010100011 < 0. \end{array}$$

Из примера следует:

1) при получении сумм слагаемых с одинаковыми знаками (C_1 и C_4) переполнения разрядной сетки не произошло, так как знак суммы соответствует знакам слагаемых и переносы в знаковый и из знакового согласуются;

2) при получении сумм C_2 и C_4 образовался перенос из знакового разряда, который следует исключить;

3) суммы C_1 и C_2 — положительные, C_3 и C_4 — отрицательные, т. е. знаки результатов получены при сложении чисел со знаками.

Полученные суммы заносятся в разрядную сетку памяти в дополнительном коде, т. е. без изменения.

Проверка. Для этого следует перевести полученные суммы любым способом в десятичную систему и

сравнить с заданием:

$$C_1 = 110101101 =$$
$$= 1 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 =$$
$$= 256 + 128 + 32 + 8 + 4 + 1 = 429;$$
$$C_1 = 254 + 175 = 429.$$

Пример 5. Дано $A = 25756$, $B = 7125$.

Найти суммы положительных и отрицательных значений в 16-ти разрядном формате.

Решение.

$A_{10} = 25756 \sim A_{16} = 649C \sim A_2 = 110010010011100;$
$B_{10} = 7125 \sim B_{16} = 1BD5 \sim B_2 = 1101111010101.$

$$C_1 = A + B$$

$$\begin{array}{l} [A]_{пк} = 0.110010010011100 \\ + \quad [B]_{пк} = 0.001101111010101 \\ \hline [C_1]_{пк} \neq 1.000000001110001 < 0; \end{array}$$

$$C_2 = -A - B$$

$$\begin{array}{l} [A]_{дк} = 1.001101101100100 \\ + \quad [B]_{дк} = 1.110010000101011 \\ \hline [C_2]_{дк} = 10.111111110001111 > 0. \end{array}$$

Полученные суммы не соответствуют ожидаемым результатам, поскольку произошло переполнение разрядной сетки за допустимые значения (32767). Переполнение разрядной сетки легко определить либо по первому признаку — знаки сумм C_1 и C_2 не соответствуют знакам слагаемых, либо по второму — переносы в знаковый и из знакового разряда в суммах C_1 и C_2 не согласуются. В этих случаях в больших ЭВМ вырабатывается запрос на прерывание программы, в некоторых типах малых ЭВМ производится автоматический переход к нормальной форме представления данных.

Замечание. Операции умножения и деления производятся над абсолютными значениями чисел — в прямом коде. Знак произведения определяется сложением по модулю 2 знаков сомножителей ($0+0=0$, $1+0=1$, $1+1=0$), знак частного — сложением по модулю 2 знаков делимого и делителя, а знаку остатка присваивается знак делимого.

Б. Действия над числами, представленными в нормальной форме

При алгебраическом сложении чисел, представленных в нормальной форме, необходимо учитывать:

1. Числа в нормальной форме хранятся в памяти в прямом коде с нормализованными мантиссами.

2. Сложение кодов чисел производится путем сложения мантисс только при одинаковых порядках (характеристиках) слагаемых. За общий выбирается наибольший порядок. Выравнивание порядков слагаемых осуществляется изменением мантиссы меньшего числа.

Пусть $A = m_A \cdot q^{P_A}$, $B = m_B \cdot q^{P_B}$ и $P_A > P_B$, тогда разность порядков $\Delta P = P_A - P_B$ и $P_B = P_A - \Delta P$.

$$C = A + B = m_A \cdot q^{P_A} + m_B \cdot q^{P_B} = m_A \cdot q^{P_A} + m_B \cdot q^{P_A - \Delta P} =$$
$$= m_A \cdot q^{P_A} + m_B \cdot q^{-\Delta P} \cdot q^{P_A} = (m_A + m_B^*)\, q^{P_A},$$

где $m_B^* = m_B \cdot q^{-\Delta P}$ — мантисса, приведенная к большему порядку путем ее сдвига вправо на ΔP разрядов основания q.

3. При сложении мантисс с одинаковыми знаками возможно переполнение разрядной сетки, что является признаком нарушения нормализации.

4. Результаты в прямом коде нормализуются.

5. Действия в сумматоре выполняются только над кодами мантисс, которые поступают из регистров слагаемых в младшие 24 разряда сумматора, знаки мантисс и значения характеристик заносятся в специ-

альные схемы, которые обеспечивают выравнивание характеристик, нормализацию мантиссы результата, формирование знака и характеристики суммы. В старшие разряды (0–7) сумматора вводятся нули.

6. Алгоритмы операции алгебраического сложения после выравнивания характеристик зависят от знаков слагаемых:

а) если знаки слагаемых одинаковы (положительные или отрицательные), то модули мантисс (прямые коды) суммируются. Переполнение определяется наличием переноса 1 из старшего разряда мантиссы в 7-й разряд регистра результата сумматора (в поле характеристики), что вызывает нарушение нормализации мантиссы влево. Нормализация результата производится в регистре сдвигом мантиссы на одну шестнадцатиричную цифру вправо. Старшая тетрада мантиссы при этом заполняется 0001, а характеристика результата увеличивается на 1. После этого в регистре результата сумматора формируется результат операции: из схем анализа знаков и характеристик заносятся в нулевой разряд знак одного из слагаемых, в 1–7 разряды — характеристика, а в 8–31 разрядах сохраняется мантисса суммы в прямом коде.

б) если знаки слагаемых различны, то отрицательная мантисса преобразуется в дополнительный код и мантиссы суммируются. Признаком положительного результата является перенос 1 из старшего разряда мантиссы в 7-й разряд суммы, которая стирается. Признаком отрицательного результата — отсутствие переноса 1 в 7-й разряд, при этом мантисса суммы представлена в дополнительном коде и должна быть преобразована в прямой код. При сложении кодов чисел с разными знаками может произойти нарушение нормализации мантиссы суммы вправо (старшая шестнадцатиричная цифра мантиссы в прямом коде равна нулю).

Нормализация мантиссы суммы производится сдвигом ее влево на одну шестнадцатиричную цифру и уменьшением характеристики на единицу. После этого в регистре сумматора формируется результат: в нулевой разряд заносится знак большего по модулю слагаемого, в 1–7 разряды — характеристика и в 8–31 разряды — мантисса результата в прямом коде.

Реализацию этих положений рассмотрим на примерах.

Замечание. При действиях над кодами мантисс знаки не указываются, перед старшими разрядами мантисс через запятую сверху записывается два шестнадцатиричных нуля вместо характеристики и знака мантиссы.

Пример 6. Дано: $A = 15\frac{7}{8}$; $B = \frac{5}{16}$. Найти $C_1 = A + B$, $C_2 = -A - B$.

Решение.

$A_{16} = \text{F,E}$; $B_{16} = 0{,}5$.

а) Нормализация мантисс и определение характеристик:

$m_A = 0{,}FE$; $P_{XA} = 40 + 1 = 41$;

$m_B = 0{,}5$; $P_{XB} = 40 + 0 = 40$.

б) Выравнивание характеристик:

$\Delta P = 41 - 40 = 1$; $m_B^* = m_B \cdot 16^{-\Delta P} = m_B \cdot 16^{-1} = 0{,}05$.

в) Выполнение действий.

Вычислим сначала $C_1 = A + B$:

$$
\begin{array}{ll}
[m_A]_{\text{пк}} = 00'\text{FE}0000; & P_{XA} = 41 \\
+\ [m_B]_{\text{пк}} = 00'050000; & P_{XB} = 41 \\
\hline
[m_{C_1}]_{\text{пк}} = 01'030000; & P_{XC_1} = 41.
\end{array}
$$

Так как слагаемые имеют одинаковые знаки, то перенос единицы из старшего разряда мантиссы характеризует переполнение разрядной сетки, что является нарушением нормализации мантиссы результата влево.

Нормализация мантиссы:

$[m_{C_1}]_{\text{пк}} = 00'103000$; $P_{C_1} = 41 + 1 = 42$.

Проверка. $C_1 = 10{,}3 = 16\frac{3}{16}$; $C_1 = 15\frac{7}{8} + \frac{5}{16} = 16\frac{3}{16}$.

В разрядной сетке регистра сумматора формируется результат действия: заносятся знак результата (+), характеристика (42) и сохраняется мантисса в прямом коде.

Ответ.

$$C_1 \rightarrow \underset{\text{Зн}}{0}\ \underbrace{1000010}_{P_{XC_1}}\ \underbrace{0001000001100000000000}_{m_{C_1}}$$

или $C_1 \rightarrow 42103000$.

Теперь найдем $C_2 = -A-B$.

Так как числа A и B и результат C_2 хранятся в прямом коде, то нет необходимости выполнять двойное преобразование: данных в дополнительный, а результата в прямой коды.

Поэтому действие выполняется $C_2 = -(A+B)$, т. е. производится сложение прямых кодов мантисс, а результату приписывается знак одного из чисел (минус), т. е. $C_2 = -C_1$.

Ответ.

$$C_2 \rightarrow 1100\,0010\,0001\,0000\,0011\,0000\,0000\,0000$$

или $C_2 \rightarrow$ C2103000.

Пример 7. $A = 15\frac{7}{8}$, $B = \frac{5}{16}$. Найти $C_3 = A-B$ и $C_4 = B-A$.

$m_B^* = 0{,}05$; $P_{XB} = 41$; $[-m_B^*]_{\text{доп}} = 1{,}00-0{,}05 = 0{,}$FB; $[-m_B^*]_{\text{дк}} = 00'$FB0000.

$m_A = 0{,}$FE; $P_{XA} = 41$; $[-m_A]_{\text{дк}} = 00'020000$.

Найдем сначала $C_3 = A-B$:

$[m_A]_{\text{пк}} = 00'$FE0000; $P_{XA} = 41$

$+\,[-m_B^*]_{\text{дк}} = 00'$FB0000; $P_{XB} = 41$

$[m_{C_3}]_{\text{дк}} = 01'$F90000 > 0; $P_{XC_3} = 41$.

Так как слагаемые с разными знаками, то единица переноса из старшего разряда является признаком положительного результата и стирается, компенсируя константу дополнительного кода.

Проверка. $C_3 = \text{F},9 = 15\frac{9}{16}$.

Ответ.

$$C_3 \rightarrow 0100\,0001\,1111\,1001\,0000\,0000\,0000\,0000$$

или $C_3 \rightarrow$ 41F90000.

Теперь найдем $C_4 = B - A$.

$$\begin{array}{ll} [m_B^*]_{\text{пк}} = 00'050000; & P_{XB} = 41 \\ +\,[-m_A]_{\text{дк}} = 00'020000; & P_{XA} = 41 \\ \hline [m_{C_4}]_{\text{дк}} = 00'070000 < 0; & P_{XC_4} = 41. \end{array}$$

Так как слагаемые с разными знаками и перенос из старшего разряда отсутствует, то результат отрицательный в дополнительном коде и должен быть представлен в прямом коде.

$[m_{C_4}]_{\text{пк}} = 00'\text{F}90000; \quad P_{XC_4} = 41.$

Проверка. $C_4 = -\text{F},9 = -15\frac{9}{16}$.

Ответ.

$$C_4 \rightarrow 1100\,0001\,1111\,1001\,0000\,0000\,0000\,0000$$

или $C_4 \rightarrow$ C1F90000.

Пример 8. Дано: $A = 129$, $B = 115\frac{3}{4}$.

Найти: $C_1 = A - B$, $C_2 = B - A$.

Решение. $A_{16} = 81$, $B_{16} = 73,\text{C}$.

а) Нормализация мантисс и определение характеристик:

$m_A = 0,81; \quad P_{XA} = 40 + 2 = 42;$

$m_B = 0,73\text{C}; \quad P_{XB} = 40 + 2 = 42.$

б) Выполнение действий. Вычислим $C_1 = A - B$.

$$\begin{array}{ll} [m_A]_{\text{пк}} = 00'810000; & P_{XA} = 42 \\ +\,[-m_B^*]_{\text{дк}} = 00'8\text{C}4000; & P_{XB} = 42 \\ \hline [m_{C_1}]_{\text{дк}} = 01'0\text{D}4000 > 0; & P_{XC_1} = 42. \end{array}$$

Единица переноса стирается, результат положительный, но произошло нарушение нормализации вправо.

Нормализация мантиссы:

$[m_{C_1}]_{\text{пк}} = 00'\text{D}40000; \quad P_{XC_1} = 42-1 = 41.$

Проверка. $C_1 = \text{D},4 = 13\frac{1}{4}$.

Ответ.

$$C_1 \rightarrow 0100\,0001\,1101\,0100\,0000\,0000\,0000\,0000$$

или $C_1 = 41\text{D}40000$.

Вычислим $C_2 = B - A$.

$$\begin{array}{ll} [m_B]_{\text{пк}} = 00'73\text{C}000; & P_{XB} = 42 \\ +[-m_A]_{\text{дк}} = 00'7\text{F}0000; & P_{XA} = 42 \\ \hline [m_{C_2}]_{\text{дк}} = 00'\text{F}2\text{C}000 < 0; & P_{XC_2} = 42. \end{array}$$

Так как Зн $A \neq$ Зн B и перенос отсутствует из старшего разряда, то результат отрицательный, представлен в дополнительном коде и должен быть преобразован в прямой код.

$[m_{C_2}]_{\text{пк}} = 00'0\text{D}4000; \quad P_{XC_2} = 42.$

Старшая тетрада мантиссы равна нулю — нарушение нормализации вправо.

Нормализация мантиссы:

$[m_{C_2}]_{\text{пк}} = 00'\text{D}40000; \quad P_{XC_2} = 42-1 = 41.$

Ответ.

$$C_2 \rightarrow 1100\,0001\,1101\,0100\,0000\,0000\,0000\,0000$$

или $C_2 = \text{C}1\text{D}40000$.

Замечание. При выполнении операций умножения и деления порядки не выравниваются, нормализованные мантиссы чисел умножаются или делятся, порядки соответственно складываются или вычитаются, а знаки произведения и частного определяются, как и в естественной форме, сложением по модулю 2 знаков сомножителей или делимого и делителя.

Задачи для самостоятельного решения

Коды чисел в формате с фиксированной точкой

Задача. Получить машинные коды двух целых десятичных чисел A и B с фиксированной точкой в 16-ти разрядной сетке, используя 2-ю или 16-ю систему счисления.

	A	B		A	B
1.	378	−456	8.	384	−428
2.	−516	314	9.	128	−382
3.	256	−412	10.	199	−581
4.	−264	−332	11.	314	−489
5.	444	−563	12.	−576	320
6.	268	−365	13.	−388	260
7.	−260	−252	14.	129	−583

Коды чисел в формате с плавающей точкой

Задача. Получить машинные коды двух вещественных десятичных чисел A и B с плавающей точкой в 32-х разрядной сетке, используя 2-ю или 16-ю систему счисления.

	A	B		A	B
15.	−259,5	524 1/16	22.	−124,0625	124 3/32
16.	−524,09375	14 3/4	23.	30,625	−285 9/16
17.	250,375	15 1/2	24.	40,1875	−295 1/2
18.	−260,625	13 7/8	25.	0,015625	1/32
19.	−0,03125	1/64	26.	51,0625	216 3/4
20.	12,125	24 5/8	27.	14,125	−166 1/4
21.	128,25	−140 1/8	28.	−216,625	−50 3/8

Сложение чисел в формате с фиксированной точкой

Задача. Найти сумму (разность) двух чисел A и B, заданных в виде машинных кодов в формате с фиксированной точкой в 16-ти разрядной сетке. В качестве

ответа записать код суммы чисел (в 2-й или 16-й системе счисления) и десятичное число, соответствующее этому коду.

	A	*B*		*A*	*B*
29.	FE38	017A	36.	FE54	0180
30.	013A	FDFC	37.	FE82	0080
31.	FE64	0100	38.	FDBB	00C7
32.	FEB4	FEF8	39.	FE17	013A
33.	FDCD	01BC	40.	0140	FDC0
34.	FE93	010C	41.	0104	FE7C
35.	FF04	FEFC	42.	FDB9	0081

Сложение чисел в формате с плавающей точкой

Задача. Найти сумму (разность) двух чисел *A* и *B*, заданных в виде машинных кодов в формате с плавающей точкой в 32-х разрядной сетке. В качестве ответа записать код суммы чисел (в 2-й или 16-й системе счисления) и десятичное число, соответствующее этому коду.

	A	*B*		*A*	*B*
43.	41EC0000	C3103800	50.	427C1800	C27C1000
44.	4320C100	C320C180	51.	C3F1D900	421EA000
45.	41F80000	42FA6000	52.	C3127800	42283000
46.	41DE0000	C3104A00	53.	3F800000	3F400000
47.	3F400000	BF800000	54.	42D8C000	42331000
48.	4218A000	41C20000	55.	C2A64000	41E20000
49.	C28C2000	42804000	56.	C232C000	C2D8A000

Часть вторая

Логические основы ЭВМ

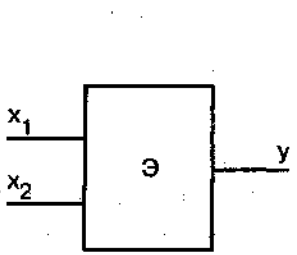
Основные понятия алгебры логики

Изучение темы «Логические основы ЭВМ» и решение логических задач — это залог успешного решения учебных задач по информатике и практических задач во всех сферах человеческой деятельности. Знание логических операций и умение строить логические выражения помогают легко и быстро освоить условные выражения и операторы любого языка программирования.

В отличие от обычной алгебры, изучающей математические функции, алгебра логики изучает логические функции. Известно, что функция — это закон соответствия между переменными. Следовательно, *логическая функция* — это закон соответствия между логическими переменными. *Логическая переменная* — это такая переменная, которая может принимать одно из двух возможных значений: 0 («ложь») и 1 («истина»).

Логическая функция может также принимать два значения. Из отмеченного следует, что логические переменные и функции определены на множестве двух значений — {0, 1}.

ЭВМ строятся из компонентов с двумя устойчивыми состояниями. Одно состояние обозначается нулем, другое — единицей. На такие компоненты воздействуют двоичные сигналы. Под воздействием сигналов компоненты изменяют свои состояния, т. е. состояние компонентов или значения их выходных сигналов за-



x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

а *б*

***Схема 12.** Пример логического элемента и его таблицы истинности*

висят от значений воздействующих сигналов. Очевидно, что функционирование компонентов ЭВМ следует описывать логическими функциями. По этой причине алгебра логики находит непосредственное и широкое применение при разработке и использовании средств электронной вычислительной техники.

Логические функции характеризуются (задаются) так называемыми таблицами истинности, или соответствия. *Таблица истинности* — это таблица, устанавливающая соответствие между возможными наборами значений логических переменных и значениями функций. Для иллюстрации сказанного рассмотрим компонент (элемент) ЭВМ, представленный на схеме 12*а*.

На элемент воздействуют два входных двоичных сигнала, описываемые логическими переменными x_1 и x_2. Выходной сигнал Y есть тогда, когда есть и сигнал x_1, и сигнал x_2, т. е. выходной сигнал как функция $Y = f(x_1, x_2)$ принимает единичное значение при единичных значениях x_1 и x_2 ($x_1 = 1, x_2 = 1$).

Во всех остальных случаях выходной сигнал принимает нулевое значение. Такому словесному описанию функционирования элемента Э соответствует таблица истинности, представленная на схеме 12*б*. Воз-

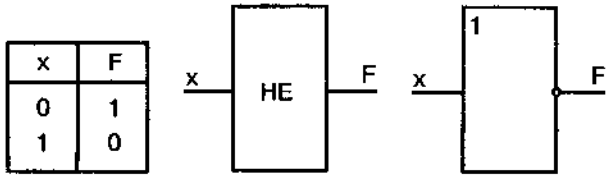
можным наборам значений логических переменных x_1, x_2 (00, 01, 10, 11) соответствуют десятичные цифры 0, 1, 2 и 3. Поэтому можно считать, что функция Y принимает единичное значение на наборе 3. Количество возможных наборов переменных зависит от числа этих переменных. Если функция Z зависит от одной переменной, т. е. $Z = f(x)$, то всего наборов значений x два ($x = 0$, $x = 1$). Для рассмотренной ранее функции $Y = f(x_1, x_2)$, которая зависит от двух переменных, возможных наборов значений переменных четыре: 00, 01, 10, 11. Для некоторой логической функции $Q = f(x_1, x_2, x_3)$ от трех переменных возможных наборов будет восемь: 000, 001, 010, 011, ..., 111. Очевидно, что в общем случае количество N возможных наборов значений переменных логической функции можно определить по формуле $N = 2^r$, где r — количество переменных, от которых зависит логическая функция.

Сложные логические функции, как правило, выражаются через простые. Простые логические функции, зависящие от одной или двух логических переменных называются элементарными. Далее логические переменные будут называться переменными.

Элементарные логические функции

К основным относятся следующие элементарные логические функции:

1) отрицание; 2) логическое умножение (конъюнкция); 3) отрицание от логического умножения (отрицание от конъюнкции); 4) логическое сложение (дизъюнкция); 5) отрицание от логического сложения (отрицание от дизъюнкции); 6) равнозначность; 7) отрицание равнозначности.



а *б* *в*

Схема 13. Элемент «НЕ»

Логическое отрицание (инверсия)

Отрицание — это логическая функция от одной переменной, которая принимает единичное значение при нулевом значении переменной и наоборот. Запись этой функции: $F = \bar{x}$. Любая логическая функция имеет свое обозначение. Возможные обозначения инверсии: $\neg x$; $\bar{x}$; NOT x. В данном пособии используется обозначение $\bar{x}$, т. е. черта над переменной x является признаком отрицания (инверсии).

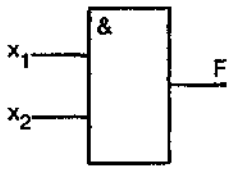
Таблица истинности этой функции представлена на схеме 13*а*. Из таблицы истинности следует, что инверсия высказывания истинна, когда высказывание ложно, и ложна, когда высказывание истинно.

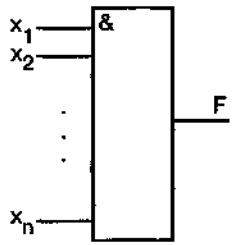
Функция логического отрицания описывает функционирование логического элемента «НЕ» (схема 13*б*). Условно-графическое обозначение элемента «НЕ» приведено на схеме 13*в*. Единичный сигнал на выходе элемента «НЕ» появляется при нулевом сигнале на входе ($x = 0$, $F = 1$), и наоборот, нулевой сигнал на выходе появляется при единичном сигнале на входе ($x = 1$, $F = 0$). Элемент «НЕ» реализует функцию отрицания.

Логическое умножение (конъюнкция)

Логическое умножение — это логическая функция по крайней мере от двух переменных, которая принимает единичное значение при единичных значениях

x_1	x_2	F
0	0	0
0	1	0
1	0	0
1	1	1





а *б* *в*

Схема 14. *Элемент «И»*

всех переменных. Возможные обозначения конъюнкции: x_1 И x_2; $x_1 \wedge x_2$; $x_1 \& x_2$; $x_1 \cdot x_2$; x_1 AND x_2; $x_1 x_2$; $x_1 \& \& x_2$. Эта функция называется также конъюнкцией. Элементарная конъюнкция зависит от двух переменных. Она принимает единичное значение только тогда, когда и первая переменная и вторая переменная равны единице. В данном пособии возможны такие варианты записи конъюнкции:

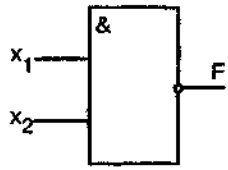
$$F = x_1 \wedge x_2; \quad F = x_1 \cdot x_2; \quad F = x_1 x_2; \quad F = x_1 \& x_2.$$

Конъюнкция характеризуется таблицей истинности, представленной на схеме 14*а*. Из рассмотрения таблицы следует, что эта функция принимает единичное значение на наборе 3. Логическое умножение описывает работу элемента «И» (схема 14*б*). Единичный сигнал появляется на выходе этого элемента только при наличии единичного сигнала и на входе 1, и на входе 2. Элемент И реализует функцию логического умножения. В общем случае элемент И может иметь n входов (схема 14*в*). При этом он реализует конъюнкцию от n переменных, т. е.

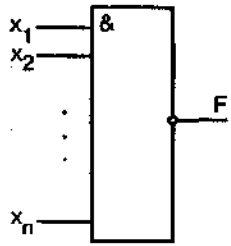
$$F = x_1 \cdot x_2 \cdot x_3 \cdot \ldots \cdot x_n.$$

x_1	x_2	F
0	0	1
0	1	1
1	0	1
1	1	0

а



б



в

Схема 15. Элемент «И-НЕ»

Отрицание от логического умножения (отрицание от конъюнкции)

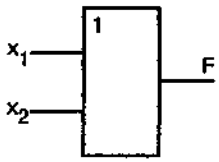
Отрицание от логического умножения (или отрицание от конъюнкции) — это логическая функция по крайней мере от двух переменных, которая принимает нулевое значение при единичных значениях всех переменных. Эта функция называется также функцией Шеффера. Таблица истинности элементарной функции отрицания от конъюнкции представлена на схеме 15*а*. Функция принимает единичные значения на наборах 0, 1, 2 и нулевое значение — только на наборе 3, т. е. тогда, когда обе переменные равны единице. Запись функции: $F = \overline{x_1 \cdot x_2}$, или $F = \overline{x_1 \wedge x_2}$, или $F = \overline{x_1 x_2}$.

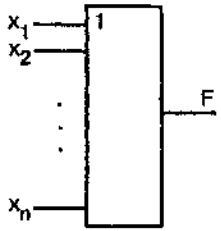
Функция отрицания от конъюнкции описывает работу элемента «И-НЕ» (схема 15*б*). Нулевой сигнал на выходе этого элемента появляется только при наличии единичного сигнала и на входе 1 и на входе 2.

В общем случае элемент «И-НЕ» может иметь *n* входов (схема 15*в*). При этом он реализует функцию Шеффера от *n* переменных, т. е.

$$F = \overline{x_1 \wedge x_2 \wedge \cdots \wedge x_n}.$$

x_1	x_2	F
0	0	0
0	1	1
1	0	1
1	1	1





а *б* *в*

Схема 16. Элемент «ИЛИ»

Логическое сложение (дизъюнкция)

Логическое сложение — это логическая функция по крайней мере от двух переменных, которая принимает нулевое значение при нулевых значениях всех переменных. Функция логического сложения называется также дизъюнкцией.

Таблица истинности элементарной дизъюнкции представлена на схеме 16*а*. Элементарная дизъюнкция принимает единичное значение на наборах 1, 2, 3 и нулевое значение — только на наборе 0. В данном пособии функция записывается в одном из двух видов: $F = x_1 \vee x_2$ или $F = x_1 + x_2$, хотя в литературе встречаются и такие обозначения: x_1 ИЛИ x_2; x_1 OR x_2; $x_1 \mid x_2$; $x_1 \parallel x_2$.

Знак «плюс» не является алгебраическим, так как при $x_1 = 1$, $x_2 = 1$ дизъюнкция $F = x_1 + x_2 = 1$, т. е. она не может быть равной 2.

Дизъюнкция описывает функционирование элемента «ИЛИ» (схема 16*б*). Единичный сигнал на выходе этого элемента возникает тогда, когда или на входе 1, или на входе 2, или на двух входах есть единичные сигналы. И только в том случае, когда на оба входа поступают нулевые сигналы, на выходе элементов появляется нулевой сигнал. Функционирование элемента «ИЛИ» полностью соответствует таблице истинно-

сти для дизъюнкции.

В общем случае элемент «ИЛИ» может иметь *n* входов (схема 16*в*). При этом он реализует дизъюнкцию от *n* переменных.

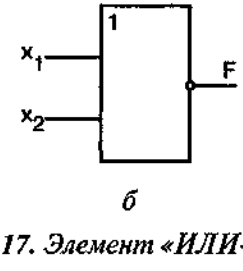
Отрицание от логического сложения (отрицание от дизъюнкции)

Отрицание от логического сложения — это логическая функция по крайней мере от двух переменных, которая принимает единичное значение при нулевых значениях всех переменных. Эта функция называется также функцией Пирса, или функцией отрицания от дизъюнкции. Элементарная функция Пирса зависит от двух переменных. Она принимает единичное значение только тогда, когда обе переменные принимают нулевые значения. Запись функции: $F = \overline{x_1 \vee x_2}$ или $F = \overline{x_1 + x_2}$.

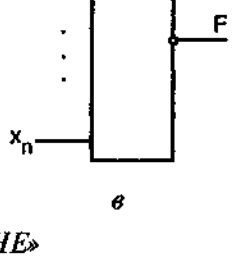
Таблица истинности элементарной функции отрицания от дизъюнкции представлена на схеме 17*а*. Функция принимает единичное значение на наборе 0 и нулевые значения на наборах 1, 2, 3. Функция отрицания от дизъюнкции описывает работу элемента «ИЛИ-НЕ» (схема 17*б*). В соответствии с таблицей истинности (схема 17*а*) единичный сигнал на выходе этого элемента появляется при наличии на входах 1 и 2 ну-

x_1	x_2	F
0	0	1
0	1	0
1	0	0
1	1	0

а



б

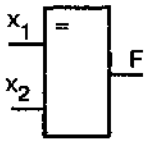


в

Схема 17. Элемент «ИЛИ-НЕ»

x_1	x_2	F
0	0	1
0	1	0
1	0	0
1	1	1

а



б

Схема 18. Элемент равнозначности

левых сигналов. В других случаях на выходе элемента имеет место нулевой сигнал. Элемент «ИЛИ-НЕ» может иметь *n* входов (схема 17*в*). При этом он реализует функцию Пирса от *n* переменных, т. е.

$$F = \overline{x_1 \vee x_2 \vee \ldots \vee x_n}.$$

Равнозначность

Равнозначность — это логическая функция от двух переменных, которая принимает единичное значение при одинаковых значениях переменных. Одинаковые по значению переменные называются равнозначными, поэтому функция носит название «равнозначность».

Запись функции: $F = x_1x_2 + \overline{x_1}\,\overline{x_2}$.

Таблица истинности функции равнозначности представлена на схеме 18*а*. Эта функция реализуется элементом равнозначности (сравнения), который показан на схеме 18*б*.

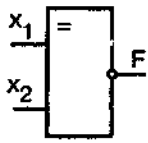
Элемент используется для сравнения двоичных сигналов.

Отрицание равнозначности

Отрицание равнозначности — это логическая функция от двух переменных, которая принимает единич-

x_1	x_2	F
0	0	0
0	1	1
1	0	1
1	1	0

а



б

***Схема 19.** Элемент отрицания равнозначности*

ное значение при разных по значимости переменных. Таблица истинности функции представлена на схеме 19*а*.

Запись функции: $F = \overline{x_1}x_2 + x_1\overline{x_2}$.

Отрицание равнозначности называют функцией сложения переменных по модулю два. Сложить по модулю — это значит представить сумму в виде остатка от деления ее на модуль.

Используя таблицу истинности (схема 19*а*), легко показать, что на любом наборе сумма переменных, представленная по модулю два, является значением функции F, например, на наборе 3: $x_1 = 1$, $x_2 = 1$, $x_1 + x_2 = 2$.

Эта сумма по модулю два равна нулю, что совпадает со значением F в таблице.

Функция отрицания равнозначности реализуется одноименным элементом, представленным на схеме 19*б*. Элемент выдает единичные сигналы при поступлении на входы 1 и 2 сигналов, отличающихся значением.

Формы логических функций

Одна и та же логическая функция может быть записана различным образом. Например, функция $F(x_1, x_2)$ может быть записана следующими эквивалентными выражениями:

$$F(x_1, x_2) = \overline{x_1}x_2 + \overline{x_1}\,\overline{x_2} + x_1x_2; \quad (2.1)$$

$$F(x_1, x_2) = \overline{x_1} + x_1x_2; \quad (2.2)$$

$$F(x_1, x_2) = \overline{x_1(\overline{x_1} + \overline{x_2})}. \quad (2.3)$$

Эквивалентность выражений легко проверить подстановкой в них значений x_1 и x_2. Для исключения неоднозначности записи логические функции представляют в унифицированных формах. Такими формами являются: дизъюнктивная и конъюнктивная. В них используются элементарные дизъюнкции и конъюнкции.

Элементарной называется конъюнкция, в которую входят только переменные и их отрицания, например, x_1x_2; $x_1\overline{x_2}$; $\overline{x_1} \cdot \overline{x_2}$; $x_1x_2\overline{x_3}$; $\overline{x_1}\,\overline{x_2}\,\overline{x_3}$ и т. п.

Элементарной называется дизъюнкция, представляющая собой логическую сумму переменных и их отрицаний. Например:

$$x_1 + x_2; \quad x_1 + \overline{x_2}; \quad x_1 + \overline{x_2} + \overline{x_3}.$$

В элементарные конъюнкции (дизъюнкции) не могут входить одинаковые переменные, а также переменные с их отрицаниями. Такие дизъюнкции (конъюнкции) должны преобразовываться. При этом они упрощаются, а также превращаются в 0 или 1, например, $x_1x_1x_1 = x_1$; $x_1\overline{x_1} = 0$; $x_1 + x_1 + x_1 = x_1$; $x_2 + \overline{x_2} = 1$ и т. п.

Правильность преобразований может быть проверена подстановкой значений переменных. Элементарная конъюнкция (дизъюнкция) может характеризо-

ваться рангом, равным количеству переменных в конъюнкции (дизъюнкции). Понятия элементарной дизъюнкции и конъюнкции позволяют достаточно просто определить дизъюнктивную и конъюнктивную формы записи логических функций.

Дизъюнктивная нормальная форма (ДНФ) — это форма, в которой логическая функция представляется в виде дизъюнкции элементарных конъюнкций, например:

$$F = x_1x_2 + \overline{x_1}x_3 + x_1x_2x_3. \qquad (2.4)$$

Функции выражений (2.1), (2.2) записаны также в ДНФ.

Конъюнктивной нормальной формой (КНФ) называется такая форма, в которой функция представляется в виде конъюнкции элементарных дизъюнкций. Например: $F = (x_1 + \overline{x_2})(\overline{x_1} + x_2 + \overline{x_3})$.

Использование нормальных форм не устраняет полностью неоднозначности записи логических функций. Например, функция (2.4) может быть записана также выражениями:

$$F = x_1x_2 + \overline{x_1}x_3 + x_2x_3; \qquad (2.5)$$

$$F = x_1x_2 + \overline{x_1}x_3. \qquad (2.6)$$

Совершенные формы записи логических функций

Среди нормальных форм выделяются такие, в которых функции записываются единственным образом. Их называют совершенными. Применяются *совершенная дизъюнктивная* и *совершенная конъюнктивная нормальные формы* (СДНФ и СКНФ). Формы СДНФ и СКНФ имеют две отличительные особенности:

1) все элементарные конъюнкции и дизъюнкции имеют одинаковый ранг;

2) в элементарные конъюнкции (дизъюнкции) входят все те переменные или их отрицания, от которых зависит функция.

Функция (2.5) содержит конъюнкции одинакового ранга, но записана в ДНФ, а не в СДНФ. Это объясняется тем, что элементарные конъюнкции не содержат всех тех переменных или их отрицаний, от которых зависит функция. Функция $F(x_1, x_2, x_3) = x_1x_2x_3 + x_1x_2\overline{x_3} + \overline{x_1}x_2x_3 + \overline{x_1}\,\overline{x_2}x_3$ записана в СДНФ.

Функции в СДНФ и СКНФ обычно записываются по таблицам истинности с использованием определенных правил.

Правило записи СДНФ функции по таблице истинности.

Для всех наборов переменных, на которых функция принимает единичные значения, записать конъюнкции, инвертируя те переменные, которым соответствуют нулевые значения. Затем конъюнкции соединить знаками дизъюнкции.

Например, логическая функция задана таблицей истинности, представленной на схеме 20*а*. Для наборов 3, 5, 6, 7 записываем конъюнкции через пробел:

$$\overline{x_1}x_2x_3 \quad x_1\overline{x_2}x_3 \quad x_1x_2\overline{x_3} \quad x_1x_2x_3.$$

В пробелы ставим знак дизъюнкции и получаем функцию в СДНФ, т. е.

$$F(x_1, x_2, x_3) = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3.$$

Для задания функции не обязательно всегда составлять таблицу истинности. Можно указать, что функция $F(x_1, x_2, x_3)$ равна единице, например, на наборах 3, 5, 6, 7 (011, 101, 110, 111).

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

а

x_1	x_2	x_3	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

б

Схема 20. *Таблицы истинности логических функций*

Правило записи СКНФ функции по таблице истинности.

Для всех наборов переменных, на которых функция принимает нулевые значения, записать дизъюнкции, инвертируя те переменные, которым соответствуют единичные значения. Затем дизъюнкции соединить знаками конъюнкции.

Например, пусть логическая функция задана прежней таблицей истинности, представленной на схеме 20*а*. Для наборов 0, 1, 2, 4 записываем элементарные дизъюнкции:

$$(x_1 + x_2 + x_3) \quad (x_1 + x_2 + \overline{x_3}) \quad (x_1 + \overline{x_2} + x_3) \quad (\overline{x_1} + x_2 + x_3).$$

Дизъюнкции соединяем знаками конъюнкции и получаем функцию в СКНФ, т. е.

$$F(x_1, x_2, x_3) =$$
$$= (x_1 + x_2 + x_3)(x_1 + x_2 + \overline{x_3})(x_1 + \overline{x_2} + x_3)(\overline{x_1} + x_2 + x_3).$$

При построении ЭВМ широко используются компоненты, работа которых описывается функциями, представленными в дизъюнктивных формах. Поэтому целесообразно в дальнейшем рассматривать эти формы логических функций.

Законы алгебры логики и их следствия

Основные законы

В алгебре используются три основных закона: переместительный, сочетательный и распределительный. В алгебре логики кроме трех перечисленных законов используется четвертый — закон отрицания. Кроме того, распределительный закон алгебры логики имеет две модификации. Можно также считать, что в алгебре логики используется два распределительных закона (первый и второй). Рассмотрим законы алгебры логики.

Переместительный закон

Формулировка: логические переменные можно менять местами. Возможные варианты записи:

$$x_1 + x_2 + x_3 = x_2 + x_1 + x_3 = x_3 + x_1 + x_2$$

и

$$F = x_1x_2x_3 = x_1x_3x_2 = x_2x_1x_3.$$

Сочетательный закон

Формулировка: логические переменные в конъюнкциях и дизъюнкциях можно объединять в группы. Возможные варианты записи:

$$F = x_1 + x_2 + x_3 = (x_1 + x_2) + x_3 = x_1 + (x_2 + x_3);$$
$$F = x_1x_2x_3x_4 = (x_1x_2) \cdot (x_3x_4) = x_1(x_2x_3x_4) = (x_1x_2x_3)x_4.$$

Распределительный закон

Формулировка: одинаковые переменные в конъюнкциях и дизъюнкциях можно выносить за скобки. Закон имеет две модификации. Первая модификация называется распределением конъюнкции по дизъюнкции. Форма записи закона в первой модификации:

$$F = x_1(x_2 + x_3) = x_1x_2 + x_1x_3.$$

Вторая модификация закона называется распределением дизъюнкции по конъюнкции. Форма записи закона во второй модификации:

$$F = x_1 + (x_2 x_3) = (x_1 + x_2)(x_1 + x_3).$$

Распределительный закон в первой модификации аналогичен распределительному закону обычной алгебры. Вторая модификация закона применима только к логическим функциям.

Закон отрицания (инверсии)

Закон отрицания имеет две формулировки. Первая: отрицание от конъюнкции равно дизъюнкции отрицаний переменных. Форма записи:

$F = \overline{x_1 x_2} = \overline{x_1} + \overline{x_2}$.

Вторая формулировка: отрицание от дизъюнкции равно конъюнкции отрицаний переменных. Формы записи:

$F = \overline{x_1 + x_2 + x_3} = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$ и $F = \overline{\overline{x_1} + x_2} = x_1 \cdot \overline{x_2}$.

Следует обратить внимание на то, что отрицание от отрицания переменной равно самой переменной. Законы алгебры логики и следствия из них используются для преобразования и упрощения логических функций. Для этих же целей применяются так называемые тождественные соотношения. Рассмотрим тождественные соотношения, а затем следствия из законов алгебры логики.

Тождественные соотношения

Тождественные соотношения проверяются подстановкой возможных значений логических переменных. Основные тождественные соотношения:

1) $x_1 \wedge x_1 \wedge x_1 \wedge \cdots \wedge x_1 = x_1$;
2) $x_1 + x_1 + x_1 + \cdots + x_1 = x_1$;
3) $x_1 \overline{x_1} = 0$;
4) $x_1 + \overline{x_1} = 1$;

5) $\overline{\overline{x_1}} = x_1$; $\overline{\overline{\overline{x_1}}} = \overline{x_1}$;
6) $1 + F = 1$;
7) $1 \wedge F = F$;
8) $0 \wedge F = 0$.

В тождествах 6, 7, 8 символом F обозначена любая логическая функция или переменная.

Следствия из законов алгебры логики

Следствия из законов алгебры логики применяются в качестве правил для упрощения логических функций. Упрощение логических функций называется также минимизацией, а упрощенная функция — минимальной. Минимальная функция обычно записывается в ДНФ. Она содержит наименьшее количество конъюнкций минимально возможного ранга. Минимальная логическая функция не поддается дальнейшему упрощению. Для минимизации логических функций используются следующие правила: поглощения, свертки, расширения и склеивания. Рассмотрим эти правила.

1. *Правило поглощения.* Данное правило является следствием из распределительного закона. Оно может быть записано в следующем виде:

$$x_1 + x_1x_2 + x_1x_2x_3 = x_1.$$

Правило доказывается следующим образом. Переменная x_1, общая для всех конъюнкций, выносится за скобки, т. е.

$$x_1 + x_1x_2 + x_1x_2x_3 = x_1(1 + x_2 + x_2x_3).$$

При $F = x_2 + x_2x_3$ получаем $x_1(1 + x_2 + x_2x_3) = x_1(1 + F)$.

Выражение $(1 + F)$ равно единице и, следовательно, правило доказано. Вид правила может быть различным. Важным является то, что одна конъюнкция

должна быть общей группой для всех других конъюнкций. Например:

$$xy + xyz + xyr + xyrs = xy(1 + z + r + rs) = xy.$$

2. *Правило свертки.* Правило является следствием второго распределительного закона. Запись правила:

а) $x_1 + \overline{x_1}x_2 = x_1 + x_2$;

б) $\overline{x_1} + x_1x_2 = \overline{x_1} + x_2$.

Доказательство. К левой части выражения применяется второй распределительный закон, т. е. $x_1 + \overline{x_1}x_2 = (x_1 + \overline{x_1})(x_1 + x_2)$.

Первая скобка правой части равна единице (см. тождественное соотношение 4), поэтому $x_1 + \overline{x_1}x_2 = x_1 + x_2$. Также доказывается равенство $\overline{x_1} + x_1x_2 = \overline{x_1} + x_2$.

3. *Правило расширения.* Правило записывается в следующем виде:

$$x_1x_2 + \overline{x_1}x_3 + x_2x_3 = x_1x_2 + \overline{x_1}x_3.$$

Понятие расширения объясняется возможностью добавления к правой части конъюнкции x_2x_3. Справедливость правила доказывается искусственным приемом. Конъюнкция x_2x_3 умножается на дизъюнкцию $(x_1 + \overline{x_1}) = 1$, а затем делаются простые преобразования, т. е.

$$x_1x_2 + \overline{x_1}x_3 + x_2x_3(x_1 + \overline{x_1}) = x_1x_2 + \overline{x_1}x_3 + x_1x_2x_3 + \overline{x_1}x_2x_3 = x_1x_2(1 + x_3) + \overline{x_1}x_3(1 + x_2) = x_1x_2 + \overline{x_1}x_3.$$

4. *Правило склеивания.* Правило склеивания базируется на понятии соседних конъюнкций. Соседними называются конъюнкции, отличающиеся представлением одной переменной. Например, конъюнкции $x_1x_2x_3$ и $x_1\overline{x_2}x_3$, $\overline{x_1}\,\overline{x_2}\,\overline{x_3}$ и $x_1\overline{x_2}\,\overline{x_3}$ являются попарно соседними. В первой паре конъюнкции отличаются представлением x_2, а во второй — представлением x_1. По этим переменным конъюнкции склеиваются.

Формулировка правила: две соседние конъюнкции склеиваются с образованием одной конъюнкции меньшего ранга; исчезает та переменная, по которой конъюнкции склеиваются.

Пример: Задана логическая функция в СДНФ $F = x_1\overline{x_2}\,\overline{x_3} + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$.

Необходимо упростить функцию. Так как конъюнкции функции соседние и отличаются представлением x_1, то путем их склеивания получаем $F = \overline{x_2}\,\overline{x_3}$. Справедливость преобразования (склеивания) доказывается вынесением общих переменных в конъюнкциях за скобку, т. е.

$$F = x_1\overline{x_2}\,\overline{x_3} + \overline{x_1}\,\overline{x_2}\,\overline{x_3} = \overline{x_2}\,\overline{x_3}(x_1 + \overline{x_1}) = \overline{x_2}\,\overline{x_3}. \qquad (2.7)$$

При решении логических задач следует строго соблюдать порядок выполнения логических операций, согласно их приоритету:

1. инверсия
2. конъюнкция
3. дизъюнкция
4. равнозначность.

Типовые задачи по преобразованию логических функций

Задачи по преобразованию логических функций весьма разнообразны. Однако их можно подразделить на следующие типовые группы:

- Упрощение логических функций, заданных различным образом;
- Построение таблиц истинности функций;
- Вычисление значения логического выражения для заданного набора значений переменных;
- Определение тождественности логических функций.

Упрощение логических функций, заданных различным образом

I. Функция задана в произвольной форме.

Пример 1. Упростить логическую функцию, заданную выражением

$$F = (x + \bar{y} + z)\overline{(x + y + z)}.$$

а) Приведение функции к ДНФ путем использования законов и правил, т. е. $F = (x + \bar{y} + z)\overline{(x + y + z)} = (x + \bar{y} + z) \cdot (\bar{x} \cdot \bar{y} \cdot \bar{z}) = x \cdot \bar{x}\bar{y} \cdot \bar{z} + \bar{x} \cdot \bar{y} \cdot \bar{y} \cdot \bar{z} + z\bar{x} \cdot \bar{y} \cdot \bar{z}$.

б) Вычеркивание конъюнкций, равных нулю.

Конъюнкция $\bar{x} \cdot \bar{y} \cdot \bar{y} \cdot \bar{z} = \bar{x} \cdot \bar{y} \cdot \bar{z}$, конъюнкции $\bar{x} \cdot x \cdot \bar{y} \cdot \bar{z}$, $z \cdot \bar{x} \cdot \bar{y} \cdot \bar{z}$ равны нулю, поэтому получаем упрощенную функцию $F = \bar{x} \cdot \bar{y} \cdot \bar{z}$.

Пример 2. Упростить логическую функцию, заданную выражением

$$F = (x_1 + x_2 + \overline{\overline{x_3} \cdot x_1})\ \overline{(\overline{x_1 x_2} + \overline{x_3})}.$$

а) Применение закона отрицания с целью последующего перехода к ДНФ, т. е.

$$F = (x_1 + x_2 + \overline{\overline{x_3}} + \overline{x_1})\ \overline{(\overline{x_1 x_2})}\ \overline{\overline{x_3}} = (x_1 + x_2 + x_3 + \overline{x_1})(x_1 x_2 x_3).$$

б) Анализ промежуточного результата. Устанавливаем, что первая скобка равна единице, так как $(x_1 + \overline{x_1}) = 1$ и $(x_1 + \overline{x_1} + x_2 + x_3) = (1 + f) = 1$, где $f = x_2 + x_3$.

Окончательно получаем $F = x_1 x_2 x_3$.

Пример 3. Упростить логическую функцию, заданную выражением

$$F = (\overline{x\bar{y}} + \overline{x\bar{z}})\ (x + \overline{y\bar{z}}).$$

а) Упрощение функции путем использования закона отрицания и перемножения скобок

$$F = (\bar{x} + y + \bar{x} + \bar{z})(x + \bar{y} + z) = (\bar{x} + y + \bar{z})(x + \bar{y} + z) = \\ = 0 + xy + x\bar{z} + \bar{x}\bar{y} + 0 + \bar{y}\bar{z} + \bar{x}z + yz + 0.$$

б) Применение правила расширения:

— из группы конъюнкций $xy + \bar{x}z + yz$ исключаем yz;

— из группы конъюнкций $xy + x\bar{z} + \bar{y} \cdot \bar{z}$ исключаем $x\bar{z}$;

— из группы конъюнкций $\bar{x}\bar{y} + \bar{y}\bar{z} + \bar{x}z$ исключаем $\bar{x} \cdot \bar{y}$.

Окончательно получаем $F = xy + \bar{y} \cdot \bar{z} + \bar{x}z$.

Покажем, что если применять другую последовательность исключения лишних конъюнкций, можно получить другой вид функции, которая не поддается дальнейшему упрощению.

Применяем правило расширения в следующей последовательности:

— из группы конъюнкций $x\bar{z} + \bar{x} \cdot \bar{y} + \bar{y} \cdot \bar{z}$ исключаем $\bar{y} \cdot \bar{z}$;

— из группы конъюнкций $xy + \bar{x}z + yz$ исключаем yz.

В результате получаем $F = xy + x\bar{z} + \bar{x}\bar{y} + \bar{x}z$.

Эта функция по рассмотренным правилам и законам упрощению не поддается.

Заметим, что если имеется несколько форм одной функции, не поддающихся дальнейшим упрощениям, то они называются тупиковыми. Одна из них является минимальной.

II. Функция задана таблицей истинности.

Пример 4. Упростить функцию $F(x_1, x_2, x_3)$, равную единице на наборах 3, 5, 6, 7 (011, 101, 110, 111).

а) Построение таблицы истинности (схема 20*а*).

В первых трех столбцах записываются возможные наборы. В столбце F на наборах 011, 101, 110 и 111 проставляются единицы; на остальных наборах проставляются нули.

б) Запись функции в СДНФ (см. правило записи).

Наборам 011, 101, 110, 111 соответствуют конъюнкции $\overline{x_1}x_2x_3$, $x_1\overline{x_2}x_3$, $x_1x_2\overline{x_3}$, $x_1x_2x_3$, поэтому функция будет записана в следующем виде:

$$F = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3.$$

в) Упрощение функции. Функции, записанные в СДНФ, первоначально упрощаются по правилу склеивания. Затем применяются другие правила и тождественные соотношения.

В данной функции первые три конъюнкции являются соседними с четвертой. Функция не изменится, если к ней подписать еще две конъюнкции $x_1x_2x_3$, т. е.

$$F = \overline{x_1}x_2x_3 + x_1x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2x_3 + x_1x_2\overline{x_3} + x_1x_2x_3.$$

После склеивания пар соседних конъюнкций окончательно получим $F = x_2x_3 + x_1x_2 + x_1x_3$.

Можно было не подписывать конъюнкцию $x_1x_2x_3$, а просто склеить поочередно три первые конъюнкции с четвертой конъюнкцией.

Построение таблиц истинности функций

Пример 5. Построить таблицу истинности функции: $F = x_1x_2 + \overline{x_1}x_3$.

а) Запись заданной функции в СДНФ.

Данная функция зависит от трех переменных и записана в ДНФ. Для записи функции в СДНФ первая конъюнкция умножается на выражение $(x_3 + \overline{x_3}) = 1$,

а вторая — на выражение $(x_2 + \overline{x_2}) = 1$. В скобках используются те переменные и их отрицания, которые отсутствуют в конъюнкциях:

$$F = x_1x_2(x_3 + \overline{x_3}) + \overline{x_1}x_3(x_2 + \overline{x_2}) = \\ = x_1x_2x_3 + x_1x_2\overline{x_3} + \overline{x_1}x_2x_3 + \overline{x_1}\,\overline{x_2}x_3.$$

б) Определение наборов, на которых функция принимает единичное значение.

Так как по правилу записи конъюнкций в СДНФ единице в наборе соответствует переменная, а нулю — ее отрицание, то конъюнкциям $x_1x_2x_3$, $x_1x_2\overline{x_3}$, $\overline{x_1}x_2x_3$, $\overline{x_1}\,\overline{x_2}x_3$ соответствуют наборы 111, 110, 011, 001, т. е. 7, 6, 3, 1.

в) Непосредственное построение таблицы.

В столбце F таблицы (см. схему 20*б*) на наборах 111, 110, 011, 001 проставляются единицы, а на остальных наборах ставятся нули.

Вычисление значения логического выражения для заданного набора значений переменных

Для вычисления значения логического выражения на заданном наборе значений переменных можно применять два способа: использование СДНФ и способ подстановки. Рассмотрим первый из них.

Пример 6. Вычислить значение логического выражения $V = \overline{y} \cdot \overline{z} + x\overline{y}$ при $x = 1$, $y = 1$, $z = 0$, т. е. на наборе 6, или 110.

а) Получение СДНФ заданной функции (см. выше пример 4):

$$V = \overline{y} \cdot \overline{z}(x + \overline{x}) + x\overline{y}(z + \overline{z}) = x \cdot \overline{y} \cdot \overline{z} + \overline{x} \cdot \overline{y} \cdot \overline{z} + x\overline{y}z + x \cdot \overline{y} \cdot z.$$

После исключения повторяющейся конъюнкции получаем $V = x \cdot \overline{y} \cdot \overline{z} + \overline{x} \cdot \overline{y} \cdot \overline{z} + x \cdot \overline{y} \cdot z$.

б) Определение значения выражения.

Выражение V принимает единичное значение только на наборах 100, 000 и 101, или 4, 0, 5 так как задан набор 6, то логическое выражение принимает нулевое значение ($V = 0$).

Второй способ не нуждается в особых пояснениях. При заданных значениях $x = 1$, $y = 1$ и $z = 0$ имеем $\bar{y} = 0$, $\bar{z} = 1$. Подставляем эти значения в выражение и получаем $V = 0 \cdot 1 + 1 \cdot 0 = 0 + 0 = 0$.

В случае вычисления значений сложных логических выражений второй способ не исключает ошибок.

Определение тождественности логических функций

Тождественными являются те логические функции, которые имеют одинаковые СДНФ, т. е. одинаковые таблицы истинности. Поэтому при определении тождественности для логических функций должны быть построены таблицы истинности или получены СДНФ. Таблицы или СДНФ сравниваются и делается вывод о тождественности функций.

Пример 7. Проверить тождественность логических функций:

$$F = (x_1x_2 + x_2x_3)\left(\overline{x_1\overline{x_2} + x_2x_3} + \overline{x_1}x_3\right);$$
$$f = \overline{x_2}(\overline{x_1}x_3 + x_1\overline{x_3});$$
$$P = x_2(\overline{x_1}x_3 + x_1\overline{x_3}).$$

а) Упрощение функции F.

Применяем закон отрицания и перемножаем скобки, т. е.

$$F = (x_1x_2 + x_2x_3)\left((\overline{x_1} + x_2)(\overline{x_2} + \overline{x_3}) + \overline{x_1}x_3\right) =$$
$$= (x_1x_2 + x_2x_3)(\overline{x_1} \cdot \overline{x_2} + 0 + \overline{x_1} \cdot \overline{x_3} + x_2\overline{x_3} + \overline{x_1}x_3).$$

Во второй скобке конъюнкции $\overline{x_1} \cdot \overline{x_3}$ и $\overline{x_1}x_3$ склеиваются, поэтому получаем $F = (x_1x_2 + x_2x_3)(\overline{x_1} + \overline{x_1}\,\overline{x_2} + x_2\overline{x_3})$.

Переменная $\overline{x_1}$ поглощает конъюнкцию $\overline{x_1}\,\overline{x_2}$, что дает

$$F = (x_1x_2 + x_2x_3)(\overline{x_1} + x_2\overline{x_3}),$$

или

$$F = 0 + \overline{x_1}x_2x_3 + x_1x_2\overline{x_3} + 0.$$

Функция F оказалась записанной в СДНФ, так как содержит конъюнкции одинакового ранга и в них входят все переменные, от которых она зависит.

б) Преобразование функции f.

$$f = \overline{x_2}(\overline{x_1}x_3 + x_1\overline{x_3}) = \overline{x_1}\,\overline{x_2}x_3 + x_1\overline{x_2}\,\overline{x_3}.$$

Функция f также записана в СДНФ.

Так как СДНФ функций F и f не совпадают, то они не являются тождественными.

в) Преобразование функции P.

$$P = x_2(\overline{x_1}x_3 + x_1\overline{x_3}) = \overline{x_1}x_2x_3 + x_1x_2\overline{x_3}.$$

Получена СДНФ функции P.

Функции F и P являются тождественными, так как имеют одинаковые СДНФ.

Пример 8. Проверить тождественность логических функций

$$F = (\overline{x_1}x_2 + x_2x_3)\,(x_1\overline{x_2} + \overline{x_1}x_3 + \overline{x_2}\overline{x_3}) \quad \text{и} \quad f(x_1, x_2, x_3),$$

которая принимает единичные значения на наборах 2, 3.

а) Упрощение функции F.

Применяется закон отрицания

$$F = (\overline{x_1}x_2 + x_2x_3)(x_1\overline{x_2} + \overline{x_1}x_3 + \overline{x_2} + \overline{x_3}).$$

Во второй скобке переменная $\overline{x_2}$ поглощает конъюнкцию $x_1\overline{x_2}$, что приводит к следующему результату:

$$F = (\overline{x_1}x_2 + x_2x_3)(\overline{x_1}x_3 + \overline{x_2} + \overline{x_3}).$$

Во второй скобке используется правило свертки и затем скобки перемножаются:

$$F = (\overline{x_1}x_2 + x_2x_3)(\overline{x_1} + \overline{x_2} + \overline{x_3}) =$$
$$= \overline{x_1}x_2 + \overline{x_1}x_2x_3 + \overline{x_1}x_2\overline{x_3} + 0 + 0 + 0 =$$
$$= \overline{x_1}x_2 + \overline{x_1}x_2x_3 + \overline{x_1}x_2\overline{x_3} = \overline{x_1}x_2(1 + x_3 + \overline{x_3}) = \overline{x_1}x_2.$$

б) Получение СДНФ функции F.

$$F = \overline{x_1}x_2(x_3 + \overline{x_3}) = \overline{x_1}x_2x_3 + \overline{x_1}x_2\overline{x_3}.$$

в) Получение СДНФ функции f.

Так как функция f принимает единичные значения на наборах 2 и 3, то ее СДНФ будет иметь вид $f = \overline{x_1}x_2\overline{x_3} + \overline{x_1}x_2x_3$.

Функции F и f имеют одинаковые СДНФ, следовательно, они тождественны.

Пример 9. Проверить тождественность логических функций: $F = (\overline{x} + y + \overline{z})(x + \overline{y} + z)$ и $f(x_1, x_2, x_3)$.

Функция f имеет следующую минимальную форму $f = xy + \overline{y}\,\overline{z} + \overline{x}z$.

а) Упрощение функции F.

— перемножение скобок:

$$F = (\overline{x}+y+\overline{z})(x+\overline{y}+z) = 0+xy+x\overline{z}+\overline{x}\,\overline{y}+0+\overline{y}\,\overline{z}+\overline{x}z+yz+0;$$

— исключение лишней конъюнкции $\overline{y}\,\overline{z}$ из группы $x\overline{z} + \overline{x} \cdot \overline{y} + \overline{y}\,\overline{z}$;

— исключение лишней конъюнкции yz из группы $xy + \overline{x}z + yz$.

В результате получаем $F = xy + x\overline{z} + \overline{x}\,\overline{y} + \overline{x}\,\overline{y}$.

Упрощенная форма функции F и минимальная форма функции f не совпадают. Однако это не значит, что функции не тождественны. Для окончательного вывода нужно получить СДНФ обеих функций.

б) Получение СДНФ функции F.

$$F = xy + x\bar{z} + \bar{x}\bar{y} + \bar{x}z =$$
$$= xy(z + \bar{z}) + x\bar{z}(y + \bar{y}) + \bar{x}\bar{y}(z + \bar{z}) + \bar{x}z(y + \bar{y}) =$$
$$= xyz + xy\bar{z} + xy\bar{z} + x\bar{y}\bar{z} + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z.$$

После удаления повторяющихся конъюнкций получаем $F = xyz + xy\bar{z} + x\bar{y}\bar{z} + \bar{x}\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z$.

в) Получение СДНФ функции f:

$$f = xy + \bar{y}\bar{z} + \bar{x}z = xy(z + \bar{z}) + \bar{y}\bar{z}(x + \bar{x}) + \bar{x}z(y + \bar{y}) =$$
$$= xyz + xy\bar{z} + x\bar{y}\bar{z} + \bar{x}\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z.$$

Функции F и f имеют одинаковые СДНФ и принимают единичные значения на одних и тех же наборах 0, 1, 3, 4, 6, 7. Эти функции тождественны. Так как минимальные формы функций не совпадают, то можно сделать вывод, что для функции F была получена тупиковая форма.

Задачи для самостоятельного решения

Задача. Построить таблицу истинности логической функции трех переменных $F(x_1, x_2, x_3)$ и определить номера наборов, на которых функция равна 0.

1. $x_1 \cdot \overline{x_2} \cdot \overline{x_3} + x_1 \cdot x_2 \cdot x_3$
2. $\overline{x_1} \cdot x_2 \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3$
3. $\overline{x_1} \cdot \overline{x_2} \cdot x_3 \vee x_1 \cdot \overline{x_2} \cdot \overline{x_3}$
4. $x_1 \cdot x_2 \vee x_3 \cdot \overline{x_2} \vee x_1 \cdot x_3$
5. $\overline{x_1} \cdot x_3 \vee \overline{x_2} \cdot \overline{x_3} \vee x_1 \cdot \overline{x_2}$
6. $x_1 \cdot \overline{x_3} \vee x_1 \cdot \overline{x_2} \vee x_2 \cdot \overline{x_3}$
7. $x_1 \cdot \overline{x_2} \vee \overline{x_2} \cdot x_3 \vee x_1 \cdot x_3$
8. $x_1 \cdot x_2 \vee \overline{x_1} \cdot \overline{x_2} \vee x_2 \cdot x_3$
9. $\overline{x_1} \cdot \overline{x_2} \vee x_2 \cdot x_3 \vee x_1 \cdot \overline{x_3}$
10. $x_2 \cdot x_3 \vee \overline{x_1} \cdot \overline{x_2} \vee x_3 \cdot \overline{x_1}$

11. $x_2 \cdot \overline{x_3} \vee \overline{x_2} \cdot x_3 \vee \overline{x_1} \cdot \overline{x_2}$
12. $\overline{x_1} \cdot \overline{x_2} \vee x_2 \cdot \overline{x_3} \vee \overline{x_1} \cdot \overline{x_3}$
13. $x_3 \vee x_1 \cdot x_2 \vee \overline{x_1} \cdot \overline{x_2}$
14. $x_1 \cdot x_3 \vee \overline{x_1} \cdot \overline{x_3} \vee x_2 \cdot \overline{x_3}$

Задача. Получить кратчайшую форму записи логической функции трех переменных $F(x_1, x_2, x_3)$

15. $\overline{\overline{x_1} \cdot \overline{x_2}} \vee x_1 \cdot x_3 \vee x_2 \cdot x_3$
16. $\overline{\overline{x_1} \cdot \overline{x_2}} \vee x_2 \cdot x_3$
17. $(x_2 \cdot \overline{x_3} \vee \overline{x_1 \cdot \overline{x_3}}) \cdot (\overline{x_1 \cdot \overline{x_2}} \vee \overline{x_3} \cdot x_1 \vee \overline{x_2 \cdot \overline{x_3}})$
18. $(\overline{x_3} \cdot x_1 \vee \overline{\overline{x_3} \cdot x_2}) \cdot (\overline{x_1 \cdot \overline{x_2}} \vee \overline{x_1 \cdot \overline{x_3}} \vee \overline{x_2 \cdot \overline{x_3}})$
19. $(\overline{\overline{x_3} \cdot x_1} \vee \overline{\overline{x_3} \cdot x_2}) \cdot (\overline{x_1 \cdot \overline{x_2}} \vee \overline{x_1 \cdot \overline{x_3}} \vee \overline{x_2 \cdot \overline{x_3}})$
20. $(\overline{\overline{x_2} \cdot x_1} \vee \overline{\overline{x_2} \cdot x_3}) \cdot (\overline{x_1 \cdot x_2} \vee \overline{x_2 \cdot x_3} \vee \overline{x_3 \cdot \overline{x_1}})$
21. $(\overline{x_1 \cdot x_2} \vee \overline{\overline{x_2} \cdot \overline{x_3}}) \cdot (\overline{\overline{x_1} \cdot \overline{x_2}} \vee \overline{\overline{x_1} \cdot x_3} \vee \overline{x_2 \cdot x_3})$
22. $(\overline{\overline{x_3} \cdot x_1} \vee \overline{\overline{x_3} \cdot x_2}) \cdot (\overline{x_1 \cdot \overline{x_2}} \vee \overline{x_1 \cdot \overline{x_3}} \vee \overline{x_2 \cdot \overline{x_3}})$
23. $(\overline{x_1 \cdot x_2} \vee \overline{\overline{x_2} \cdot x_3}) \cdot (\overline{\overline{x_1} \cdot \overline{x_3}} \vee \overline{x_1 \cdot x_2} \vee \overline{x_2 \cdot x_3})$
24. $(\overline{\overline{x_3} \cdot x_1} \vee \overline{\overline{x_3} \cdot x_2}) \cdot (\overline{x_1 \cdot \overline{x_2}} \vee \overline{x_1 \cdot \overline{x_3}} \vee \overline{x_2 \cdot \overline{x_3}})$
25. $(\overline{\overline{x_3} \cdot x_1} \vee \overline{x_2 \cdot \overline{x_3}}) \cdot (\overline{x_1 \cdot \overline{x_3}} \vee \overline{x_2 \cdot \overline{x_3}} \vee \overline{x_1 \cdot \overline{x_2}})$

Часть третья
Алгоритмизация и начала программирования

Основы алгоритмизации

Введение

Восхищаясь возможностями компьютера решать разнообразные задачи и моделировать на экране дисплея реальные процессы, мы часто забываем о том, что ЭВМ всего лишь выполняет программу, составленную человеком. Поэтому для успешного применения в практической деятельности компьютерной техники, помимо общих знаний о принципах ее работы, требуется уметь строить собственные алгоритмы и составлять программы для решения возникающих задач.

Освоить основы алгоритмизации и программирования, приемы и методы решения типовых задач, предусмотренные для выпускников средних учебных заведений программой курса «Основы информатики и вычислительной техники», Вам поможет данная часть настоящего пособия.

Предлагаемые в данной части методические рекомендации имеют следующую структуру. В первом разделе изложены основные понятия, требования и методические основы составления алгоритмов и программ, следующие четыре раздела посвящены основам практического программирования. В качестве языка программирования, иллюстрирующего приемы и методы составления программ, выбран язык Basic. Причины данного выбора подробно изложены далее. Предполагается, что содержание материала пособия будет до-

ступно даже тем, кто еще не знает данного языка программирования.

Раздел «Начала программирования на языке Basic» посвящен освоению основных элементов языка и его применению при программировании базовых алгоритмических конструкций; в этом разделе в качестве объектов алгоритма выступают простые переменные и константы. Раздел «Массивы» знакомит со структурированными типами данных — одномерными и многомерными массивами. В самостоятельный раздел вынесено рассмотрение техники работы с символьными типами данных — строками. Эти разделы содержат большое количество задач, демонстрирующих как использование элементов языка Basic, так и общие принципы составления программ. Примеры программ приводятся в порядке усложнения; задачи объединены в группы в зависимости от того, какие типовые алгоритмы положены в основу их решения. Программы снабжены подробными пояснениями. Во избежание повторов в отдельных случаях даются ссылки на задачи, рассмотренные ранее.

В разделе «Неформализованные задачи» показаны основные практические методы и приемы разработки программ для решения более сложных задач. Составление подобных программ требует не только знания языка программирования и умения использовать типовые алгоритмы, но также предполагает творческий подход к решению, поскольку целью разработчика является создание нестандартного алгоритма. Следует отметить, что реальные задачи именно таковы. В этом же разделе перечислены требования, предъявляемые к оформлению текста программ.

Изучать алгоритмизацию и программирование вычислительного процесса рекомендуется последовательно. При этом весьма полезным будет такой прием: ознакомившись с теоретической частью параграфа,

постарайтесь самостоятельно решить предлагаемую в качестве примера задачу, а затем сравните полученное решение с изложенным в пособии. Не огорчайтесь, если оно оказалось не совсем правильным, а установите причину ошибки или несоответствия Вашего решения предложенному.

Самостоятельное решение задач — единственный способ не только освоить приемы алгоритмизации и программирования, но и развить алгоритмическое мышление. Дополнительные задачи для самостоятельного решения приведены в соответствующем разделе.

Несколько слов об использовании компьютера при освоении материала данной части пособия. Конечно, научиться программировать по-настоящему без компьютера достаточно сложно. Составление же программ на компьютере представляется многим значительно более простым делом, так как данный подход позволяет:

- автоматически находить в введенной программе синтаксические ошибки;
- быстро и легко протестировать программу и выявить большинство алгоритмических ошибок;
- не только понять, но и почувствовать, как работает тот или иной алгоритм;
- превратить процесс программирования в весьма увлекательное занятие, когда разработчик программы сразу видит результат своей работы.

Однако в этом подходе присутствуют и отрицательные моменты. Тот, кто привык составлять программы только с помощью компьютера, часто оказывается неспособным распознать логические ошибки в собственном алгоритме. А ведь компьютер не всегда есть под рукой. Кроме того, не продуманная заранее, «рождающаяся с клавиатуры» программа сплошь и рядом оказывается сумбурной и нерациональной. Поэтому при освоении материала данной части пособия стоит при-

держиваться «золотой середины». Только в том случае, когда составленные Вами на бумаге программы безошибочно работают при введении их в компьютер, можно считать, что успех в деле освоения алгоритмизации и программирования достигнут полностью. Пусть это и будет для Вас основным критерием при оценке уровня собственных навыков.

Алгоритм и его свойства

Освоение алгоритмизации программирования необходимо для решения информационных задач, к которым относятся задачи, связанные с моделированием различных явлений и процессов, анализом результатов эксперимента, кодированием и декодированием информации, разработкой алгоритмов решения математических и иных задач, выполнением вычислений, поиском нужной информации и т. п.

Для успешного решения информационных задач необходимо понимать, что такое модель вообще и компьютерная модель в частности, как осуществляются формализация задачи и моделирование, а также владеть языком программирования, поскольку составление программы — важнейший этап решения информационной задачи.

По мере развития вычислительной техники и усложнения решаемых с помощью компьютера задач совершенствуется и технология программирования. На каждом этапе формируется новый подход к написанию программ. При этом новые методы не отрицают своих предшественников: выбор подхода к решению задачи определяется степенью ее сложности. Можно выделить три принципиально разных подхода.

1. *Алгоритмическое (операционное) программирование* — подход, при котором программа представляет собой единое целое; структурными единицами программы являются операторы и операторные блоки,

исполняемые компьютером один за другим. Подобный подход применим для составления небольших программ простой структуры. Может быть реализован в любом языке программирования, в частности, в ассемблере.

2. *Структурное программирование* предполагает разбиение задачи на подзадачи, каждая из которых может быть решена отдельно. В соответствии с этим программа делится на точно обозначенные автономные подпрограммы (модули), исполняемые по мере необходимости. Структурными единицами программы являются подпрограммы. Этот подход поддерживается большинством современных языков высокого уровня общего назначения, в том числе и QBasic, и вполне пригоден для решения задач средней степени сложности.

3. *Объектно-ориентированное программирование* аккумулирует лучшие идеи структурного программирования, сочетая их с концепциями инкапсуляции, наследования и полиморфизма. Данная технология предполагает построение модели объекта реального мира. Для описания модели программист создает особые типы данных — классы. Класс объединяет набор свойств семейства однотипных объектов и методы для работы с этими свойствами, описывая поведение объекта указанного типа. Механизм инкапсуляции обеспечивает это объединение и защищает «внутренний мир» объекта от внешнего вмешательства. Механизм наследования позволяет формировать новые, более сложные классы на основе уже существующих, поддерживая концепцию иерархии классов. Полиморфизм дает возможность использовать одноименные методы для выполнения подобных по сути действий применительно к объектам разных классов. Этот подход положен в основу языков C++, Smalltalk, Turbo Pascal (начиная с версии 5.5), Visual Basic, VBA и др.

Естественный путь освоения материала — от простого к сложному, поэтому в настоящем пособии используется преимущественно первый подход. Это позволяет сосредоточиться на рассмотрении базовых алгоритмических конструкций и разобрать решения типовых задач, которые, по сути, являются теми «кирпичиками» знаний программиста, из которых он сможет в дальнейшем освоить структурное и объектно-ориентированное программирование, построить сколь угодно сложную программу.

Людям в повседневной жизни постоянно приходится решать различные задачи. Это могут быть как бытовые задачи («приготовить обед», «перейти улицу» и т. п.), так и профессиональные («составить расписание занятий в школе», «изготовить деталь на токарном станке», «доставить груз в пункт назначения» и т. п.). Все задачи можно классифицировать по различным признакам, например: математические, научные, инженерные, экономические, управленческие.

Некоторые задачи мы решаем легко, порой «автоматически», т. е. не задумываясь над тем, как решаем задачу. Над другими приходится «поломать голову».

Решить задачу — это значит получить результат. Например, результатом в задаче вычисления корней многочлена K-й степени является K значений переменной, которые при подстановке их вместо переменной обращают многочлен в ноль. Для каждой точно сформулированной задачи всегда известно, что считать результатом.

Процесс решения задачи представляет собой совокупность определенных действий над исходными данными. В большинстве случаев эту совокупность действий программист задает в виде инструкции настолько подробно, что ее исполнение становится механическим процессом. Инструкция может быть пригодна для решения не только конкретной задачи, но и для

решения ряда однотипных задач. Такая инструкция, которая представляет собой *описание способа решения класса однотипных задач*, называется алгоритмом.

Понятие алгоритма. Понятие алгоритма возникло и используется давно. В настоящее время это одно из основных понятий информатики. Широким распространением это понятие обязано идее автоматизации поведения исполнителя-автомата. В ряду всевозможных исполнителей-автоматов ЭВМ является лишь частным, хотя и наиболее впечатляющим примером.

Понятие алгоритма не имеет строгого определения, так как понятие алгоритма — это абстрактное обобщение, основанное не на определенной совокупности существенных признаков конкретного объекта (последовательности конкретных действий), а на совокупности признаков, которые дают нам основание отнести данный объект к числу обозначаемых данным термином. Остановимся на интуитивно-содержательном определении: *алгоритмом* называется конечный набор точных и понятных предписаний (правил, инструкций, команд), позволяющих механически решать конкретную задачу из определенного класса однотипных задач. При этом подразумевается:

- исходные данные могут изменяться в определенных пределах;
- процесс применения предписаний (правил, инструкций, команд) к исходным данным (путь решения задачи) определен однозначно в виде последовательности шагов;
- на каждом шаге известно, что считать результатом.

Основные свойства алгоритма. К свойствам алгоритма обычно относят: дискретность, детерминированность, массовость, результативность, понятность.

Дискретность означает, что путь решения задачи определен в виде последовательности шагов — четко

разделенных друг от друга предписаний (правил, инструкций, команд). Только выполнив одно предписание, можно приступить к выполнению следующего.

Детерминированность означает, что способ решения задачи определен однозначно в виде последовательности шагов. На любом шаге не допускаются никакие двусмысленности или недомолвки.

Массовость означает, что алгоритм применим к целому классу задач, а при решении конкретной задачи из этого класса исходные данные могут меняться в определенных пределах.

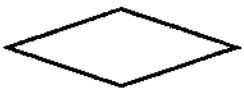
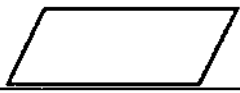
Результативность означает содержательную определенность результата на каждом шаге и в итоге применения всего алгоритма. При этом известно, какой результат должен быть получен через конечное число шагов.

Понятность означает, что алгоритм создается в расчете на определенного исполнителя. В качестве исполнителя может выступать не только человек, но и техническое устройство — автомат, робот, ЭВМ. Для того чтобы исполнитель мог исполнить алгоритм, необходимо, чтобы он мог выполнить каждый шаг предписания.

Совокупность команд, которые могут быть выполнены исполнителем, называют *системой команд исполнителя*. Для правильного построения алгоритма и программы необходимо знать систему команд исполнителя.

Способы записи алгоритмов. Существует несколько способов записи алгоритмов, отличающихся друг от друга наглядностью, компактностью, степенью формализации. Наибольшее распространение получили такие общеизвестные способы, как графический, словесный и в виде программ для ЭВМ.

Таблица 7. Блоки, используемые при графической форме записи алгоритма

Наименование блока	Обозначение блока	Содержание
Процесс		Обработка информации
Принятие решения		Логический блок проверки истинности или ложности некоторого условия
Передача данных		Ввод или вывод информации
Пуск, остановка		Начало или конец программы
Модификация		Организация циклического процесса — заголовок цикла

Графический способ записи алгоритмов предполагает использование определенных графических символов — блоков (см. таблицу 7).

Каждый блок предписывает выполнение определенных действий. Совокупность блоков образует так называемую схему алгоритма, или блок-схему. Пример блок-схемы приведен на схеме 21. Этот способ характеризуется большой наглядностью и может оказаться полезным на начальных стадиях обучения программированию.

Словесная запись алгоритма ориентирована прежде всего на исполнителя-человека и допускает запись предписаний в свободной форме. При этом запись должна быть такова, чтобы человек-исполнитель мог

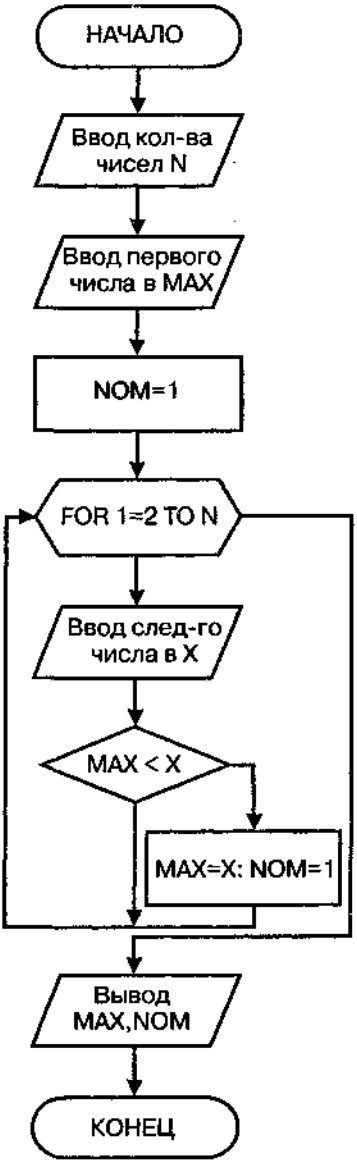


Схема 21. *Нахождение максимального числа и его номера в последовательности из N чисел*

понять суть предписаний и механически (формально) их выполнить. Можно использовать так называемые алгоритмические языки, представляющие собой систему обозначений (кодировки) предписаний и правил их записи.

Основу словаря любого алгоритмического языка составляют *ключевые слова*, смысл и способ употребления которых строго определен. Алгоритмические языки включают в себя и математическую символику: обозначения величин и функций, знаки операций, скобки и т. п. Словесный способ записи алгоритмов характеризуется высокой степенью понятности для исполнителя.

При записи алгоритмов в виде программ для ЭВМ используются языки программирования, объединяющие систему кодирования предписаний и правила их использования. По степени детализации предписаний алгоритма различают языки программирования низкого и высокого уровня.

Языки низкого уровня (типа ассемблеров) «понятны» лишь компьютеру и программистам высокой квалификации, поэтому они называются машинными языками.

Языки высокого уровня более понятны человеку, но всегда запись алгоритмов в виде программ для ЭВМ характеризуется высокой степенью формализации.

Написанию программы может предшествовать представление алгоритма в виде схемы или текста на алгоритмическом языке. Чтобы упростить переход от алгоритма к программе, используют идентичную символику алгоритмического языка и языка программирования. Это означает, что алгоритмический язык — всего лишь версия языка программирования, представляющая собой совокупность ключевых слов на естественном языке.

Объекты алгоритма

Объекты задачи и объекты алгоритма. Решение любой конкретной задачи предполагает наличие реальных объектов. Это и есть объекты задачи.

Например, при решении задачи о начислении зарплаты сотрудникам предприятия объектами задачи могут быть: табельный номер сотрудника, его фамилия, имя, отчество, оклад, отработанное время и т. д. При решении системы уравнений объектами задачи являются: число уравнений, коэффициенты уравнений, правые части уравнений.

Во всех случаях саму задачу тоже можно рассматривать как объект. Объекты разных задач многообразны, и невозможно их как-то классифицировать. Важно отметить, что каждый объект задачи имеет свои характеристики (атрибуты). Фамилии и наименования — это строки символов. А такие объекты, как «коэффициенты уравнений», «количество выпускаемой продукции» — это числовые данные, представленные в виде арифметических выражений или чисел.

Если решается бытовая задача, и ее алгоритм предполагает в качестве исполнителя человека, то она может быть строго не формализована, и в алгоритме могут фигурировать объекты задачи:

1. Взять столовую ложку меда.
2. Налить в стакан 200 г кипяченой воды и т. д.

Если предполагается исполнение алгоритма возложить на автомат, робот или ЭВМ, необходима строгая формализация задачи.

Формализация задачи предполагает замену объектов задачи объектами алгоритма. Каждому объекту задачи должен соответствовать объект алгоритма, который наследует его атрибуты. При разработке алгоритма могут появиться вспомогательные объекты алгоритма, не соответствующие никаким объектам задачи. В практике программирования число базовых типов объек-

тов алгоритмов и, соответственно, программ невелико. Это: константы, переменные, массивы, файлы и некоторые другие объекты.

Понятие константы. Понятие константы — одно из важнейших понятий в программировании. Допустим, что в некоторой задаче в одном из предписаний алгоритма вычисляется длина окружности по ее диаметру. Здесь «длина окружности» (L) и «диаметр окружности» (D) — это объекты задачи.

Зависимость, используемая для вычисления, общеизвестна: $L = \pi D$, где $\pi = 3{,}1415926\ldots$ — величина, постоянная в любой задаче, т. е. π — это константа.

Другой пример: вычисляется количество сотрудников (объект задачи), которые не выполнили план. План — это тоже объект задачи, представляющий собой некоторое число. Если выполнение плана считать в процентах, то для решения задачи необходимо процент выполнения плана каждым сотрудником сравнивать с числом 100. Число «100» в соответствующем алгоритме есть константа.

Константа может быть не только числом. Например, в некотором списке фамилий (фамилия — объект задачи) определяется наличие фамилии «Иванов». В соответствующем алгоритме «Иванов» — символьная константа.

Константа — это объект алгоритма, хотя и не является объектом задачи. Каждая константа как объект алгоритма имеет определенный фиксированный тип (арифметический, символьный или другой) и имеет фиксированное, не изменяемое в данном алгоритме значение, соответствующее ее типу.

Значение константы обычно определено в условии задачи и известно до начала разработки алгоритма. Существуют константы, которые в условии задачи не определяются, так как их значение общеизвестно. К числу таких общеизвестных констант можно отнести

число π, *ускорение свободного падения* g и ряд других. Значения этих констант не всегда определены средой программирования. Об определении числового значения константы в программе должен позаботиться ее разработчик.

Понятие переменной. Переменные — это объекты алгоритма. Пусть в задаче требуется вычислить и напечатать значение некоторой функции при изменении аргумента от заданного начального значения до заданного конечного с заданным шагом. Здесь «заданное начальное значение аргумента», «заданное конечное значение», «заданный шаг изменения аргумента» — объекты задачи, величины которых к моменту разработки алгоритма, т. е. в условии задачи, не определены. Эти значения станут известны по ходу работы программы: введены пользователем с клавиатуры либо получены в результате вычислений. Если не предусмотреть механизма их определения, дальнейшее исполнение программы станет невозможно.

Значение функции в этой задаче вычисляется для каждого значения аргумента и имеет единственное значение. Общее число значений функции зависит от выбранных программистом начального и конечного значений аргумента, а также от шага изменения последнего. Например, если начальное значение — за принять за 1, конечное значение — 100, а шаг изменения аргумента равен 1, то общее число значений функции будет 100. Если шаг изменения аргумента задать равным 0,1, то число вычисляемых компьютером значений функции будет 1000, а если шаг уменьшить до 0,001, то компьютер вычислит 100000 значений заданной функции. Рассмотренные в данном примере переменные — это переменные арифметического типа. Все они — объекты алгоритма.

Другой пример: в задаче анализируются наименования продуктов (это объекты задачи). Наименование

продукта может принимать такие значения: «сахар», «масло» и т. п. Переменная может принимать только одно значение в каждый момент. Наименование продукта — это символьная переменная.

Итак, *переменная* — это объект алгоритма, который имеет определенный фиксированный тип (арифметический, символьный или другой) и который в каждый момент исполнения алгоритма имеет единственное значение соответствующего типа. К моменту использования переменной в алгоритме ее значение должно быть определено. В ходе исполнения программы значение переменной может изменяться.

Имя объекта алгоритма. Каждый объект алгоритма фигурирует в алгоритме под своим именем. Имя объекта в алгоритме неизменно, фиксировано, уникально и часто отличается от имени объекта задачи. Так, имя объекта задачи «начальное значение аргумента», а имя соответствующего объекта алгоритма, например, «NZA»; имя объекта задачи «конечное значение аргумента», имя аргумента может быть «KZA» и т. д.

Имена для переменных, массивов и других объектов устанавливает автор алгоритма. Константы не нуждаются в обозначении, тем не менее, рекомендуется и для них назначать имена. Например, число π можно обозначить «PI».

Имена переменных, констант, массивов и других объектов алгоритма в программировании называют *идентификаторами*. Идентификаторы состоят, как правило, из латинских букв и цифр.

Общепринято, что первым символом в идентификаторе должна быть латинская буква, за которой могут следовать другие латинские буквы, цифры. Буквы русского алфавита в идентификаторах, как правило, не допускаются. В некоторых языках программирования ограничивается длина идентификатора.

Рекомендуется выбирать мнемонические имена, т. е. такие, которые отражают физическую суть соответствующего объекта задачи, например: SUMMA (сумма), FIO (фамилия, имя, отчество), CENA (цена), PLAN (план), FAKT (факт). Такие имена позволяют хотя бы ориентировочно связать объекты задачи и объекты алгоритма, что очень важно для работы с алгоритмами.

Общепринято для обозначения счетчиков использовать однобуквенные имена: I, J, K, L, M, N.

Понятие массива. Так же как константа и переменная, массив — это объект алгоритма. Например, требуется вычислить значение многочлена P (многочлен — объект задачи):

$$P = a + bx + cx^2 + dx^3 + ex^4 + fx^5,$$

где a, b, c, d, e, f — коэффициенты многочлена (тоже объекты задачи арифметического типа). В алгоритме коэффициенты a, b, c, d, e, f — это переменные. Их количество (шесть) определено в условии задачи, т. е. до начала разработки алгоритма.

Во многих случаях разрозненные переменные удобно объединить в совокупность — массив, именуя все коэффициенты общим именем (именем массива) и индексами (номерами в массиве). В рассматриваемом примере введем объект алгоритма — массив A, который состоит из шести элементов: $(A_1, A_2, A_3, A_4, A_5, A_6)$. Здесь $A_1 = a$, $A_2 = b$, $A_3 = c$, $A_4 = d$, $A_5 = e$ и $A_6 = f$. Тогда многочлен P можно представить:

$$P = A_1 + A_2X + A_3X^2 + A_4X^3 + A_5X^4 + A_6X^5.$$

Отметим, что индекс элемента массива позволяет обратиться к элементу, т. е. возможен «прямой» доступ к элементу массива по его индексу. В то же время возможна выборка элементов массива по любому

критерию, благодаря их строгой упорядоченности по индексам.

Таким образом, *массивом* будем называть конечную упорядоченную совокупность данных одного типа, при этом доступ к каждому элементу массива осуществляется по его индексу. В памяти компьютера для массива выделяется единое поле для размещения значений его элементов. Например, для рассматриваемого одномерного массива, содержащего значения коэффициентов многочлена P (массив A):

A_1	A_2	A_3	A_4	A_5	A_6

В задачах используются не только одномерные, но и многомерные массивы, в частности — двумерные. Для индексации элементов двумерного массива указываются два индекса — сначала номер строки, затем номер столбца. Для наглядности такой массив можно представить в виде матрицы:

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$

Здесь представлены оценки трех учеников по четырем предметам массивом из 3-х строк и 4-х столбцов. Имя массива «X».

В памяти ЭВМ и этот массив хранится в виде одномерной последовательности элементов сначала первой строки, потом второй и затем третьей. Но это не означает, что при использовании двумерного массива можно указывать для его элементов один индекс.

Возможны не только числовые (арифметические) массивы, но и массивы других типов. В частности — символьные. Например, список фамилий (фамилии — это объект задачи) можно рассматривать как одномерный символьный массив.

В программировании для обозначения элементов массивов (индексированных переменных) их индексы указываются в скобках, в Basic'е для этого используют круглые скобки. Если индексов несколько, то они разделяются запятыми. Например, $A(1)$; $A(2)$; $X(1, 1)$; $X(2, 3)$.

В качестве индексов можно использовать не только константы, но и переменные и даже выражения. Например, $A(K)$; $A(I+J)$; $X(I, J)$; $X(I+2, J)$. Очень важно помнить, что переменные, используемые для индексации, к моменту обработки элементов массивов должны быть определены.

Основные элементы алгоритма

Основные элементы алгоритма. В любом алгоритме (программе) всегда присутствует раздел операторов. При этом независимо как от формы записи алгоритма, так и от языка программирования раздел операторов имеет одну точку входа (НАЧАЛО) и одну (в отдельных языках — несколько) точку выхода (КОНЕЦ). В остальном разделы, входящие в состав алгоритма, могут существенно различаться в зависимости от используемого языка. В качестве примера можно назвать разделы объявлений (описаний) пользовательских типов и имен объектов программы, заголовки и т. д.

Назначение раздела объявления имен и типов объектов алгоритма очевидно из его названия. Например, в языке Pascal этот раздел включает объявление констант, типов, переменных и массивов.

В алгоритмическом языке Basic часто ограничиваются только объявлением массивов. Для объявления переменных широко используется принцип «по умолчанию»: переменная автоматически «объявляется» своим появлением в разделе операторов, а ее тип указывается специальным символом, который приписывается к идентификатору.

Ключевым словом «НАЧАЛО» открывается раздел операторов, образующий тело алгоритма. В некоторых алгоритмических языках это ключевое слово может отсутствовать.

Тело алгоритма содержит инструкции по обработке объектов алгоритма. Конец алгоритма идентифицируется ключевым словом «КОНЕЦ» («END»).

В разных языках программирования используются различные правила записи программы: в одних — каждое «предложение» записывается в отдельной строке, в других — через двоеточие и т. п. В некоторых версиях языка Basic, например, строки программы нумеруются, в других — нет.

Понятие оператора. Закодированная форма инструкции, несущая определенный смысл, называется *оператором*. В любом алгоритмическом языке и в любом языке программирования используется определенное число инструкций (операторов). Каждый оператор имеет установленную форму записи (формат оператора) и предполагает выполнение определенных действий.

Различают простые и составные операторы. К числу простых операторов относятся: оператор присваивания, оператор перехода, операторы ввода и вывода данных. В группу составных операторов входят: условные операторы, операторы цикла и другие.

Естественно, что формы записи операторов в различных языках программирования различны, но смысловые нагрузки конструкций, как правило, идентичны. Например, оператор присваивания в различных языках выглядит по-разному:

Basic	Pascal	C++
=	:=	=

Комментарий в программе. В алгоритмах и программах важную роль играют комментарии. Програм-

мисты говорят, что ничто не заслуживает большего порицания, чем алгоритм и программа без комментариев. Комментарии повышают понятность и наглядность алгоритма и программы, позволяют их документировать и облегчают отладку. В качестве комментария можно использовать любой текст, поясняющий строку программы или ее блок.

Обычно комментарии могут быть записаны в любом месте программы. Строка программы или часть строки с пометкой «Комментарий» компьютером не исполняется и служит исключительно для человека, читающего и отлаживающего программу. Форма обозначения комментария в программе различна. В одних языках комментарий заключается в специальные скобки, например, `{...}`, `(...)`, `*...*`, `/*...*/`, в других — открывается ключевым словом, например, REM. В ряде языков применяются однострочные комментарии. Признаком однострочного комментария является определенный символ или набор символов, например, ' (апостроф) или `//`, а далее (до конца строки) следует текст пояснения.

Комментировать в алгоритме и программе в первую очередь необходимо раздел объявлений. В разделе операторов целесообразно выделять этапы решения. Не имеет смысла комментировать отдельные операторы (указывая, например, что это оператор присваивания или вывода), так как тип оператора и его функции очевидны для каждого программиста.

Базовые алгоритмические конструкции. Любой алгоритм может быть реализован в виде комбинации трех базовых алгоритмических конструкций: линейной, выбора, циклической (см. схему 22).

Линейная структура (а) — это группа выполняемых друг за другом в естественной последовательности блоков или операторов *S*.

Структура выбора (б) обеспечивает при каждой

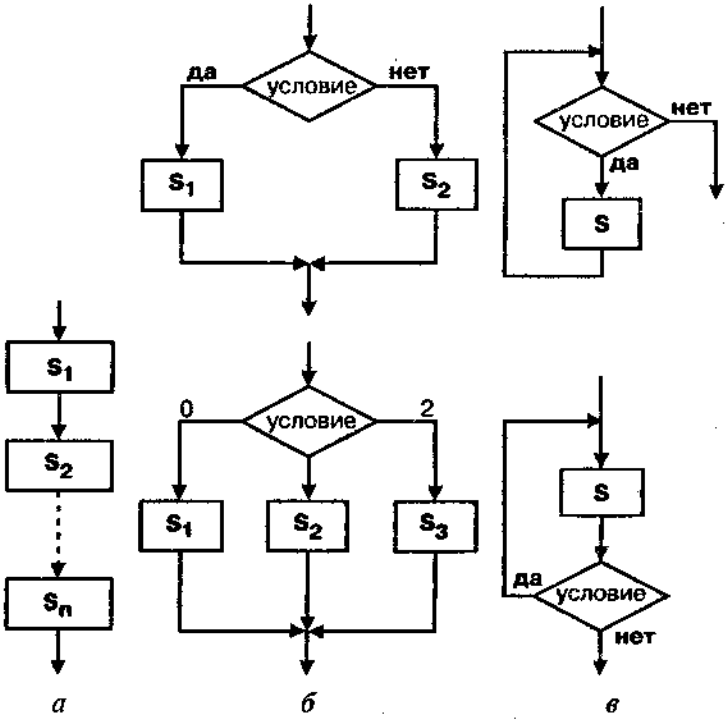


Схема 22. *Базовые алгоритмические конструкции: а — линейная; б — выбора; в — циклическая*

своей реализации выполнение, в зависимости от условий, только одного из содержащихся в ней функциональных узлов или блоков *S*.

Циклическая структура (в) обеспечивает повторяющуюся реализацию содержащегося в ней функционального узла *S* при выполнении условия.

Как видно из схемы 22, все конструкции имеют общее свойство: один вход и один выход. Именно благодаря этому свойству на их основе можно создавать наглядные алгоритмы, уменьшая тем самым вероятность ошибок при решении сложных информационных задач.

Допускается неограниченное соединение структур и их вложение друг в друга, что позволяет проектировать сложные алгоритмы, последовательно детализируя функциональные блоки, входящие в какую-либо конструкцию.

В различных алгоритмических языках и языках программирования используются конструкции, реализующие базовые структуры в той или иной модификации. Линейной структуре соответствует составной оператор или блок, структуре выбора — условные операторы, циклической структуре — операторы циклов.

Этапы разработки алгоритма

Этапы разработки алгоритма — это этапы решения информационной задачи. Разработка любого алгоритма состоит из нескольких взаимосвязанных этапов. На каждом из них решаются свои специфические задачи, определяющие в конечном счете общий результат.

Анализ условия задачи. Этап анализа условия задачи — это основополагающий этап ее решения. Именно на этом этапе определяются объекты задачи, необходимые для разработки алгоритма.

Главная цель анализа — понять задачу и правильно выделить все ее объекты. Если совершена ошибка на этом этапе, т. е. неверно определены и обозначены объекты задачи, вся последующая работа становится бессмысленной.

Последовательность анализа такова. Во-первых, необходимо в условии задачи выделить и осмыслить все исходные данные как *объекты задачи*. К объектам задачи относятся также промежуточные и окончательные результаты. Во-вторых, необходимо определить, какие типы объектов алгоритма будут представлять в этом алгоритме выделенные и проименованные объекты задачи. В-третьих, необходимо сформулировать

вопрос задачи в виде одного короткого, конкретного предложения. Чтобы предложение получилось коротким, необходимо в условии задачи «выбросить» все уточняющие детали. Однако эти детали необходимо зафиксировать где-то в «стороне». Процесс анализа условия задачи включает в себя: составление *макета исходных данных, макета печати результатов* и *таблицы идентификаторов*. Эти действия предполагают детальное изучение задачи, разложение ее «по полочкам», консультации с постановщиком задачи (заказчиком). Цель консультаций — уточнение всех вопросов, данных, увязка противоречивых требований и т. п. Основной тип вопросов, возникающих и ставящихся при анализе, должен быть: «Каковы объекты задачи?», «Сколько их?», «Каковы их имена?», «Каковы отношения между объектами задачи?», «Какими математическими выражениями можно их отразить ?» и так далее.

Очень важно в процессе анализа фиксировать все уточняющие детали и «напрашивающиеся» шаги решения.

Например: эти объекты задачи одного типа, их всего 10. Следовательно, они могут быть представлены одномерным массивом. Результатов много (сколько, какого типа?). Необходим цикл (какого типа?). Исходных данных много. В условии задачи их количество не определено. Поэтому для ввода данных и их обработки нужен итерационный цикл, параметры которого выбираются программистом.

Макет исходных данных. Макет исходных данных — это форма представления исходных данных с выделением начальных значений и типов объектов. Целесообразно указывать соответствующие «заголовки», которые в дальнейшем будут служить «подсказками» в алгоритме и комментариями в программе.

При составлении макета удобно сразу же назначать имена объектам задачи и объектам алгоритма и записывать их на макете.

Очень важно установить факт возможности объединения данных в массивы. Использование массивов может существенно упростить алгоритм решения задачи.

Макет печати результатов. Макет печати результатов — это форма выходного документа. В нем должны быть предусмотрены необходимые поясняющие тексты, а также фамилия исполнителя, дата исполнения. Макет печати результата часто зависит не только от условия задачи, но и от вкуса и знаний программиста.

В тех случаях, когда макет печати в условии задачи не оговорен, его форма и содержание определяется программистом, который создает форму выходного документа максимально удобной для пользователя.

Таблица идентификаторов. Таблица идентификаторов должна содержать возможно более полную информацию об объектах задачи и объектах алгоритма. Можно рекомендовать форму, представленную в таблице 8.

Таблица 8. Таблица идентификаторов

Объект задачи	Объект алгоритма	Идентификатор	Атрибуты
Задача	Алгоритм	SREDV	—
Возраст	Переменная	V	Целое
...	...	...	...

«Заготовка» таблицы делается в начале анализа условия задачи. Она не закрывается до завершения решения задачи, так как в любой момент составления алгоритма таблица может быть дополнена.

Таблица идентификаторов служит основой для записи раздела объявлений в алгоритме, а также для удобства при составлении программы.

Метод пошаговой детализации алгоритма. В процессе анализа условия задачи необходимо дать ее лаконичную формулировку в виде одного предложения. Что это за предложение и какую роль оно играет в решении задачи? Ответ состоит в том, что это предложение, по сути, есть алгоритм решения задачи, записанный в «системе команд» очень высокого уровня. В этой системе есть всего лишь одна «команда», выполнение которой обеспечивает решение задачи.

Например, в результате анализа условия некоторой задачи ее сформулировали: «Построить дом».

Безусловно, в исходной постановке задачи указаны многие «мелочи»: число этажей, характеристики комнат на этажах и другие параметры объектов задачи. Эти «мелочи» должны быть зафиксированы в *макете исходных данных*, *макете результатов*, в *таблице идентификаторов* в виде объектов алгоритма.

Если конкретизировать алгоритм или, другими словами, сформулировать его в системе команд более низкого уровня, то появятся дополнительные команды.

Например:

1. *Поставить фундамент.*
2. *Подвести коммуникации.*
3. *Построить этажи.*

. . . . . .

n. Сделать кровлю.

Если этажи все одинаковые, то возможно, например, следующее уточнение команды: «Построить этажи». *Выполнить К раз: «Построить этаж».* Это уже процесс детализации алгоритма.

Процесс детализации алгоритма можно продолжать, записывая его каждый раз в системе команд все

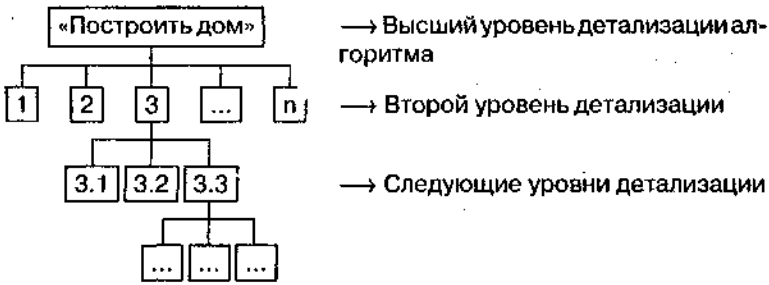


Схема 23. Дерево уровней детализации алгоритма

более низкого уровня. Схематически его можно представить в виде дерева уровней детализации, пример которого приведен на схеме 23.

На каждом уровне алгоритм «расширяется» по мере добавления и уточнения деталей. Уровень команд (инструкций) постепенно понижается. Весь процесс заканчивается тогда, когда будет достигнут уровень системы команд непосредственного исполнителя алгоритма.

Такой метод разработки алгоритмов называется *методом пошаговой детализации*, или нисходящим проектированием (сверху-вниз от системы команд высокого уровня к системе команд низкого уровня).

Рекомендации к использованию метода пошаговой детализации:

1. Не спешите заниматься «мелочами». За один шаг делайте только небольшие «расширения». Не вдавайтесь в детали слишком рано, концентрируйте внимание прежде всего на самом существенном.

2. Тщательно взвешивайте каждый шаг детализации. Пытайтесь понять последствия того, что делаете.

3. Внимательно следите за данными и изменениями их значений. По мере детализации могут появляться промежуточные (дополнительные, вспомогательные) данные — объекты алгоритма. Постоянно фик-

сируйте все данные в таблице идентификаторов.

4. Будьте готовы отменить принятые решения. Попытайтесь реализовать другой вариант на одном или нескольких уровнях детализации.

Пример анализа условия задачи. Рассмотрим все эти этапы пошаговой детализации на конкретном примере.

Задача 3.1. Деятельность предприятий города характеризуется 8-ю показателями. Число предприятий не задано. Требуется определить предприятие, имеющее наибольшее значение одного из показателей. Показатель задан.

Решение.

1. *Анализ условия задачи.*

Очевидно, что условие задачи следует уточнить и конкретизировать.

Что означает уточнить и конкретизировать?

Это значит, выявить все объекты задачи и уточнить их имена.

О чем идет речь в задаче? — О предприятиях. Следовательно, «предприятие» — это объект данной задачи.

Сколько таких объектов? — не задано. Поэтому мы конкретизируем второй объект задачи: «число предприятий», обозначим число предприятий латинской буквой N. N — это переменная, объект алгоритма, который соответствует объекту задачи «число предприятий».

Каждое предприятие должно быть представлено в памяти компьютера именем этого предприятия, т. е. его названием, например: «Зенит», «Рубин», «Москвич» и т. д. Можно объединить данные в массив (поскольку они однотипные), что существенно упростит алгоритм решения этой задачи. Массив данных — это объект алгоритма. В данном случае мы имеем дело с

Таблица 9. Заготовка для таблицы идентификаторов

Объект задачи	Объект алгоритма	Идентификатор	Тип, значение

массивом символьных переменных. Массив одномерный. Ввод имен предприятий будет осуществляться в цикле с числом повторений *N*. Пора заготовить таблицу идентификаторов, которую будем постепенно заполнять по мере анализа условия задачи в процессе построения макета исходных данных (см. таблицу 9).

Пользователь программы должен будет ввести число предприятий *N* при исполнении программы по ее «запросу», после чего будут в цикле «запрошены» все имена предприятий. Поскольку в условии задачи имена предприятий не заданы, то пользователь введет либо имена действительно существующих предприятий, либо условные имена, например: «P1», «P2», ..., «PN». Важно помнить, что имена предприятий — это переменные символьного типа.

Продолжаем анализировать условие задачи. В задаче требуется определить предприятие, имеющее наибольшее значение одного из 8-ми показателей. Этот показатель (один из восьми) задан. В таблицу идентификаторов впишем новые выявленные объекты задачи: «показатели, характеризующие работу предприятия», «общее число показателей», «заданный показатель». Значения констант (в данной задаче это число показателей, характеризующих работу предприятия и номер заданного показателя) должно быть определено в условии и известно до начала разработки алгоритма: в нашем случае $M = 8$, $ZADAN = 3$. Номер заданного показателя *ZADAN* мы выбрали условно и присвоили ему значение «3». Можно поступить и иначе. Считать заданный показатель не константой,

а переменной, значение которой — запрашивается программой в начале ее исполнения. Тогда пользователь может по своему желанию выбирать показатель, наибольшее значение которого послужит критерием выбора победителя, т. е. лучшего предприятия. В таблицу идентификаторов следует записать еще два объекта задачи — «имя лучшего предприятия» и «значение заданного показателя» — *REZ*. Имя лучшего предприятия (*NAME*) — это символьная переменная, которую следует «запомнить». Можно запомнить и вывести на экран значение заданного показателя для лучшего предприятия (*REZ*). Таким образом, мы выявили и конкретизировали все объекты задачи, наметили в таблице идентификаторов объекты алгоритма, определили их имена и имена идентификаторов, определили типы переменных и констант. Мы помним, что таблица идентификаторов остается «открытой» до завершения решения задачи.

Теперь мы можем представить макет исходных данных нашей задачи в виде таблицы (см. таблицу 10).

В задаче не определена форма представления результата, поэтому результаты напечатаем в виде справки о лучшем предприятии, которую оформим, например, так:

```
СПРАВКА О ЛУЧШЕМ ПРЕДПРИЯТИИ
Заданный показатель: 3 - ZADAN
Лучшее предприятие: **** - NAME
Значение показателя: **** - REZ
Исполнил: Ф.И.О. исполнителя - FIO
Дата: **** - DAT
```

Отметим, что появились новые объекты: Ф.И.О. исполнителя (FIO) и дата расчета справки (DAT). Внесем их тоже в таблицу идентификаторов.

Наконец, попытаемся кратко сформулировать суть задачи, т. е. сформулировать ее в системе команд самого высокого уровня. Один из возможных вариантов:

«Определить лучшее предприятие».

Таблица 10. Таблица идентификаторов на стадии анализа условия задачи

Объект задачи	Объект алгоритма	Идентификатор	Тип, значение
Количество предприятий	Переменная	*N*	Целый
Названия предприятий	Одномерный массив из *N* элементов	*P*$	Символьный
Количество показателей	Константа	*K*	Целый, 8
Значения показателей	Двумерный массив из (*N*, *K*) элементов	*PRP*(*N*, *K*)	Вещественный
Заданный показатель	Переменная	*ZADAN*	Целый
Максимальное значение заданного показателя	Переменная	*REZ*	Вещественный
Название лучшего предприятия	Переменная	*NAME*$	Символьный
...	...	...	...

2. *Детализация алгоритма*

Как мы помним, детализация алгоритма — это формулирование его в системе команд более низкого уровня. На каждом более низком уровне алгоритм «расширяется». При анализе постановки задачи мы уже установили первый уровень детализации:

Ввод заданного показателя (ZADAN).
Цикл ввода и обработки данных о предприятиях.
Вывод результатов.

Ввод заданного показателя — инструкция, не требующая уточнений. Вывод результатов реализуется линейной последовательностью тоже простых инструкций.

Для детализации более низкого уровня остается один этап: цикл ввода и обработки всех данных о предприятиях. Здесь может быть реализовано несколько подходов. Первый подход — ввести число предприятий N и обработать данные в цикле с наперед заданным числом повторений. Второй подход, более интересный с нашей точки зрения, — это цикл с заранее неизвестным числом повторений. Что является условием повторения действий? Очевидно, наличие данных о предприятиях. Поэтому можно сделать такое уточнение этапа:

Пока есть данные о предприятиях, выполнить:
Ввод данных об очередном предприятии
Обработать данные.

Ввод данных об очередном предприятии можно разбить на две операции:

Ввод названия предприятия
Ввод значений показателей.

В свою очередь, обработка данных может быть конкретизирована:

Обработать значение данного показателя.

Итак, получили очередной шаг детализации:

Пока есть данные о предприятиях, выполнить:
Ввод названия предприятия
Ввод значений показателя
Обработать значение заданного показателя.

Однако пока не ясно, как реализовать проверку окончания ввода данных.

Можно, например, принять, что ввод названия предприятия в виде пробела будет означать исчерпание исходных данных. Тогда заголовок цикла примет вид:

Пока (NAZV <> «пробел») выполнить:

Это заставит «перекроить» выполняемые действия следующим образом:

Ввод названия предприятия (NAZV).
Пока (NAZV <> «пробел») выполнить:
Ввод значений показателей (POKAZ).
Обработать значение заданного показателя.
Ввод названия предприятия или «пробела».

Ввод значений показателей — это ввод элементов массива, следовательно, для этого необходим цикл с известным числом повторений:

Для I = от 1 до 8 выполнить:
Ввод POKAZ(I).

Отметим, что появился новый объект алгоритма — счетчик *I*, запишем его в имеющуюся таблицу идентификаторов (см. таблицу 11).

Ввод названия предприятия и элемента массива — это инструкции, не требующие детализации. Остается уточнить последнюю:

Обработать значение заданного показателя.

Из постановки задачи следует, что эта обработка есть нахождение наибольшего значения показателя, т. е. его сравнение с другими:

Таблица 11. Данные, внесенные в таблицу идентификаторов в ходе разработки алгоритма

Объект задачи	Объект алгоритма	Идентификатор	Тип/значение
...	...	...	...
Ф.И.О. исполнителя	Переменная	*FIO$*	Символьный
Дата расчета справки	Переменная	*DAT$*	Символьный
Счетчик	Переменная	*I*	Целый
...	...	...	...

Если (POKAZ (ZADAN) > REZ) тогда:
REZ = POKAZ (ZADAN).
NAME=NAZV.

И окончательно можно записать:

Ввод заданного показателя (ZADAN).
Ввод названия предприятия (NAZV).
Пока (NAZV <> «пробел») выполнить:
Для I=от 1 до 8 выполнить:
Ввод POKAZ(I).
Если (POKAZ (ZADAN) > REZ) тогда:
REZ = POKAZ (ZADAN).
NAME=NAZV.
Ввод названия предприятия или «пробела».
Вывод результатов.

Формальное исполнение алгоритма. Полученный текст еще нельзя считать завершением работы над алгоритмом, так как следует проверить, все ли требования, предъявляемые к алгоритмам, выполняются:

1) *Дискретность* — составленный текст есть последовательность конкретных предписаний.

2) *Массовость* — мы не накладываем никаких ограничений на исходные данные.

3) *Результативность* — мы знаем, что конкретно получится в итоге исполнения алгоритма.

4) *Понятность* — допускаем, что текст понятен исполнителю (читателю), так как алгоритм разработан с ориентацией на вполне определенного исполнителя — человека — и в него включены команды только из системы команд данного исполнителя.

5) *Детерминированность* — путь решения определен вполне однозначно. Но, с учебной целью, позволим себе усомниться: так ли это? Для проверки следует осуществить формальное исполнение алгоритма. При этом действуем строго механически, педантично следуя инструкциям. Результаты исполнения каждого шага инструкций будем записывать справа от текста алгоритма в соответствующих строках.

В подавляющем большинстве случаев формальное исполнение алгоритма позволяет не только обнаружить допущенные ошибки, но и определить пути их исправления. Так, в рассматриваемом примере для первого же предприятия оказывается невозможным выполнить сравнение *POKAZ(ZADAN)>REZ* по той причине, что значение *REZ* не определено. Ошибку можно исправить, принимая перед началом вычислений:

REZ=0.

Читателю предлагается определить место этой инструкции в алгоритме и убедиться в правильности, осуществив повторно формальное исполнение.

Также самостоятельно предлагается предусмотреть в алгоритме одновременный поиск не только наибольшего значения показателей, но и наименьшего, т. е. предприятия с более низким уровнем, например, себестоимости продукции.

Начала программирования на языке Basic

Введение

Язык программирования — это система команд, «понятных» ЭВМ. Разделяют языки низкого и высокого уровня. Первые (в качестве примера можно назвать Ассемблер) позволяют управлять вычислительным процессом напрямую, при помощи машинных команд. Написание программ на языках низкого уровня — процесс сложный и трудоемкий: это удел узкого круга специалистов. Языки высокого уровня требуют для общения с машиной «переводчика» — транслятора, но процесс программирования при этом существенно упрощается. В настоящее время используют большое количество высокоуровневых языков в различных версиях, например, Basic, Pascal, C, C++, Perl и т. д. Каждый из них имеет свои особенности, но при этом общие принципы программирования на разных языках одни и те же. Освоение одного из языков облегчает изучение других.

В настоящем разделе пособия изложены начала программирования на языке Basic. Подобный выбор продиктован тремя обстоятельствами.

Во-первых, этот язык достаточно прост в освоении, как и следует из его названия — Beginner's All-purpose Simbolic Instruction Code — универсальный язык символических инструкций для начинающих.

Во-вторых, этот язык рассматривается в курсе основ информатики и вычислительной техники во многих средних учебных заведениях.

В-третьих, он является родоначальником таких мощных современных языков программирования для среды Windows, как VB (Visual Basic) и VBA (Visual Basic For Application), что позволит освоившему язык Basic быстро перейти на качественно иной уровень програм-

мирования, не отвлекаясь на освоение нового синтаксиса.

Существует множество версий языка Basic разного уровня сложности и совершенства, созданных для ЭВМ различных типов (QBasic, GWBasic, MSX-Basic, HBasic, Visual Basic, VBA и др.). Подчас данные версии несовместимы, но, несмотря на это, все они имеют общий фундамент — совокупность основных операторов и систему команд, которые и рассмотрены в настоящем разделе.

Структура программы на языке Basic

Строки программы. Программа на языке Basic состоит из последовательности строк. Каждая строка содержит номер* и один или несколько операторов.

Номера строк (далее «нс») — это целые десятичные числа от 1 до 32767. Номер используется для:

— идентификации строк;

— установления очередности выполнения строк при запуске программы;

— обращения к операторам строки.

Первоначально строки нумеруют с шагом 10, чтобы иметь возможность вставлять в программу новые строки без перенумерации всех строк программы.

Если операторов в строке несколько, их отделяют друг от друга двоеточием. Операторы выполняются в порядке их записи слева-направо.

Комментарии. В программе может присутствовать текст, необходимый для пояснения имен объектов программы и этапов алгоритма. Такой текст на естественном (русском, например) языке и есть комментарий. Для введения комментария в программу используется специальный оператор — REM. Формат оператора:

*В ряде версий строки не нумеруются.

нс REM <текст комментария>.

Здесь:

REM — ключевое слово (название оператора);

<текст комментария> — произвольный текст, поясняющий программу.

Заключать текст в кавычки необязательно, так как оператор REM — неисполняемый оператор.

Вместо оператора REM можно применять апостроф. Апостроф не является оператором, поэтому двоеточие при его использовании не ставится.

Примеры комментариев:

```
10 REM Программа вычисления частного
20 A = 21 : REM Делимое
30 B = 7 ' Делитель
40 ' Этап вычислений
```

Отсутствие комментариев в программе считается дурным тоном.

Начало и конец программы. В Basic'е не обязательно определять заголовок программы, но делать это рекомендуется. Специального оператора для этого нет, но можно воспользоваться знаком комментария. Например:

```
10 REM Программа определения MAX
```

Выполнение программы начинается со строки с наименьшим номером и заканчивается строкой, содержащей оператор END. Формат этого оператора:

нс END.

При его достижении исполнение программы заканчивается. Следующие строки, если они есть, игнорируются. Указывать конец программы желательно, но не обязательно: если конец программы не указан, то выход из программы происходит по исполнении всех ее строк.

Алфавит. *Алфавитом языка программирования* называется совокупность букв, цифр и символов, употребляемых для записи конструкций языка.

Алфавит языка Basic включает:

— десятичные цифры от 0 до 9;

— строчные и прописные буквы латинского алфавита от A до Z;

— строчные и прописные буквы кириллицы от А до Я;

— знаки и символы (+ - = * / ? < > $ % @ # & ^ : ; пробел, скобки и др.).

Основой языка является латинский алфавит. Буквы его, наряду с цифрами, входят в состав имен (идентификаторов) объектов программы. Кроме того, из них состоят ключевые слова языка — операторы, имена встроенных функций, названия ряда операций и прочие управляющие элементы. Вот некоторые из ключевых слов:

DIM	THEN	NEXT
REM	ELSE	WHILE
END	GOTO	MOD
INPUT	ON	NOT
PRINT	FOR	AND
IF	TO	OR

Их назначение поясняется ниже.

Русские буквы (кириллица) используются только в комментариях (без кавычек) и в символьных константах (в кавычках).

Константы в языке Basic. *Константой* называется величина, не изменяющая своего значения в ходе выполнения программы. В Basic'е различают арифмети-

ческие (цифровые) и символьные константы*. В свою очередь, арифметические константы могут быть целыми и вещественными.

Тип объекта программы, и константы в том числе, наследуется от объекта задачи (см. раздел «Объекты алгоритма»).

Целая константа — это целое десятичное число, которое может иметь знак (плюс или минус) и находится в диапазоне от –32768 до +32767. Целочисленная константа не должна содержать десятичной точки.

Вещественная константа — это десятичное число, которое может иметь знак и дробную часть. Дробную часть от целой отделяют десятичной точкой.

При записи вещественных констант используют 2 формы:

1. Форма с фиксированной точкой (естественная форма). Например:

5.6 0.825 –34.1931 261.00 + 1.67

2. Форма с плавающей точкой (экспоненциальная форма). Например, числа $-4.58 \cdot 10^{-3}$ и $0.872 \cdot 10^{8}$ в программах записываются, соответственно, как –4.58E–3 и +0.872E + 8.

Знаки «+» перед мантиссой и порядком можно опускать.

Диапазон допустимых значений для вещественных чисел находится в пределах от –E+38 до +E+38. Диапазон и точность определяются форматом (см. Арифметические основы ЭВМ).

Символьная константа — это произвольная последовательность допустимых символов языка (не

*В более поздних версиях имеются также константы других типов, в частности, логического.

Таблица 12. Примеры идентификаторов

Идентификаторы					Тип объекта алгоритма
PLAN	Sum2	a	X	Y	Вещественный
flag%	KOL%	i%	j%	N%	Целый
FIO$	name$	str$	lex$	s2$	Символьный

исключая пробел). Символьные константы в программах заключаются в верхние кавычки. Примеры символьных констант:

"BASIC" "Петров-Водкин" "14.07.01"

"ivanov@mail.ru" "С Новым Годом!"

Идентификаторы. Идентификаторы — это имена, которые программист назначает объектам программы*. Правила составления идентификаторов:

1. Идентификаторы могут содержать латинские буквы, цифры и некоторые символы и должны начинаться с буквы.

2. В конце идентификатора может быть суффикс — символ «$» или «%»**.

Последний символ идентификатора содержит информацию о типе объекта: имена объектов символьного типа заканчиваются суффиксом «$», целого — суффиксом «%», буква или цифра в конце имени — признак вещественного числа***.

Примеры идентификаторов приведены в таблице 12.

При отсутствии суффикса «%» в имени целой переменной сообщения об ошибке не последует, и в подавляющем большинстве случаев на исполнении программы это не отразится.

*В более сложных версиях также процедурам, функциям и т. д.

**В ряде версий используются также символы !, #, &, @.

***В Visual Basic тип объекта определяется явно и использование суффиксов не обязательно.

Имена объектов программы (идентификаторов) должны быть удобочитаемы и мнемоничны (отражать сущность объекта).

Объявление объектов программы. Каждый объект программы требует определенного места в оперативной памяти. Объем требуемой памяти зависит от типа объекта программы. Цель объявления — выделение места в памяти под конкретный объект программы в соответствии с его типом.

В Basic'е нет специальных разделов описания (объявления) объектов программы. Объявлять их можно в любой строке программы, но непременно до первого использования. Поскольку информация о типе объекта содержится в его имени, простые переменные в особом объявлении не нуждаются: память для них будет выделена при первом упоминании их имени в программе (объявление непосредственно предваряет первое использование).

Массивы необходимо объявлять, хотя тип их элементов, как и в случае простых переменных, следует из их идентификатора. Для того чтобы компьютер мог выделить нужное количество памяти, ему надо сообщить, сколько будет в массиве элементов. Для объявления массива используется оператор DIM*:

нс DIM <список объектов>.

Здесь:
DIM — ключевое слово, оператор описания (объявления);
<список объектов> — один или несколько объявляемых массивов, разделяемых запятыми.

*В Visual Basic с помощью этого оператора объявляют и простые переменные.

При объявлении массива указывают его имя и — в круглых скобках — число содержащихся в нем элементов*. Если объявляется многомерный массив, число элементов по каждому измерению указывают в круглых скобках через запятую.

Рекомендуется объявлять массивы не списком, а по одному в одной строке программы, поясняя в комментарии, что это за объект задачи. Примеры объявления массивов:

```
20 DIM FAM$(30)          ' Список учащихся
30 DIM VIPUSK(10,4)      ' Поквартальный выпуск продукции завода
40 DIM STUD$(25), OCEN%(25,5)    ' Информация о студентах
```

В строке 20 примера объявлен одномерный символьный массив, в строке 30 — двумерный массив вещественных чисел, в строке 40 — два массива: одномерный символьный и двумерный целочисленный.

Операции

Любой язык программирования располагает строго определенным набором операций для работы с данными. Операции являются основой вычислительного процесса.

Арифметические операции (в порядке убывания приоритета):

^	возведение в степень;
*	умножение;
/	деление;
\	целочисленное деление;
MOD	получение остатка от деления;
+	сложение;
-	вычитание.

*В зависимости от версии индексация может начинаться с нуля или единицы. В случае индексации от нуля количество элементов в массиве будет на 1 превышать число, указанное при объявлении. В настоящем пособии принято, что индексация начинается с единицы.

Таблица 13. Действие операций \, / и MOD

A	B	A / B	A \ B	A MOD B
33	7	4.7142	4	5
-33	7	-4.7142	-4	-5
-21	-8	-2.625	-2	-5
21	-8	-2.625	-2	+5
2	10	0.2	0	2

Действие операций /, \, MOD видно из примеров, приведенных в таблице 13.

Операндами в операциях арифметического типа могут быть только числа.

Операции отношения:

= равно;
<> не равно;
> больше;
< меньше;
>= больше или равно, не меньше;
<= меньше или равно, не больше.

Если отношение ложно, результат операции «0», если истинно «–1».

Операции отношения используются при сравнении данных: двух чисел либо двух строк. Сравнить число со строкой нельзя.

Логические операции (см. Логические основы ЭВМ):

NOT логическое отрицание;
AND и, логическое умножение;
OR или, логическое сложение.

Логические операции часто используются при необходимости одновременной проверки нескольких условий. Символьные данные операндами в этих операциях быть не могут.

При наличии в одном выражении нескольких операций выполняться они будут не подряд, а в определенной последовательности, зависящей от приоритета

(старшинства) операций. В первую очередь выполняются операции с самым высоким приоритетом, далее — в порядке убывания.

Самым высоким приоритетом обладают арифметические операции, далее следуют операции отношения, в последнюю очередь выполняются логические операции.

Естественная последовательность выполнения операций может быть изменена при помощи круглых скобок: скобки раскрываются в первую очередь. Внутри скобок выражения вычисляются в порядке старшинства. Операции одного приоритета выполняются слева-направо (за исключением возведения в степень, выполняемого справа-налево).

Операции, используемые для работы с символьными данными, рассмотрены в соответствующем разделе.

Выражения в языке Basic

Выражения — лексические единицы программы, которые могут содержать константы и идентификаторы, связанные между собой знаками операций и скобками. В выражения могут также входить функции.

Переменные и константы — частные случаи выражения.

Выражения вычисляются (имеют некоторое вычисляемое значение). Они входят в качестве составных частей в операторы.

В Basic'е различают арифметические, символьные и логические выражения. Все элементы, входящие в выражение, должны быть совместимы. Например, запись

```
N+abc$
```

ошибочна: как видно из идентификаторов, производится попытка сложения числа со строкой.

Таблица 14. Примеры записи выражений в программе

Выражение	Запись выражения в программе
$7x-4y$	7*x-4*y
$ax^2 + bx + c$	a*x^ 2+b*x+c или a*x*x+b*x+c
$ab : cd$	a*b/c/d или a*b/(c*d)
$x : (-k)$	x/(-k)

Все арифметические типы данных совместимы между собой. Тип вычисляемого значения соответствует типу элементов выражения. Если выражение содержит элементы разных арифметических типов, результат будет относиться к наивысшему из этих типов.

Например:

A+B%.

Здесь: A — вещественное, B — целое, результат — вещественного типа.

Выражения вычисляются в соответствии с приоритетом входящих в них операций и с учетом круглых скобок. При наличии в выражении функций функции вычисляются прежде всего.

Примеры записи выражений в программе приведены в таблице 14.

Встроенные функции

Как и другие языки программирования, Basic содержит набор встроенных (стандартных) функций. Для обращения к функции указывают ее имя и аргумент, заключаемый в круглые скобки (аргументов может быть и несколько).

Например:

```
70 x = RND(1)
80 y = LOG(x)
```

Таблица 15. Стандартные функции языка Basic

Функция	Тип результата	Назначение
SIN(x)	Вещественный	Вычисление sin(x)
COS(x)	Вещественный	Вычисление cos(x)
TAN(x)	Вещественный	Вычисление tg(x)
ATN(x)	Вещественный	Вычисление arctg(x)
LOG(x)	Вещественный	Вычисление натурального логарифма
SQR(x)	Вещественный	Вычисление квадратного корня из x
ABS(x)	Вещественный	Вычисление абсолютного значения x
EXP(x)	Вещественный	Вычисление функции e^x
SGN(x)	Целый: 1, 0 или −1	Вычисление знака аргумента
FIX(x)	Целый	Вычисление целой части аргумента
INT(x)	Целый	Округление аргумента до целого значения
RND(x)	Вещественный	Датчик случайных чисел в диапазоне от 0 до 1

Каждый аргумент имеет вполне определенный тип; тип указанного в скобках аргумента должен соответствовать вызываемой функции. Нельзя, например, указать при вызове тригонометрической функции аргумент символьного типа.

В результате исполнения функции получается некоторый результат, который также имеет вполне определенный тип, что следует помнить, используя функции в выражениях. Наиболее употребимые функции приведены в таблице 15.

Аргументы всех перечисленных функций должны относиться к арифметическому типу. Аргумент датчика случайных чисел на результат влияния не оказывает и может быть любым числом. В случае тригонометрических функций аргумент указывают в радианах. Для

перевода градусов в радианы используют формулу:

$$\text{радианы} = \text{градусы} \times \pi/180.$$

Для перевода натуральных логарифмов в десятичную следует использовать формулу:

$$\lg x = \ln x / \ln 10.$$

Функции для работы со строками рассмотрены в соответствующем разделе.

Ввод данных с клавиатуры и вывод информации на экран

Ввод данных. Для ввода данных с клавиатуры в Basic'е используется оператор INPUT. Общая форма записи оператора:

нс INPUT "<текст подсказки>"; <список переменных>.

Здесь:

INPUT — ключевое слово — имя оператора;

"<текст подсказки>" — произвольный текст, поясняющий, какие данные надо ввести. Текст заключают в кавычки. Текст подсказки не обязателен, но пренебрегать им не следует: уменьшается количество ошибок при вводе.

<список переменных> — список имен переменных, значения которых вводятся. Это могут быть простые или индексированные переменные любого типа. Имена в *<списке>* разделяют запятыми.

Оператор используется для присваивания переменным значений.

Принцип действия оператора INPUT:

1. При достижении оператора исполнение программы прерывается; на экране отображаются подсказка, если она предусмотрена, и знак вопроса.

2. Для продолжения работы программы следует ввести с клавиатуры столько значений, сколько имен переменных указано в списке переменных, разделяя значения запятыми. Типы вводимых значений должны соответствовать типам переменных в списке.

3. Собственно ввод завершается нажатием клавиши «ENTER». При этом переменным, указанным в списке, присваиваются набранные значения (старые значения, если они были, теряются).

4. Продолжается исполнение программы со следующего за INPUT оператора.

Примеры записи оператора:

```
30 INPUT "Аргумент функции = "; X
40 INPUT "Число рабочих = "; KOL%
50 INPUT "ФИО студента: "; FIO$
```

Оператор вывода. Для простейшего* вывода данных на экран в Basic'е используется оператор PRINT. Общая форма записи оператора:

нс PRINT <список вывода> <завершающий символ>

или

нс ? <список вывода> <завершающий символ>.

Здесь:

PRINT (или *?*) — ключевое слово — имя оператора;

<список вывода> — список элементов, значения которых выводятся на экран. Элементами *<списка вывода>* могут быть любые выражения в любом наборе (напомним, что константа и переменная — частные случаи выражения). Элементы в списке разделяются *разделителем* — запятой либо точкой с запятой (см. далее). Возможен оператор вывода без списка.

*Форматированный вывод производится при помощи оператора PRINT USING.

<завершающий символ> — это может быть запятая или точка с запятой; завершающего символа может не быть.

Принцип работы оператора PRINT:

1. При достижении оператора PRINT в процессе исполнения программы на экран выводятся значения переменных и констант, указанных в *<списке вывода>* в форме, определяемой разделителем.

2. Значения переменных и констант сохраняются.

3. Если *<завершающий символ>* — пробел, то после исполнения оператора курсор экрана переводится на следующую строку экрана, если запятая или точка с запятой, то курсор остается в строке вывода и следующий вывод (исполнение очередного оператора PRINT) будет продолжен в этой же строке.

Разделитель элементов в *<списке вывода>* (запятая или точка с запятой) определяет формат вывода. При использовании запятой очередной элемент выводится в начало очередной 14–16-символьной зоны текущей строки экрана (зонный формат); это позволяет выводить данные ровными столбцами. Если разделитель — точка с запятой, то очередной элемент выводится через 2 пробела после предыдущего элемента (уплотненный формат).

Примечания:

1. Значения символьных переменных и констант выводятся без кавычек и без апострофов, так как кавычки и апострофы — это элементы записи программы, которые не имеют отношения к значению переменных. От арифметических значений они отделяются пробелом. Два символьных объекта выводятся без пробелов.

2. При выводе арифметических элементов знак «+» не выводится, вместо него выводится пробел.

Примеры записи оператора вывода и результаты его исполнения (предполагается, что выводимые числа

определены ранее):

```
.....................
30 PRINT "Зонный формат"
40 PRINT X1, X2, X3, X4 : PRINT X5, X6, X7, X8
60 PRINT 'Пропуск строки
70 PRINT "Уплотненный формат"
80 PRINT X1; X2; X3; X4 : PRINT X5; X6; X7; X8
```

На экране:

```
Зонный формат
12            6             8             72
7.9           .7888         4.92          4.5
Уплотненный формат
12  6  8  72
7.9  .7888  4.92  4.5
```

При выводе информации на экран полезно бывает его предварительно почистить. Для этого применяется оператор CLS. Форма записи:

нс CLS.

Обычно оператор CLS располагают в одной из первых строк программы. Далее — по мере необходимости.

Оператор присваивания

Редкая программа — как на Basic'e, так и на любом другом языке программирования — обходится без оператора присваивания. Он предназначен для определения или переопределения значения переменной — присваивания ей нового значения. Общая форма записи оператора:

нс <имя переменной> = <выражение>.

Здесь:

<имя переменной> — имя любой переменной, в том числе и индексной;

<выражение> — любое выражение, тип которого соответствует типу переменной (в частности, константа или имя другой переменной);

= — собственно оператор присваивания.

Принцип работы оператора присваивания:

1. Вычисляется значение выражения;

2. При работе с арифметическими типами результат вычисления, если необходимо, преобразуется к типу переменной;

3. Полученный результат присваивается переменной, а прежнее ее значение, если таковое было, теряется.

Примеры использования оператора присваивания:

```
.....................
20 K% = 0
30 X = 5.4
40 T = X * COS(1 - ABS(X))
50 F$ = "Сокопов Ю. А."
60 ARR(4) = X
70 K% = K% + 1
```

В операторе присваивания слева и справа от символа «=» может фигурировать имя одной и той же переменной, например: K%=K%+1 (увеличили прежнее, текущее значение переменной K% на единицу), но это допускается лишь в том случае, если значение переменной определено ранее (строка 20 в примере).

Нельзя путать оператор присваивания с операцией сравнения «равно», хотя они и обозначаются одним и тем же символом!

Задача 3.2. Пример реализации линейного алгоритма.

Дан прямоугольный треугольник: с клавиатуры вводится длина одного из катетов A и величина прилежащего к нему угла (в градусах) BU. Составить программу вычисления периметра треугольника.

Решение.

Проанализируем задачу. Она может быть разбита на три основных этапа: ввод исходных данных, этап вычислений и вывод результата. Первый и последний

этапы проблемы не составляют. Остановимся на этапе вычислений.

Периметр — это сумма длин сторон. Одна из них известна — катет, вводимый с клавиатуры. Зная катет и прилежащий к нему угол. гипотенузу и другой катет рассчитаем по формулам:

$$C=A/COS(BU) \text{ и } B=A*TAN(BU).$$

Тогда периметр P=A+B+C. Все просто, но важно не упустить из виду, что угол вводится в градусах, а тригонометрические функции в Basic'е требуют аргумента в радианах, следовательно, потребуется предварительный перевод: BU=BU*3.14/180.

Как лучше организовать вычисления? Можно соорудить одно длинное и сложное выражение, но лучше не стоит: чем сложнее выражение, тем выше вероятность ошибки. Можно для каждой стороны и периметра ввести свои переменные и рассчитать их по вышеприведенным формулам. Это не очень рационально: вспомогательные объекты загромождают программу. Для решения задачи достаточно ввести всего единственную переменную, если воспользоваться тем, что в операторе присваивания одна и та же переменная может присутствовать как справа, так и слева от знака оператора. Этим приемом мы уже воспользовались при переводе градусов в радианы.

Несколько слов о типах данных: все объекты задачи (и, следовательно, алгоритма) — вещественные числа. Учтем это при выборе идентификаторов.

Программа будет иметь следующий вид:

```
10 'Вычисление периметра треугольника
20 CLS                              'Очистка экрана
30 INPUT "Катет ="; A               'Ввод длины катета
40 INPUT "Прилежащий угол ="; BU 'Ввод величины угла
50 BU = BU * 3.14 / 180  'Перевод в радианы
60 P = A / COS(BU)       'Вычисление гипотенузы
70 P = P + A * TAN(BU)   'Суммируем гипотенузу и 2-й катет
80 P = P + A             'Прибавляем к общей сумме 1-й катет
```

```
90 PRINT "Периметр ="; P 'Вывод результата
100 END
```

Разветвляющиеся алгоритмы и операторы ветвления

Ход решения подавляющего большинства задач может быть неоднозначен. На каком-то из этапов решения мы оказываемся перед необходимостью пойти по тому или иному пути, в зависимости от выполнения или невыполнения некоторого условия. Простейший пример: анализ оценки, полученной при сдаче вступительных экзаменов. Если суммарный балл не ниже проходного, абитуриент заносится в списки студентов, в противном случае ему возвращают документы. Подобного рода алгоритмы называются разветвляющимися.

Разветвляющимися называются алгоритмы, в которых последовательность выполнения операторов определяется некоторыми условиями.

Для реализации разветвлений в Basic'е имеется два оператора: условный оператор IF-THEN-ELSE и оператор множественного выбора ON-GOTO. Порой бывает полезен оператор безусловного перехода GOTO.

Условный оператор IF-TNEN-ELSE. Условный оператор применяется для выбора из двух альтернатив. Он может быть представлен в следующем виде:

нс IF<выражение> THEN<оп-1> [ELSE<оп-2>].

Здесь:

IF, THEN, ELSE — ключевые слова;

<выражение> — выражение арифметического типа: как правило, оно содержит некоторое условие;

<оп-1> — один или несколько операторов, разделенных двоеточиями (*<оп-2>* — аналогично).

Фрагмент оператора, заключенный в квадратные скобки (ELSE-ветвь), может отсутствовать: имеет место условный оператор в сокращенной форме.

Вся IF-конструкция должна располагаться в одной строке программы*.

Принцип действия оператора:

1. Вычисляется <*выражение*> при текущих значениях входящих в него переменных.

2. Если полученное значение есть истина (отлично от нуля), то выполняются <*оп-1*> (THEN-ветвь); <*оп-2*> игнорируются. Если полученное значение ложно (равно нулю), то выполняются <*оп-2*> (ELSE-ветвь); <*оп-1*> игнорируются.

3. После выполнения любой из ветвей управление передается следующей строке программы. Если ELSE-ветвь отсутствует и условие ложно, то управление сразу передается следующей строке программы: никакие из операторов не выполняются.

Примеры записи условного оператора:

— для нахождения большего из двух значений:

```
40 IF A > B THEN X = A ELSE X = B
```

— для нахождения большего из двух значений с фиксацией его имени:

```
50 IF A > B THEN X = A : F$ = "A" ELSE X = B : F$ = "B"
```

Один условный оператор в полной форме может быть заменен двумя в сокращенной. Произведем эквивалентную замену оператора из строки 50:

```
51 IF A > B THEN X = A : F$ = "A"
52 IF A <= B THEN X = B : F$ = "B"
```

Среди операторов, следующих за ключевыми словами THEN и (или) ELSE, может располагаться другой

*В более развитых версиях это условие не обязательно: признаком окончания области действия оператора является ключевое слово END IF.

условный оператор. Подобные конструкции называются вложенными условными операторами.

Например:

```
60 IF A > B THEN X = A - B: IF X > C THEN X = X - C
70 IF F$ = "Петров" THEN IF G% = 18 THEN PRINT "Привет!"
80 IF X > 0 THEN IF X <= 1 THEN Y = 1 + X ELSE Y = X * 2
```

К какому из IF относится ELSE в строке 80? Для исключения неоднозначности принимается, что ELSE принадлежит последнему «неукомплектованному» IF.

Вложенные условные операторы затрудняют чтение программы, и от них следует избавиться. Сделать это можно всегда, используя логические операции. Произведем эквивалентную замену операторов из строк 60, 70 и 80:

```
61 IF A > B THEN X = A - B
62 IF A > B AND X > C THEN X = X - C
71 IF F$ = "Петров" AND G% = 18 THEN PRINT "Привет!"
81 IF X > 0 AND X <= 1 THEN Y = 1 + X
82 IF X > 1 THEN Y = X * 2
```

Задача 3.3. Пример использования условного оператора.

С клавиатуры вводятся координаты точки на плоскости (X и Y). Указать, в какой из четвертей находится точка (1: X>=0 и Y>=0; 2: X>=0 и Y<0; 3: X<0 и Y<0; 4: X<0 и Y>=0).

Решение.

Задача состоит из трех этапов: ввод исходных данных (двух координат точки, вещественных чисел), анализа введенных данных и выдачи результата. На экране мы должны увидеть целое число от 1 до 4 и пояснительный текст. Число мы получаем в результате анализа координат; это — третий объект программы, который нам потребуется: переменная целого типа (назовем ее PART).

Анализ осуществим при помощи условного оператора. Возможно несколько способов записи с разной

степенью вложенности. Приведем два полярных варианта. Первый:

```
10 'Определение местонахождения точки
20 CLS
30 INPUT "X ="; X
40 INPUT "Y ="; Y
50 IF X >= 0 AND Y >= 0 THEN PART = 1
60 IF X >= 0 AND Y < 0 THEN PART = 2
70 IF X < 0 AND Y < 0 THEN PART = 3
80 IF X < 0 AND Y >= 0 THEN PART = 4
90 PRINT "Точка расположена в"; PART; "четверти"
100 END
```

Второй вариант получим, заменив строки 50–80 одной строкой (длина программной строки в Basic'е, в отличие от книжной, может содержать до 255 символов) следующего вида:

```
IF X >= 0 THEN IF Y >= 0 THEN PART = 1 ELSE PART = 2
          ELSE IF Y >= 0 THEN PART = 4 ELSE PART = 3
```

Напомним, что вся IF-конструкция должна находиться на одной строке.

Оператор безусловного перехода GOTO. Оператор перехода используется для изменения естественной последовательности строк программы. Оператор имеет следующую форму записи:

нс GOTO <номер строки>.

Здесь:

GOTO — ключевое слово, название оператора;

<номер строки> — указывает строку, которой требуется передать управление.

Принцип действия оператора: он передает управление в указанную строку программы. Далее выполняются строки, следующие за указанной.

GOTO — оператор, использовать который надо только при крайней необходимости: он портит структуру программы, делает ее запутанной и затрудняет отладку. Говорят, что квалификация программиста

обратно пропорциональна количеству операторов перехода в его программах. Наличие циклических операторов в языке (см. ниже) позволяет обходиться без этого оператора вообще. Между тем применение его в рассматриваемой версии Basic'а иногда бывает удобно при реализации разветвляющихся алгоритмов (что вызвано требованием уместить всю IF-конструкцию на одной строке). В частности, использование GOTO оправдано в случае проверки данных при вводе их с клавиатуры.

Задача 3.4. Пример использования оператора перехода.

С клавиатуры вводится положительное двузначное число *N*. Определить, кратно ли оно 12. Предусмотреть проверку вводимых значений.

Решение.

Цель задачи — продемонстрировать, как осуществляется контроль вводимой информации. Каким условиям должно удовлетворять вводимое число? Во-первых, раз оно положительное и двузначное, оно должно находиться в диапазоне между 10 и 99. Во-вторых, мы должны проверить его на кратность, а это означает, что вводимое число не должно иметь дробной части. С помощью условного оператора будем проверять, отвечает ли число данным требованиям (строка 40). Если хотя бы одно не выполняется, отправляем пользователя вновь на этап ввода (в строку 30): пока не введет правильно.

```
10 'Проверка кратности
20 CLS
30 INPUT "N ="; N
40 IF N < 10 OR N > 99 OR N <> FIX(N) THEN GOTO 30
50 IF N MOD 12 = 0 THEN PRINT N; "кратно 12"
60 IF N MOD 12 <> 0 THEN PRINT N; "не кратно 12"
70 END
```

Примечания.

1. Обратите внимание, как используются в данном примере функция FIX и операция получения остатка от деления MOD.

2. При использовании оператора перехода непосредственно после ключевых слов THEN и ELSE ключевое слово GOTO может быть опущено. В этом случае строка 40 будет иметь вид:

```
40 IF N < 10 OR N > 99 OR N <> FIX(N) THEN 30
```

3. Операторы строк 50 и 60 можно объединить в один.

Оператор множественного выбора ON-GOTO. Если требуется осуществить выбор более чем из двух альтернатив, применяется оператор множественного выбора — ON-GOTO*. Он имеет следующую форму записи:

нс ON <ар. выраж> GOTO <список номеров строк>.

Здесь:

ON, GOTO — ключевые слова;

<ар. выраж> — выражение, вычисляемое при текущих значениях входящих в него переменных;

<список номеров строк> — перечень номеров строк (через запятую), на одну из которых должно быть передано управление.

Принцип действия оператора:

1. Вычисляется значение выражения.

2. Полученное значение округляется до целого числа.

3. Управление передается строке, очередность которой в списке номеров строк (а не в программе!) соответствует вычисленному значению. Если значение

*Более развитые версии Basic'a оснащены также оператором множественного выбора SELECT-CASE-END SELECT. Принцип его работы иной.

выражения меньше 1 или превышает количество элементов в списке номеров строк, управление передается оператору, следующему непосредственно за оператором ON-GOTO.

Например:

```
50 ON I% * X GOTO 180, 120, 20, 90
```

В списке номеров четыре элемента. Если результат выражения (I%*X) равен 1, управление будет передано строке 180. Если равен 2 — строке 120, 3 — 20, 4 — 90. Если значение выражения больше 4 или меньше 1 — строке 60.

Операторы множественного выбора широко используются при организации меню.

Задача 3.5. Организация меню с помощью оператора выбора.

Компьютер должен «запросить» два числа (*M* и *N*), которые вводятся с клавиатуры. Произвести над этими числами одно из арифметических действий (сложение, вычитание, умножение, деление) — по выбору. Вывести результат на экран с пояснительным текстом.

Решение.

M и N — числа, с которыми предстоит произвести одно из четырех действий. Имеем четыре альтернативы, следовательно, нужен оператор выбора. Еще нам потребуется некоторый критерий, на основании которого будет осуществляться выбор действия. Самое простое — «закрепить» за каждой альтернативой свое число от 1 до 4 и, отобразив на экране собственно меню, предложить пользователю ввести значение. Потребуется переменная, в которую будет заноситься число, вводимое пользователем. Назовем ее I%. При описании альтернативы «деление» не забудем, что знаменатель не должен быть равен нулю.

```
10 'Простейшее меню - выбор из предлагаемых вариантов
20 CLS
30 INPUT "M ="; M : INPUT "N ="; N
40 PRINT "Что сделать с этими числами? 1 +; 2 -; 3 *; 4 /."
50 INPUT I%
60 ON I% GOTO 70, 90, 110, 130
70 PRINT M; "+"; N; "="; M + N
80 GOTO 150
90 PRINT M; "-"; N; "="; M - N
100 GOTO 150
110 PRINT M; "*"; N; "="; M * N
120 GOTO 150
130 IF N <> 0 THEN PRINT M; "/"; N; "="; M / N
140 IF N = 0 THEN PRINT "Делить на ноль нельзя."
150 END
```

Для того чтобы программа после реализации оператора выбора вновь «слилась в единое русло» (чтобы работала только одна из альтернатив), нужно указать общую точку завершения каждого из вариантов. В нашем случае это строка 150 — конец программы.

Циклические операторы

Реализация циклических алгоритмов. Большинство практических задач требует многократного повторения одних и тех же действий, т. е. повторного использования одного или нескольких операторов.

Пусть требуется ввести и обработать последовательность чисел. Если чисел всего пять, можно составить линейный алгоритм. Если их тысяча, записать линейный алгоритм можно, но очень утомительно и нерационально. Если количество чисел к моменту разработки алгоритма неизвестно, то линейный алгоритм принципиально невозможен. Преодолеть подобные трудности можно с помощью циклов.

Циклом называется многократно исполняемый участок алгоритма (программы). Соответственно циклический алгоритм — это алгоритм, содержащий циклы. Итерация — это повторение тела цикла, виток.

Различают два типа циклов: с известным числом повторений и с неизвестным числом повторений (итерационные циклы). При этом в обоих случаях имеется в виду число повторений на стадии разработки алгоритма. Для реализации циклов в Basic'е имеются специальные операторы.

Иногда для организации цикла используют условный оператор в сочетании с оператором безусловного перехода. Лучше этого не делать.

Цикл с известным числом повторений FOR. Цикл с известным числом повторений называют также циклом с параметром. В Basic'е он имеет следующую форму записи:

```
нс FOR <переменная> = <выр-1> TO <выр-2> [STEP <выр-3>]
   <тело цикла>
нс NEXT <переменная>.
```

Здесь:

FOR, TO, STEP, NEXT — ключевые слова;

<переменная> — управляющая переменная (параметр) цикла. Это может быть любая переменная арифметического типа. После ключевых слов *FOR* и *NEXT* должна использоваться одна и та же переменная;

<выр-1>, *<выр-2>*, *<выр-3>* — арифметические выражения, определяющие соответственно начальное значение параметра, конечное его значение и приращение (шаг). Шаг может быть как положительным, так и отрицательным; если он равен +1, заключенную в квадратные скобки часть оператора можно опустить. Эти выражения вычисляются только на входе в цикл: переменные, входящие в выражения, могут в теле цикла измениться, но на ход цикла это не влияет.

<тело цикла> — любое количество пронумерованных строк программы, содержащих операторы, которые требуется повторять. Ключевое слово *NEXT* — признак окончания тела цикла.

Первая строка — заголовок цикла — определяет условие повторения цикла.

Принцип действия оператора:

1. Вычисляются выражения <*выр-1*>, <*выр-2*>, <*выр-3*>.

2. Параметру присваивается начальное значение <*выр-1*>.

3. Значение параметра сравнивается с конечным значением <*выр-2*>:

— если значение параметра не больше <*выр-2*> (при положительном шаге) или не меньше <*выр-2*> (при отрицательном шаге), то выполняется тело цикла;

— в противном случае осуществляется выход из цикла, и выполняется оператор, следующий за ключевым словом *NEXT*.

4. Если не произошел выход из цикла, значение параметра изменяется на величину шага, и повторяется п. 3.

Иногда по условию задачи требуется досрочный выход из цикла. Сделать это можно двумя способами:

— в теле цикла присвоить параметру цикла значение больше <*выр-2*> (при положительном шаге) или меньше <*выр-2*> (при отрицательном шаге);

— с помощью оператора перехода передать управление строке, следующей за той, что содержит ключевое слово *NEXT*.

Ниже приведен пример использования оператора цикла (см. задачу 3.6).

Часто приходится решать задачи, требующие использования вложенных операторов цикла (см. задачу 3.7). В этом случае в теле одного из циклов (внешнего, охватывающего) находится другой оператор цикла (внутренний). Внешний и внутренний циклы должны иметь параметры с разными именами, а NEXT внутреннего цикла предшествовать NEXT'у внешнего.

Задача 3.6. Пример использования цикла FOR.

В простую переменную с клавиатуры последовательно ввести *N* чисел и посчитать их сумму.

Решение.

Проанализируем условие. Нам потребуются: переменная целого типа, в которой будет храниться количество вводимых чисел (`N%`), вещественная переменная, куда будем вводить по очереди с клавиатуры числа (`X`), вещественная переменная для фиксации суммы (`S`) и переменная-параметр цикла (`I`).

Предстоят повторяющиеся действия: каждое число нужно ввести и присовокупить к общей сумме (тело цикла составят два оператора). Значит нужен цикл. Количество чисел не оговорено: предложим пользователю ввести его с клавиатуры. При этом к моменту запуска цикла оно будет известно, следовательно, используем цикл с известным числом повторений FOR.

Важный момент: прибавлять к текущей сумме очередное число будем по формуле:

```
S=S+X.
```

А к чему мы прибавим первое число? Использовать неопределенные переменные нельзя, значит нужно определить исходное значение суммы — обнулить его (действительно, до ввода первого числа сумма равна нулю).

```
10 'Определение суммы чисел последовательности
20 CLS
30 INPUT "Сколько чисел в последовательности"; N%
40 S = 0
50 FOR I = 1 TO N% STEP 1
60 INPUT "Очередное число ="; X
70 S = S + X
80 NEXT I
90 PRINT "Сумма равна"; S
100 END
```

Задача 3.7. Пример использования вложенных циклов FOR.

Имеется экзаменационная ведомость по 4 группам студентов. В каждой по 25 человек. Вывести средний балл для каждой группы и подсчитать общий средний балл.

Решение.

Среднее арифметическое вычисляется путем деления суммарного балла на количество студентов. Количества нам известны, их считать не надо. Способ вычисления суммы для одной группы не отличается от рассмотренного в предыдущей задаче. Оценки студентов каждой группы — та же числовая последовательность, и последовательностей таких в настоящей задаче четыре (4 группы).

Нам потребуются две переменные вещественного типа (среднее арифметическое не следует делать целым, даже если анализируемые числа — целые): одна — для общего среднего, другая — для средних значений по группам. Посчитав для очередной группы среднее, будем выводить его на экран и использовать данную переменную повторно — уже для другой группы. Не забудем обнулить ее перед каждой новой группой.

Ввод данных будет производиться по группам (сначала оценки всех студентов 1-й группы, потом — 2-й и т.д.), т.е. внешний цикл — по числу групп, внутренний — по числу студентов группы. Тело внутреннего цикла — строки 80–90, внешнего — строки 50–120.

```
10 'Определение среднего балла
20 CLS
30 S = 0      'Общий средний балл
40 FOR I = 1 TO 4 STEP 1
50 PRINT "Группа №"; I
60 SGR = 0    'Средний балл в текущей группе
70 FOR J = 1 TO 25 STEP 1
80 INPUT "Балл очередного студента ="; BALL
90 SGR = SGR + BALL
100 NEXT J
```

```
110 S = S + SGR
120 PRINT "Средний балл в группе №"; I; "равен"; SGR/25
130 NEXT I
140 PRINT "Общий средний балл равен"; S/4
150 END
```

Циклы с неизвестным числом повторений. Цикл с неизвестным числом повторений (итерационный цикл) можно считать универсальным, т. е. его можно использовать в любых случаях, когда необходим цикл, в том числе и если количество повторов известно.

Итерационные циклы делятся на циклы с предусловием (WHILE-WEND) и с постусловием (DO-LOOP-WHILE и другие — в зависимости от версии). В первых сначала осуществляется проверка условия цикла, а потом, если оно истинно, выполняется тело цикла. В циклах с постусловием выполнение тела цикла предваряет проверку условия, поэтому он непременно выполняется хотя бы один раз. Это делает его источником труднообнаружимых ошибок при невнимательном использовании; кроме того, его использование действительно оправдано в достаточно редких случаях, поэтому изучать его работу лучше на практике. Поскольку цикл с постусловием всегда можно заменить циклом с предусловием, сосредоточимся на последнем.

В Basic'е цикл WHILE-WEND он представляется следующей конструкцией:

нс WHILE <выражение>
<тело цикла>
нс WEND.

Здесь:

WHILE, *WEND* — ключевые слова;

<выражение> — определяет условие выполнения цикла;

<тело цикла> — любое количество последовательно пронумерованных строк программы, содержащих операторы, которые требуется повторять. Ключевое слово *WEND* — признак конца тела цикла.

Принцип действия оператора:

1. Вычисляется *<выражение>*.

2а. Если полученное значение истинно, то выполняется *<тело цикла>* и вновь выполняется п. 1 — вычисляется *<выражение>*.

2б. Если полученное значение ложно, то *<тело цикла>* не выполняется и осуществляется выход из цикла: управление передается оператору, который следует за оператором WEND.

Из сказанного выше следует, что тело цикла должно содержать операторы, которые изменяют значения *<выражения>*. В противном случае будет иметь место бесконечный цикл, и программа «зависнет».

Итерационные циклы допускают вложение других циклов без каких-либо ограничений.

Цикл FOR вида

```
50 FOR I = A TO B STEP ST
60 <тело цикла>
70 NEXT I
```

можно заменить итерационным циклом следующим способом:

```
40 I = A
50 WHILE I < B  'При положительном шаге
60 <тело цикла>
70 I = I + ST
80 WEND
```

Шаг и условие окончания цикла могут изменяться в теле цикла. В отличие от цикла FOR, это окажет влияние на ход цикла.

Задача 3.8. Пример использования цикла WHILE–WEND.

В простую переменную с клавиатуры последовательно вводятся положительные числа. Условие окончания ввода — ввод символа 0. Вычислить произведение этих чисел.

Решение.

Проанализируем условие. Предстоит несколько раз повторять один и тот же набор операторов (ввод очередного числа и его обработка), следовательно, проектируем циклический алгоритм. Сколько будет введено чисел, предугадать невозможно, поэтому потребуется итерационный цикл.

Объекты программы: вещественная переменная (X), куда будем вводить по очереди с клавиатуры числа, вещественная переменная для хранения произведения (P). Для произведения, как и для суммы, нужно определить стартовое значение, но это будет уже не 0, а 1, иначе, каковы бы ни были вводимые числа, произведение так и останется равно нулю.

Важно, чтобы общее произведение не было умножено на ноль, который является сигналом к окончанию цикла. Во избежание этого проверка введенного числа на неравенство нулю должна непосредственно предшествовать умножению. Поэтому первое число введем отдельно, до входа в цикл.

```
10 'Определение произведения чисел последовательности
20 INPUT "Первое число =";  X
30 P = 1
40 WHILE X <> 0
50 P = P * X
60 INPUT "Очередное число =";  X
70 WEND
80 PRINT "Произведение равно";  P
90 END
```

Задачи на вычисление суммы, произведения, количества, среднего арифметического, максимума, минимума в числовой последовательности. Задачи данного типа решаются по общей схеме с использованием циклов. До входа в цикл производится подготовка — задание исходных значений суммы, максимума и т. д. Тип цикла определяется тем, известно будет число повторений или нет (соответственно FOR либо WHILE). В теле цикла производятся ввод очередного числа и его обработка: последовательность операторов зависит от условия задачи и типа цикла. После выхода из цикла производится вывод результата.

Простейший случай — вычисление суммы, произведения, количества, среднего арифметического и т. п. для всех членов последовательности. Подобные задачи рассмотрены выше (см. раздел «Циклические операторы»).

Задача 3.9. Вычисление среднего арифметического по условию.

В переменную последовательно вводят числа. Окончание ввода — по желанию пользователя либо когда сумма отрицательных чисел достигнет –1000. Определить среднее арифметическое отрицательных чисел.

Решение.

Как и в задаче 3.8, используем итерационный цикл. Но теперь окончание ввода определяется не положением нуля в последовательности, а одним из двух условий. Оба условия введем в выражение в заголовке цикла, объединив логической операцией AND: достаточно одному из них стать ложным и произойдет выход из цикла. При этом потребуется специальная переменная для контроля продолжения ввода, в которую пользователь будет вводить числа 1

либо 0. Для того чтобы войти в цикл, инициализируем эту переменную: присвоим ей отличное от 0 значение. Зададим также исходные значения для суммы и количества отрицательных чисел — именно такое среднее арифметическое указано в условии. В этой задаче нужно считать и количество.

В теле цикла вводим очередное число и обрабатываем его: если число отрицательное, прибавляем его к сумме и на 1 наращиваем количество. В противном случае игнорируем введенное число. Далее спрашиваем пользователя, хочет ли он продолжать ввод, и фиксируем введенное значение.

При выводе результата необходимо помнить, что оговоренных в условии чисел в последовательности может не оказаться. Нужно предусмотреть вывод соответствующего сообщения в этом случае.

```
10 S = 0 : KOL = 0 : F = 1
20 WHILE F <> 0 AND S > -1000
30 INPUT "Очередное число ="; X
40 IF X < 0 THEN S = S + X : KOL = KOL + 1
50 INPUT "Продолжаем ввод (1 - да, 0 - нет)"; F
60 WEND
70 IF KOL > 0 THEN PRINT "Среднее отр. чисел равно"; S / KOL
80 IF KOL = 0 THEN PRINT "Отрицательных чисел нет."
90 END
```

Задача 3.10. Определение максимального значения.

В простую переменную последовательно вводятся *N* вещественных чисел. Вычислить максимальное значение.

Решение.

Как и в задаче 3.6, используем цикл FOR. Потребуется переменная MAX, чтобы заносить число, которое будет наибольшим среди всех уже введенных чисел. Если очередное введенное число окажется больше, чем текущее значение MAX, значение MAX переопределяется, т. е. мы в цикле сравниваем введенное число с переменной MAX.

Каково должно быть исходное значение максимума? Возможны два варианта. В первом случае до входа в цикл вводим 1-е число последовательности и присваиваем это значение переменной MAX; цикл тогда начинаем со 2-го числа. Во втором случае в цикле вводим и анализируем все числа, а переменной MAX присваиваем такое значение, которое наверняка окажется меньше или равно первому же введенному числу. Очевидно, что это может быть лишь нижняя граница диапазона вещественных чисел (при определении минимума — берется верхняя граница диапазона).

```
10 'Определение максимального значения - вариант 1
20 INPUT "Сколько чисел в последовательности"; N%
30 INPUT "Первое число"; MAX
40 FOR I = 2 TO N%
50 INPUT "Очередное число ="; X
60 IF X >= MAX THEN MAX = X
70 NEXT I
80 PRINT "MAX ="; MAX
90 END
```

```
10 'Определение максимального значения - вариант 2
20 INPUT "Сколько чисел в последовательности"; N%
30 MAX = - E + 38
40 FOR I = 1 TO N%
50 INPUT "Очередное число ="; X
60 IF X >= MAX THEN MAX = X
70 NEXT I
80 PRINT "MAX ="; MAX
90 END
```

Задача 3.11. Определение количества чисел, равных максимуму.

В простую переменную последовательно вводятся 10 чисел. Определить максимальное значение и количество чисел, равных максимуму.

Решение.

См. задачу 3.10. В этой задаче появляется новый объект — количество максимумов (переменная KOL). Сколько максимумов имеем, пока не вошли в цикл? 0. Значит, KOL надо обнулить. Что мы делаем, введя

число? 1. Проверяем, не равно ли оно текущему максимуму. Если да, увеличиваем значение количества на единицу. 2. Проверяем, не больше ли введенное число текущего максимума. Если нет, переходим к следующей итерации цикла. А если да, переопределяем текущий максимум и сбрасываем счетчик: KOL=1. Это — вариант 1. Если же поменять эти две проверки местами (вариант 2), счетчик нужно будет сбрасывать до 0, чтобы дважды не считать один и тот же максимум.

В любом случае про подобные алгоритмы говорят: лидер сменился — сбрасываем счетчик.

```
10 'Определение количества максимумов - вариант 1
20 KOL = 0 : MAX = -E+38
30 FOR I = 1 TO 10
40 INPUT "Очередное число ="; X
50 IF X = MAX THEN KOL = KOL + 1
60 IF X > MAX THEN MAX = X : KOL = 1
70 NEXT I
80 PRINT "MAX="; MAX; "Таких чисел"; KOL
90 END
```

```
10 'Определение количества максимумов - вариант 2
..........
50 IF X > MAX THEN MAX = X : KOL = 0
60 IF X = MAX THEN KOL = KOL + 1
..........
```

Задача 3.12. Первый и последний минимумы (максимумы).

В простую переменную последовательно вводятся N положительных чисел. Определить: 1. Минимальное значение и порядковый номер последнего числа, равного этому значению. 2. Максимальное значение и порядковый номер первого числа, равного этому значению.

Решение.

См. задачу 3.10. Определение минимального значения принципиально не отличается от поиска максимального. В задаче 3.10 мы находили (фиксировали)

последний максимум: переопределение происходило, если очередное введенное число оказывалось больше или равно текущему значению переменной MAX. Случай равенства тоже приводил к переопределению. Для нахождения первого максимума нужно исключить переопределение максимального значения в случае равенства, т. е. в условии будет не >=, а >. С минимумами — аналогично.

Для фиксации порядковых номеров нужно завести отдельные переменные, `FirstMaxPos` и `LastMinPos`. Естественно, позиция будет переопределяться вместе с максимумом (минимумом). Именно порядковый номер указывает, какой из максимумов (минимумов) найден.

Замечание: при определении первых максимумов использование варианта 2 (задача 3.10) приводит к некорректному результату — в случае, если максимум равен `-E+38`. Значение собственно максимума будет верным, но позиция найдена не будет. С первым минимумом — аналогично.

```
10 'Определение позиций максимумов и минимумов
20 INPUT "Сколько чисел в последовательности"; N%
30 INPUT "Первое число"; X
40 MAX = X : FirstMaxPos = 1
50 MIN = X : LastMinPos = 1
60 FOR I = 2 TO N%
70 INPUT "Очередное число ="; X
80 IF X > MAX THEN MAX = X : FirstMaxPos = I
90 IF X <= MIN THEN MIN = X : LastMinPos = I
100 NEXT I
110 PRINT "MAX ="; MAX; "; первый MAX в позиции"; FirstMaxPos
120 PRINT "MIN ="; MIN; "; последний MIN в позиции"; LastMinPos
130 END
```

Задачи на анализ числовых последовательностей

Задача 3.13. Сравнение чисел последовательности (1).

В простую переменную последовательно вводятся *N* чисел. Сколько чисел больше своих соседей слева?

Решение.

Сколько чисел всего может оказаться больше своих предшественников? Общее количество минус 1: первое число не имеет предшественника, посему из рассмотрения выпадает. Все прочие числа подобны, действия с ними производим одни и те же, значит с ними работаем в цикле. Первое принципиально отлично от иных чисел, поэтому разумно будет ввести его отдельно от прочих, вне цикла, до входа в него. При сравнении мы работаем каждый раз с парой чисел, так что надо и переопределять два числа. Одно — будет новое, свежевведенное. Его записываем в переменную X. Другое — предыдущее, то, которое вводили на предыдущей итерации, — сосед слева. Каждое из введенных чисел после обработки запоминаем в переменной PRED (на той же итерации, где и ввели его). Это позволяет нам использовать переменную X для ввода нового числа на следующей итерации.

```
10 'Сколько чисел больше своих соседей слева?
20 INPUT "Сколько будет чисел?"; N%
30 KOL = 0
40 INPUT "Первое число ="; PRED
50 FOR I = 2 TO N%
60 INPUT "Очередное число ="; X
70 IF X > PRED THEN KOL = KOL + 1
80 PRED = X
90 NEXT
100 PRINT "Больше своих соседей слева"; KOL;
110 PRINT "чисел из"; N% - 1; "возможных."
120 END
```

Задача 3.14. Сравнение чисел последовательности (2).

В простую переменную последовательно вводятся числа. Окончание ввода — 0. Определить, сколько чисел больше своих соседей слева и справа.

Решение.

См. задачи 3.8 и 3.13. Если сравниваем с обоими соседями, выпадет как первое, так и последнее число.

В рассмотрении находятся одновременно уже три числа. И три числа переопределяем на каждой итерации. Вводим числа по одному, а сравниваем каждое с двумя, следовательно, до входа в цикл введем первые два числа.

Тонкость: считать или нет последний ноль членом последовательности? Из условия это однозначно не следует, поэтому примем, что да. Иначе пришлось бы ввести дополнительное условие в строку 70 (`... AND SLED <> 0 ...`).

```
10 'Сколько чисел больше своих соседей справа и слева?
20 KOL = 0
30 INPUT " Введи первые два числа <> 0"; PRED, X
40 IF PRED = 0 OR X = 0 THEN 40 'Проверка вводимых значений
50 WHILE X <> 0
60 INPUT "Очередное число (0 - выход)"; SLED
70 IF X > PRED AND X > SLED THEN KOL = KOL + 1
80 PRED = X
90 X = SLED
100 WEND
110 PRINT "Больше своих соседей"; KOL; "чис."
120 END
```

Задача 3.15. Сравнение чисел с заданным значением.

В простую переменную последовательно вводятся N чисел. Все ли числа меньше заданного числа K?

Решение.

См. задачу 3.6. Помимо количества чисел последовательности, необходимо ввести с клавиатуры значение (`Z`), с которым будем сравнивать каждое вводимое число.

Потребуется также переменная, в которую будем заносить результат анализа последовательности. В тех случаях, когда предполагается ответ «да» или «нет», обычно объявляют переменные-флажки (в нашей задаче `FLAG`), которые принимают одно из двух значений (как правило, 1 и 0). Перед началом анализа (до входа в цикл) им присваивают некоторое значение, связанное с ответом «да» или «нет» — в зависимости от условия.

В ходе анализа переменная может изменить значение либо нет. По окончании цикла значение флажка проверяется и выводится соответствующий ответ.

Как только ответ становится ясен, продолжение анализа делается бессмысленно: можно досрочно завершить цикл (см. строку 70).

```
10 'Все ли числа последовательности больше заданного значения?
20 INPUT "Сколько чисел в последовательности"; N%
30 INPUT "Заданное значение ="; Z
40 FLAG = 1 'Предварительно приняли, что все числа больше
50 FOR I = 1 TO N%
60 INPUT "Очередное число ="; X
70 IF X <= Z THEN FLAG = 0 : I = N% 'Можно не продолжать
80 NEXT I
90 IF FLAG = 1 PRINT "Все"; ELSE "Не все";
100 PRINT " числа последовательности больше Z"
110 END
```

Задача 3.16. Анализ завершающей цепочки в последовательности.

В простую переменную последовательно вводится *N* чисел, отличных от нуля. Положительные или отрицательные числа завершают последовательность и сколько их?

Решение.

Данная задача имеет несколько вариантов решения. Рассмотрим один из них.

Количество чисел в последовательности оговорено, значит используем цикл FOR. В теле цикла будем вводить и обрабатывать каждое число. По условию, числа не должны быть равны нулю, следовательно, необходимо осуществлять контроль вводимых значений (строка 50).

Узнать, положительные или отрицательные числа завершают последовательность, не составляет труда: достаточно, выйдя из цикла, проанализировать знак последнего введенного числа с помощью функции SGN. Определение количества таких чисел проще всего организовать при помощи двух переменных: одна

используется как счетчик положительных чисел в текущей цепочке (KP%), другая — отрицательных (KM%). Когда текущая цепочка — положительная, KM%=0, а когда отрицательная, KP%=0.

```
10 '+ или - в конце последовательности?
20 KP% = 0: KM% = 0 : N% = 20
30 FOR I = 1 TO N%
40 INPUT "Очередное число ="; X
50 IF X = 0 THEN 40
60 IF X > 0 THEN KP% = KP% + 1: KM% = 0
70 IF X < 0 THEN KM% = KM% + 1: KP% = 0
80 NEXT I
90 PRINT "Последовательность завершают ";
100 IF SGN(X) > 0 THEN PRINT "положит. числа в количестве"; KP%
110 IF SGN(X) < 0 THEN PRINT "отрицат. числа в количестве"; KM%
120 END
```

Замечания:

1. Существуют варианты решения, где используется единый счетчик.

2. Аналогично решаются задачи про четные и нечетные числа. Проверка четности: остаток от деления числа на 2 равен нулю в случае четных чисел.

Работа с целыми числами. Важная составляющая любых программ*, а для работы с целыми числами — в особенности, — проверка вводимых значений. Она делается при помощи условного оператора в сочетании с оператором безусловного перехода (в простейших случаях) либо в цикле WHILE. Если используется переменная целого типа (идентификатор оканчивается символом «%»), проверка «на целость» не нужна: при вводе числа с дробной частью последняя будет отброшена. Но, тем не менее, проверка требуется для того, чтобы вводимые числа не выходили за пределы указанного диапазона.

*Проверка вводимых значений не делается в большинстве приводимых в разделе программ из соображений экономии места и ввиду простоты данной процедуры. На практике ее надо делать всегда.

Задача 3.17. Определение суммы цифр числа.

Вводится положительное целое число $A < 10000$. Определить сумму его цифр S и разность $A-S$.

Решение.

Мы не знаем, сколько разрядов в числе, поэтому надо использовать цикл WHILE. Перечислим объекты программы: само число (вводим с клавиатуры с проверкой), сумма цифр (до входа в цикл эту переменную следует обнулить); суммировать будем цифры, причем последовательно. Значит нужна еще переменная для текущей цифры. Пока на этом остановимся и подумаем, как отделить цифру от числа и с какой стороны (младший или старший разряд). Для отделения старшего разряда нужно знать, сколько в числе цифр: мы этого не знаем. Зато операция получения остатка от деления (MOD) на 10 позволит выделить младший разряд. Прибавим его к значению переменной S. Теперь надо добраться до следующего по старшинству разряда. Для этого следует преобразовать число A, убрав цифру, которую уже обработали. Для этого используем операцию целочисленного деления: делим на 10. Три этих действия составят тело цикла (строки 60–80). Каково должно быть условие в заголовке цикла? Очевидно, что при обращении числа A в 0 надо выходить из цикла: все цифры выделены и просуммированы. Сумму выведем после завершения цикла. Но по условию нужно вывести не только сумму, но и разность между исходным значением числа и суммой его цифр. Но значение переменной A в ходе выделения цифр обратилось в ноль. Следовательно, до входа в цикл надо создать «резервную копию» числа A — объявить еще одну переменную и присвоить ей значение числа A (см. строку 40).

```
10 'Сумма цифр числа
20 INPUT "Введи целое число не более 10000"; A
30 IF A <> FIX(A) OR A < 0 OR A > 10000 THEN 20
40 S = 0 : B = A
```

```
50 WHILE A <> 0
60 C = A MOD 10
70 S = S + C
80 A = A \ 10
90 WEND
100 PRINT "Сумма цифр введенного числа равна"; S
110 PRINT "Разность между числом и суммой его цифр равна"; B - S
120 END
```

Замечания:

1. Строго говоря, здесь в переменной C необходимости нет. Строки 60 и 70 можно объединить в одну:

```
S=S+A MOD 10
```

2. Этот алгоритм лежит в основе большой группы задач, где те или иные действия надо выполнять с цифрами числа.

Задача 3.18. Является ли число палиндромом?

Вводится положительное целое число $B < 10000$. Является ли оно палиндромом? Массивы не использовать.

Решение.

Палиндром — это строка или число, которое читается как слева направо, так и справа налево.

Идея решения заключается в том, что, сделав «резервную копию» числа, изменяем порядок следования разрядов копии на обратный. Потом сравниваем инвертированную копию с исходным числом. Если они равны, значит был введен палиндром.

Для реализации идеи потребуется последовательно выделить цифры числа (см. задачу 3.17) и сформировать из них новое число (строка 60). Делать это следует в цикле WHILE, так как неизвестно, сколько разрядов окажется во введенном числе. Помимо копии вводимой переменной понадобится еще одна переменная: в ней будет формироваться перевернутое число (INVERT). Эту переменную следует обнулить.

```
10 'Является ли число палиндромом?
20 INPUT "Введи целое число не более 10000"; B
30 IF B <> FIX(B) OR B < 0 OR B > 10000 THEN 20
40 A = B : INVERT = 0
50 WHILE A <> 0
60 INVERT = INVERT * 10 + A MOD 10
70 A = A \ 10
80 WEND
90 IF INVERT = B THEN PRINT "Палиндром" ELSE PRINT "Не палиндром"
100 END
```

Замечание. Задачу можно решить и с помошью символьных переменных.

Задача 3.19. Поиск простых чисел в заданном диапазоне.

Диапазон положительных чисел задан нижней и верхней границами. Распечатать все простые числа, лежащие в указанном диапазоне.

Решение.

Простые числа делятся нацело только на 1 и на себя.

Границы диапазона следует вводить с клавиатуры с проверкой (строки 20—30). Нужно также позаботиться о том, чтобы нижняя граница диапазона была меньше верхней. Это условие можно ввести в проверку при вводе, но лучше воспользоваться оператором SWAP: он меняет местами значения двух переменных. Применим его, если нижняя граница превысит верхнюю (строка 40).

Выделение и обработку чисел проведем в цикле. Поскольку обе границы известны (введены с клавиатуры), используем цикл FOR. Подлежащее анализу число будет совпадать со значением параметра цикла. Дадим параметру для ясности имя `chislo`.

Но одного цикла тут недостаточно: он позволяет лишь выделить число. Для обработки поместим в тело описанного выше цикла еще один: в нем по оче-

реди будем перебирать числа (делители) от 2 до максимально возможного значения делителя (частного от целочисленного деления анализируемого числа на 2: если взять само число, ошибки не будет, но не имеет смысла). Параметру внутреннего цикла дадим имя `delitel`.

Чтобы сделать однозначное заключение о том, что число простое, необходимо просматривать потенциальные делители до тех пор, пока не найдется такой, на который число разделится нацело. Если он не найдется, число простое: распечатаем его. Если найдется, можно выйти из внутреннего цикла досрочно (но не из внешнего!) и перейти к проверке следующего числа диапазона. Для контроля используем переменную-флажок `flag1` (одно из двух чисел либо простое, либо нет), обнуляя ее для каждого нового числа диапазона, т.е. перед входом во внутренний цикл.

Вывод результата не так прост, как кажется с первого взгляда. Если простые числа встретились, они будут распечатаны, но в указанном диапазоне может не оказаться простых чисел. Нужно предусмотреть и такой вариант. Как правило, в таких случаях тоже используют флажок (`flag2`): в нем фиксируется факт появления первого простого числа (строка 120). Тут речь идет обо всем диапазоне, поэтому обнуляем `flag2` один раз — до входа во внешний цикл. В зависимости от значения этой переменной будет выдаваться (или нет) сообщение об отсутствии простых чисел. А чтобы вывод результата был ясен, добавим поясняющий текст (строка 60). Этот текст появится на экране в любом случае. Далее — в зависимости от наличия простых чисел.

```
10 'Нахождение простых чисел в заданном диапазоне
20 INPUT "Начало диапазона ="; b%: IF b% < 2 THEN 20
30 INPUT "Конец диапазона ="; e%: IF e% < 2 THEN 30
40 IF b% > e% THEN SWAP b%, e%
50 flag2 = 0
60 PRINT "Простые числа в заданном диапазоне ";
```

```
70 FOR chislo = b% TO e%
80 flag1 = 0
90 FOR delitel = 2 TO chislo \ 2
100 IF chislo MOD delitel = 0 THEN flag1 = 1: delitel = chislo
110 NEXT delitel
120 IF flag1 = 0 THEN PRINT chislo; : flag2 = 1
130 NEXT chislo
140 IF flag2 = 0 THEN PRINT "отсутствуют."
150 END
```

Замечание. Более корректно было бы дать переменным chislo, delitel, flag1, flag2 «целочисленные» имена, но в данном случае это не принципиально (в отличие от границ диапазона), поэтому имена упрощены.

Задачи на ряды. Задачи подобного рода можно подразделить на две категории: вычисление конечной суммы ряда (или какого-либо слагаемого) и вычисление суммы ряда (слагаемого, количества слагаемых и т. п.) по заданному условию. Задачи первого типа надо решать при помощи цикла FOR (число повторений известно), второго — с помощью WHILE (число повторений определяется условием). В остальном методология решения задач обоих типов одинакова. Сущность ее заключается в том, что нужно выделить первое слагаемое (иногда — два) и выявить закономерность формирования очередного слагаемого ряда. В формулу, отражающую эту закономерность, может входить предыдущее слагаемое и (или) порядковый номер текущего слагаемого. Часто имеет смысл рассматривать числитель и знаменатель слагаемого отдельно: для каждого из них может быть своя закономерность. Значение очередного слагаемого обычно хранят в некоторой переменной.

Ряды могут иметь «колебательный» характер: четные и нечетные члены формируются по разным закономерностям. Сюда же следует отнести и знакочере-

дующиеся ряды — когда элементы ряда имеют знаки «плюс» и «минус» через один.

Часто обработка текущего члена ряда связана с использованием как собственно текущего, так и предыдущего члена ряда; в этих случаях бывает нужна еще одна переменная (см. задачу 3.13) — для предыдущего слагаемого. Поскольку вычисления производятся в цикле, обе переменные в теле цикла следует переопределять. Например, имеем переменные `prev` и `cur`. При переопределении сперва присваиваем `prev=cur`, затем `cur` присваиваем новое значение, вычисленное на основе выявленной закономерности.

Задача 3.20. Сумма ряда с известным количеством членов.

Для произвольного значения аргумента «x» (по модулю больше 1) вычислить сумму:

$$S = 1 + 1/x + 1/x^2 + 1/x^3 + \cdots + 1/x^k.$$

Решение.

Очевидно, что для вычисления суммы нужен цикл с известным числом повторений (число слагаемых изменяется от 1 до k). Собственно вычисление суммы — это типовой алгоритм, который реализуется оператором присваивания вида:

$$S = S + \langle\text{очередное слагаемое}\rangle.$$

Этот оператор входит в тело цикла. Также в теле цикла нужно организовать вычисление очередного слагаемого (`C`). Слагаемое можно вычислить по его предыдущему значению: `C=C/X`.

Прежде чем записать сам цикл, вспомним правило: любой цикл должен быть подготовлен. В данном необходимо определить число слагаемых (ввод с клавиатуры), значение самого первого члена ряда (оно равно 1) и начальное значение суммы (`S=1`).

```
10 'Сумма ряда с известным количеством членов
20 INPUT "Аргумент X = "; X
30 INPUT "Число слагаемых = "; K% 'K% - целое!
40 S = 1: C = 1                    'Подготовка цикла
50 FOR I = 1 TO K%
60 C = C / X
70 S = S + C
80 NEXT I
90 PRINT "Сумма ряда для "; K%; " слагаемых = "; S
100 END
```

Вычисление суммы ряда по условию

Задача 3.21.

Для произвольного значения аргумента «*x*» (по модулю больше 1) вычислить сумму:

$$S = 1 + 1/x + 1/x^2 + 1/x^3 + \ldots .$$

Вычисления продолжать до тех пор, пока очередное слагаемое больше заданного значения *Z*. Вывести на экран число слагаемых *K*.

Решение.

Для вычисления слагаемых нужен цикл. Так как число слагаемых не определено, то это — итерационный цикл с условием: пока слагаемое больше `Z`. В остальном принцип решения подобен рассмотренному в задаче 3.20. В настоящем примере требуется еще вычислить количество слагаемых. Это тоже делается в цикле по стандартной схеме: `K=K+1`. Безусловно, начальное значение `K` должно быть определено: `K=0`.

Текст в строке 30 поясняет пользователю, в каком диапазоне должно находиться вводимое число. Здесь проверку опускаем, но вообще она нужна.

При выводе последнего слагаемого необходимо учесть, что последнее слагаемое и последний вычисленный член ряда — это не одно и то же. Последнее значение `C` в сумму уже не входит (см. строки 90 и 140).

```
10 'Вычисление суммы ряда по условию
20 INPUT "Аргумент X (по модулю больше 1) = "; X
30 PRINT "Заданное значение Z (не больше "; 1/X; ") = ";
40 INPUT Z
50 K = 0 : S = 0: C = 1     'Подготовка цикла
60 WHILE  C > Z
70 S = S + C
80 K = K + 1
90 C = C / X
100 WEND
110 PRINT " Для аргумента X = "; X;
120 PRINT " члены  ряда больше заданного значения "; Z
130 PRINT " для "; K; " слагаемых и их сумма равна "; S
140 PRINT " Последний член ряда при этом равен ";  C * X
```

Задача 3.22.

Вычислить сумму членов ряда. Суммировать до тех пор, пока разность между текущей и предыдущей суммами больше 0.001. Вывести сумму, последнее слагаемое, вошедшее в сумму, и его номер. Ряд имеет следующий вид:

$$\frac{1^2 \times 5}{2 \times 4} + \frac{2^2 \times 6}{2 \times 4 \times 6} + \frac{3^2 \times 7}{2 \times 4 \times 6 \times 8} + \ldots$$

Решение.

Эта задача подобна предыдущей (3.21), но тут ряд сложнее. При его анализе нетрудно заметить, что знаменатель каждого следующего слагаемого равен знаменателю предыдущего, умноженному на коэффициент, зависящий от порядкового номера слагаемого (строка 90). Числитель не зависит от числителя предыдущего слагаемого: он вычисляется только на основании порядкового номера. Для него отдельную переменную не заводим (строка 100). Если новое слагаемое меньше или равно 0.001, его уже не надо бы прибавлять, но цикл удобно организовать именно так, поэтому мы его прибавляем, но учитываем это при выводе результата (строки 130–150). Для удобства вывода фиксируем вычисленное на предыдущей итерации

слагаемое перед вычислением нового (строка 70) — для этого введена переменная pred (без нее можно обойтись, она нужна только для удобства!).

Определим условие выхода из цикла (окончания суммирования): разность между текущей и предыдущей суммами равна слагаемому, полученному на предыдущей итерации.

```
10 CLS
20 i = 1        'Номер слагаемого
30 x = 5 / 8    'Первое слагаемое
40 S = 5 / 8    'Начальное значение суммы
50 znam = 8     'Начальное значение знаменателя
60 WHILE x > .001
70 pred = x
80 i = i + 1
90 znam = znam * 2 * (i + 1)     'Новый знаменатель
100 x = (i + 4) * i ^ 2 / znam   'Новое слагаемое
110 S = S + x                    'Текущая сумма
120 WEND
130 PRINT "Сумма ="; S - x
140 PRINT "Последнее слагаемое ="; pred
150 PRINT "Номер последнего слагаемого ="; i - 1
160 END
```

Задача 3.23.

Составить программу вычисления приближенного значения суммы элементов ряда

$$S = 1 - x/1! + x^3/3! - x^5/5! + \ldots$$

для произвольного значения аргумента «x» (по модулю меньше 1). Суммирование продолжать до тех пор, пока разность между очередным и предыдущим слагаемыми не станет меньше заданной величины Z (точности вычислений).

Решение.

Задача подобна двум предыдущим. Специфика заключается в условии окончания суммирования и в закономерностях формирования очередного слагаемого.

Разность между соседними слагаемыми может быть как положительной, так и отрицательной, поэтому в условии берем абсолютное значение разности

Знак при элементе не учитываем. В сравнении принимают участие два слагаемых, поэтому два слагаемых надо определить перед входом в цикл (строка 30). Будем переопределять обе связанные с ними переменные в теле цикла (строки 70 и 100) — схема рассмотрена выше.

Проанализируем ряд. Он знакочередующийся. Можно поступить двояко: формировать слагаемое вместе со знаком или все слагаемые считать положительными, а потом четные по порядку вычитать из результата (S), нечетные — прибавлять. Остановимся на втором способе (строка 60). В любом случае не забудем определить исходное значение суммы до входа в цикл: само значение будет определяться в первую очередь слагаемыми, взятыми в качестве отправных точек. Для вычисления порядкового номера слагаемого введем переменную-счетчик (I). Определим закономерность вычисления очередного слагаемого. Тут тоже возможно несколько вариантов. Вот один из них: новое слагаемое равно предыдущему, умноженному на X во 2-й степени и разделенному на произведение (ZNAM-1)*ZNAM (строка 100). Входящая сюда переменная ZNAM определяется до входа в цикл, а потом изменяется в теле цикла (строка 90).

Сложность задач такого рода в том, что все должно быть очень тщательно согласовано; принципиально важна последовательность операторов в геле цикла. К тому же ошибки трудно выявить при отладке: необходимо проводить формальное исполнение программы (см. раздел «Этапы разработки алгоритма»).

```
10 INPUT "X ="; X
20 INPUT "E =": E
30 PREV = 1: CUR = X: S = 1
40 I = 2: ZNAM = 1
50 WHILE ABS(CUR - PREV) >= E
60 IF I MOD 2 = 0 THEN S = S - CUR ELSE S = S + CUR
70 PREV = CUR
80 I = I + 1
90 ZNAM = ZNAM + 2
```

```
100 CUR = CUR * X ^ 2 / (ZNAM - 1) / ZNAM
110 WEND
120 PRINT "Сумма ="; S
130 END
```

Массивы

Понятие массива

Понятие массива рассмотрено в разделе «Объекты алгоритма» (см. также раздел «Объявление объектов программы»). Напомним, что массив — это конечная упорядоченная совокупность данных одного типа, доступ к которым осуществляется по индексу (порядковому номеру). Массивы необходимы, когда требуется несколько раз обращаться к одной и той же группе однотипных данных.

Массивы могут содержать данные любого типа: тип элементов массива, как и в случае простых переменных, распознается по их идентификатору. В отличие от простых переменных, массивы необходимо объявлять перед использованием (с помощью оператора DIM). При объявлении указывают имя массива, размерность и количество элементов по каждой размерности (эти количества должны быть определены до объявления массива).

Использование массивов значительно упрощает работу с группами однотипных данных.

Все действия с массивами выполняются поэлементно, в цикле. В частности, нельзя скопировать один массив в другой или заполнить массив какими-либо (пусть даже и одинаковыми) значениями в один прием. Поскольку массив — это последовательность с известным числом элементов, удобнее использовать цикл FOR; WHILE требуется лишь в исключительных случаях. Для многомерных массивов циклы должны быть вложенными.

В данном разделе приведено значительное количество типовых приемов работы с массивами. Рассмотренные здесь задачи уже формализованы: объекты задачи являются объектами алгоритма. Решению неформализованных задач посвящен специальный раздел. Массивы необходимы в задачах со списками: в рассматриваемом варианте Basic'a списки организованы именно при помощи связанных массивов. Кроме того, умение работать с массивами может пригодиться при решении задач на данные символьного типа — строки (см. раздел «Использование данных символьного типа»).

Операторы определения и чтения констант

Вообще, операторы определения и чтения констант могут быть использованы не только в задачах с массивами. Но именно при работе с массивами они зачастую бывают действительно необходимы.

Оператор определения констант. Общая форма оператора:

нс DATA <список констант>.

Здесь:

DATA — ключевое слово — имя оператора;

<список констант> — неограниченный список констант, разделяемых запятыми.

Тип констант может быть любой в любом операторе DATA. В любом операторе DATA могут определяться константы разного типа (в любых сочетаниях). Символьные константы в списках можно указывать в кавычках и без них.

Оператор DATA неисполняемый. Он используется как форма хранения констант, необходимых для

исполнения программы. В программе может быть несколько операторов DATA. Позиция оператора не важна (до или после READ). Примеры записи оператора:

```
20 DATA янв, 31, фев, 28, мар, 31, апр, 30, май, 31, июн, 30
30 DATA июл, 31, авг, 31, сен, 30, окт, 31, ноя, 30, дек, 31
80 DATA 3.1415926, 2.718282, 9.81
90 DATA Понедельник, Вторник, Среда, Четверг, Пятница, Суббота
```

Аналогично можно задать список однотипных значений, которые будут использованы для определения элементов массива. Только надо помнить, что массив — это конечная совокупность элементов одного типа.

Оператор чтения констант. Общая форма оператора (формат оператора):

нс READ <список имен переменных>.

Здесь:

READ — ключевое слово — имя оператора;

<список имен переменных> — список переменных, которым присваиваются значения констант из операторов DATA.

Принцип действия оператора: переменным, указанным в операторе, присваиваются значения очередных констант из списка констант очередного оператора DATA. Типы переменных должны соответствовать типам «считываемых» констант. Примеры записи оператора:

```
150 READ X              'Определяется вещественная переменная
170 READ NAZ$           'Определяется символьная переменная
250 READ MES$, KOL%  'Определяются две переменных разного типа
```

В случае массивов используем цикл, например:

```
60 FOR I = 1 TO N%
70 RHAD ARR(I)
80 NEXT I
```

Ввод и вывод массивов

Практически все задачи с использованием массивов включают этап ввода массивов и очень многие требуют вывода элементов массива на экран. Для этого существуют стандартные приемы: рассмотрим их.

Ввод массива. Ввод массива — это определение значений его элементов. Для ввода используется цикл FOR: в случае одномерных массивов достаточно одного цикла, при вводе многомерных массивов нужны вложенные циклы — по числу измерений. Собственно ввод значений производится в теле цикла. Он может производиться с клавиатуры, с помощью датчика случайных чисел, вычисляться по какой-либо формуле, посредством операторов определения и чтения констант. В последних трех случаях определение значений элементов производится автоматически. Первый случай предполагает подсказку для пользователя, а порой и проверку вводимых значений.

Например:

```
10 M = 5: N = 6
20 DIM A(M), B(N), C(M * N), D$(M), X(M, N)
30 'Пример ввода одномерных массивов разными способами
40 FOR I = 1 TO M          'Использование датчика случайных чисел
50 A(I) = RND(1)
60 NEXT I
70 FOR I = 1 TO N          'Вычисление элемента
80 B(I) = I * I
90 NEXT I
100 FOR I = 1 TO M * N     'Ввод с клавиатуры
110 PRINT "Введи элемент массива №"; I;
120 INPUT C(I)
130 NEXT I
140 DATA Понедельник, Вторник, Среда, Четверг, Пятница
150 FOR I = 1 TO M         'Использование оператора чтения констант
160 READ D$(I)
170 NEXT I
180 'Пример ввода двумерного массива с клавиатуры
190 FOR I = 1 TO M
200 FOR J = 1 TO N
210 PRINT "Элемент "; I; "-й строки, "; J; "-го столбца = ";
220 INPUT X(I, J)
230 NEXT J
240 NEXT I
```

Замечание: в примере вычисления элемента использована простейшая формула: на практике она может иметь любую степень сложности.

Вывод массива. Вывод массива — это распечатка его элементов на экране. Для этого используют циклы FOR: один для одномерных, вложенные по числу измерений — для многомерных.

Одномерный массив распечатывают, как правило, в строку, реже — в столбик.

Двумерный массив выводят в виде таблицы, где элементы каждого столбца располагаются друг под другом. Для этого следует использовать зональный вывод (разделитель в операторе PRINT — запятая) или форматированный вывод. Часто при выводе указывают номера строк и столбцов.

В любом случае при распечатке нужно выводить пояснительный текст. Для правильного и красивого вывода нужно четко представлять себе работу оператора PRINT и роль разделителей (см. раздел «Оператор вывода»).

Выведем массивы из примера, приведенного в предыдущем разделе:

```
.................
250 'Вывод одномерного массива (в строку)
260 PRINT "Массив вещественных чисел A"
270 FOR I = 1 TO M
280 PRINT A(I);      'Для вывода в столбец убрать ";"
290 NEXT I
300 'Простой вывод двумерного массива (без индексов)
310 PRINT "Массив вещественных чисел X"
320 FOR I = 1 TO M
330 FOR J = 1 TO N
340 PRINT X(I, J),
350 NEXT J
360 PRINT
370 NEXT I
.................
```

«Пустой» PRINT в строке 360 нужен для перехода на новую строку экрана после распечатки каждой строки массива.

Иногда бывает нужно вывести массив, поменяв местами строки со столбцами. В этом случае внешний цикл организуется по столбцам, а внутренний по строкам (см. задачу 3.59).

Общие типовые задачи на массивы

В этом разделе рассмотрены задачи, распространяющиеся как на одно-, так и многомерные массивы. Иными словами, число измерений в них отражается только на количестве вложенных циклов в конструкции. Задачи этой группы решаются подобно задачам на числовые последовательности (см. разделы «Циклические операторы» и «Решение типовых задач»). Разница заключается лишь в способе хранения информации. Там числа вводятся последовательно в единственную переменную, в случае массивов — пребывают в оперативной памяти все одновременно.

Приемы, использованные в задачах данного раздела, могут оказаться полезны при решении задач без использования массивов.

Для экономии места в примерах, где это допустимо, используются простейшие ввод и вывод.

Задачи на вычисление суммы, произведения, количества, среднего арифметического, максимума, минимума элементов массива

Задача 3.24. Среднее арифметическое элементов (по условию).

Дан одномерный массив из N элементов. Посчитать среднее арифметическое элементов, кратных трем.

Решение.

См. задачи 3.7 и 3.9.

Поскольку речь идет о проверке чисел на кратность, они должны быть целыми: если применить операцию MOD к числу, имеющему дробную часть,

результат будет вычислен не для введенного числа, а для числа, получаемого при округлении введенного до большего. Таким образом, в кратные трем попадут 5.8, 9.3 и т. п. Значит нужно либо организовать проверку вводимых значений, либо использовать целочисленный массив.

Ввод и суммирование можно было бы осуществлять в одном цикле, но объединять разные этапы алгоритма не желательно*.

Представляет интерес вывод результата (строки 110–120). Вместо одного условного оператора в полной форме либо двух — в сокращенной использован лишь один в сокращенной в сочетании с оператором безусловного перехода. Оператор IF в строке 120 не нужен, поскольку мы попадаем в нее лишь если ложно условие в строке 110. Этот прием широко используется в «классическом» Basic'е.

```
10 N = 10
20 DIM A%(N)
30 FOR I = 1 TO N
40 PRINT "Введи элемент массива №"; I;
50 INPUT A%(I)
60 NEXT I
70 S = 0: KOL = 0
80 FOR I = 1 TO N
90 IF A%(I) MOD 3 = 0 THEN S = S + A%(I): KOL = KOL + 1
100 NEXT I
110 IF KOL = 0 THEN PRINT "Кратных трем нет": GOTO 130
120 PRINT "Ср. ар. чисел, кратных трем = "; S / KOL
130 END
```

Задача 3.25. Нахождение максимума и его позиции.

Определить в двумерном массиве максимальный элемент и его координаты, считая, что он единственный.

Решение.

См. задачи 3.10–3.12. При определении максимумов и минимумов в одномерном массиве в качестве

*Справедливость этого станет ясна при переходе к процедурному программированию.

исходного значения максимума (минимума) принимают значение первого элемента. Если массив многомерный — элемент, индексы которого по всем измерениям равны 1. Соответственно перед входом в цикл определяем исходные значения координат (индексов): это 1. Количество вложенных циклов, как всегда, определяется количеством измерений массива.

```
10 INPUT "Количество строк ="; M%: IF M% < 1 THEN 10
20 INPUT "Количество столбцов ="; N%: IF N% < 1 THEN 20
30 DIM AR(M%, N%)
40 FOR I = 1 TO M%
50 FOR J = 1 TO N%
60 PRINT "Элемент "; I; "-й строки, "; J; "-го столбца = ";
70 INPUT AR(I, J)
80 NEXT J
90 NEXT I
100 MAX = AR(1, 1): IMAX = 1: JMAX = 1
110 FOR I = 1 TO M%
120 FOR J = 1 TO N%
130 IF AR(I, J) >= MAX THEN MAX = AR(I, J): IMAX = I: JMAX = J
140 NEXT J
150 NEXT I
160 PRINT "Значение максимального элемента"; MAX
170 PRINT "Он расположен в строке №"; IMAX; ", столбце №"; JMAX
180 END
```

Анализ элементов массива

Задача 3.26. Поиск в массиве элементов с заданным значением.

Проверить, есть ли в целочисленном трехмерном массиве элемент с заданным значением.

Решение.

См. задачу 3.15.

Возможно лишь два ответа на поставленный вопрос — «да» и «нет», следовательно, используем переменную-флажок, которую обнуляем до входа во вложенные циклы (три яруса). Указанное значение вводим с клавиатуры. Просматриваем массив, проверяя его элементы на равенство указанному значению. Если находим равенство, флажку присваиваем значение 1 и

выходим из цикла: ответ известен, дальше можно не смотреть.

Данный алгоритм называется алгоритмом линейного поиска числа в массиве: элементы просматривают по порядку; часто по условию требуется указать координаты найденного элемента (см. задачу 3.25).

Если же в задаче требуется указать количество элементов с заданным значением, используем не флажок, а переменную kol (количество), которую наращиваем на 1 при каждом совпадении. Тут нужно пройти весь массив.

```
10 M% = 5: N% = 7: P% = 4
20 DIM AR%(M%, N%, P%)
30 FOR I = 1 TO M%
40 FOR J = 1 TO N%
50 FOR L = 1 TO P%
60 INPUT "Очередной элемент массива ="; AR%(I, J, L)
70 NEXT L, J, I
80 INPUT "Целое число ="; X%
90 FLAG = 0
100 FOR I = 1 TO M%
110 FOR J = 1 TO N%
120 FOR L = 1 TO P%
130 IF AR%(I, J, L) = X% THEN FLAG = 1: GOTO 150
140 NEXT L, J, I
150 IF FLAG = 1 THEN PRINT "Число есть" ELSE PRINT "Числа нет"
160 END
```

Задача 3.27. Все ли элементы массива разные?

Проверить, все ли элементы одномерного массива разные.

Решение.

Возможно лишь два ответа на поставленный вопрос, следовательно, потребуется флажок.

Каждый элемент надо сравнить с каждым, поэтому используем вложенные циклы FOR. Во внешнем цикле фиксируем поочередно элементы массива. Зафиксировав некоторый элемент, начинаем перебирать элементы, располагающиеся после него, и сравнивать их. Допустим, в массиве 5 элементов. Тогда сравнение

будет проходить в такой последовательности: 1 — 2, 1 — 3, 1 — 4, 1 — 5, 2 — 3, 2 — 4, 2 — 5, 3 — 4, 3 — 5, 4 — 5. Легко заметить, что во внешнем цикле достаточно дойти до предпоследнего элемента.

Если массив одномерный, все просто, если нет, самое разумное — в одномерный его преобразовать: так проверять удобнее. Иначе в два раза возрастает кратность вложения циклов.

```
10 INPUT "Количество элементов =": N%: IF N% < 1 THEN 10
20 DIM AR(N%)
30 FOR I = 1 TO N%
40 PRINT "Элемент #"; I;
50 INPUT AR(I)
60 NEXT I
70 FLAG = 0
80 FOR I = 1 TO N - 1
90 FOR J = I + 1 TO N
100 IF AR(I) = AR(J) THEN FLAG = 1: GOTO 120
110 NEXT J, I
120 IF FLAG = 0 THEN PRINT "Все"; ELSE PRINT "Не все";
130 PRINT " разные"
140 END
```

Задача 3.28. Составить перечень элементов массива.

Составить перечень значений элементов одномерного целочисленного массива.

Решение.

Надо создать одномерный массив для записи результата: каждый элемент в нем будет уникален. При выборе размерности следует исходить из предположения, что в перечень войдут все элементы исходного массива (в исходном все элементы окажутся разными). Потребуется отдельный счетчик для формирования результирующего массива. Изначально его значение равно нулю, конечное значение будет соответствовать реальному числу элементов в перечне.

В цикле FOR просматриваем поочередно все элементы исходного массива (если исходный массив многомерный, для этого потребуются вложенные циклы).

Зафиксировав текущий элемент, сравниваем его с каждым из значений, уже имеющихся в перечне: для этого необходим еще один вложенный цикл — в нашем примере по J. Если элемент в перечне уже есть, переходим к следующему элементу, досрочно выходя из внутреннего цикла при помощи оператора безусловного перехода (строка 100). Если нет, приращиваем на 1 значение счетчика и заносим элемент в перечень (строки 120–130). Счетчик независим от параметров обоих циклов! Следует обратить внимание на конечное значение параметра во внутреннем цикле — это количество элементов в перечне на данный момент. По мере формирования перечня оно будет изменяться. Естественно, при анализе первого элемента исходного массива в перечне еще пусто, K=0, внутренний цикл не будет исполнен ни разу, а рассматриваемый элемент сразу запишется в перечень.

Вывод результата — распечатка перечня.

```
10 INPUT "Количество элементов ="; N%: IF N% < 1 THEN 10
20 DIM AR%(N%), R%(N%)
30 FOR I = 1 TO N%
40 PRINT "Элемент №"; I;
50 INPUT AR%(I)
60 NEXT I
70 K% = 0
80 FOR I = 1 TO N%
90 FOR J = 1 TO K%
100 IF AR%(I) = R%(J) THEN 140
110 NEXT J
120 K% = K% + 1
130 R%(K%) = AR%(I)
140 NEXT I
150 PRINT "Перечень элементов массива:"
160 FOR I = 1 TO K%
170 PRINT R%(I);
180 NEXT I
190 END
```

Типовые задачи с двумерными массивами

Вычисления в строках и столбцах массива. К типовым задачам с двумерными массивами относятся

задачи на вычисление суммы, произведения, количества, среднего арифметического, максимума, минимума элементов каждой строки, каждого столбца, заданной строки, заданного столбца массива.

Вычисление названных значений осуществляется при помощи стандартных приемов. Особенность задач данного типа заключается в организации циклов для вычисления. Приведенные ниже методы могут быть использованы для любых многомерных массивов.

С целью экономии места будем рассматривать несколько задач в одной программе.

Задачи 3.29–31.

Ввести двумерный массив $M \times N$. Определить:

1) сумму элементов каждой строки;

2) максимальные значения для каждого столбца;

3) произведение элементов K-й строки (K вводится с клавиатуры), значение которых лежит в диапазоне от 1 до 9.

Решение.

Произведем стандартный ввод с клавиатуры элементов двумерного массива. Для обработки потребуются вложенные циклы FOR. Задача имеет много общего с задачей 3.7. Принципиальная разница заключается в том, что тут элементы для обработки уже введены.

В 1-м и 2-м заданиях вывод результата можно осуществлять двумя способами: если не предполагается как-либо использовать результат впоследствии, можно выводить результаты по каждой строке (столбцу) по мере вычисления, в теле цикла. Способ хорош лишь в самых простых случаях (он использован в задаче 3.7). Другой способ предполагает запись результатов в отдельный массив с распечаткой его по выходе из цикла вычислений. Естественно, этот массив следует предварительно объявить: число элементов в нем будет равно

числу строк (задание 1) или столбцов (задание 2). Воспользуемся вторым способом.

Самое главное при решении задач такого типа — не перепутать строки со столбцами. Принято, что первый индекс указывает номер строки, второй — столбца.

Логика вычислений такова: если требуется что-то вычислить для каждой строки, значит во внешнем цикле перебираем строки. Для каждого столбца — столбцы. Войдя во внешний цикл, определяем исходное значение искомой величины (например, обнуляем соответствующий элемент результирующего массива сумм), затем входим во внутренний цикл, в теле которого перебираем элементы столбцов (задание 1) или строк (задание 2) и производим требуемые вычисления по стандартным схемам.

Задание 3 отличается от первых двух. Прежде всего с клавиатуры следует ввести номер строки. Проверка в данном случае обязательна: число не должно превышать количества строк и быть меньше 1.

Когда в условии фигурирует фиксированная строка (столбец), следует исходить из того, что как строка, так и столбец в двумерном массиве — не что иное, как одномерный массив (двумерный массив — это одномерный массив одномерных массивов). Значит нам предстоит произвести вычисления в одном измерении, т. е. вложенные циклы не потребуются: будет достаточно одного цикла FOR. Строка зафиксирована, поэтому при указании индексов индекс, указывающий строку, всегда имеет значение K. Важный момент: в строке N% элементов (по числу столбцов), это надо учесть при организации цикла.

В задании 3 потребуются еще две переменные: P — произведение элементов K-й строки, отвечающих заданному условию (не забудем определить исходное значение: P=1), и флажок F. Флажок тут нужен для того, чтобы отслеживать факт появления в массиве чи-

сел, отвечающих заданному условию: их может не оказаться, и это следует учитывать при выводе результата. Определим исходное значение: F=0.

```
10 INPUT "Количество строк ="; M%: IF M% < 1 THEN 10
20 INPUT "Количество столбцов ="; N%: IF N% < 1 THEN 20
30 DIM AR(M%, N%)
40 FOR I = 1 TO M%
50 FOR J = 1 TO N%
60 PRINT "Элемент "; I; "-й строки, "; J; "-го столбца = ";
70 INPUT AR(I,J)
80 NEXT J, I
90 DIM SUM(M%), MAX(N%) 'Результирующие массивы
100 'ЗАДАНИЕ 1
110 FOR I = 1 TO M%
120 SUM(I) = 0
130 FOR J = 1 TO N%
140 SUM(I) = SUM(I) + AR(I,J)
150 NEXT J, I
160 PRINT "Сумма элементов строк:"
170 FOR I = 1 TO M%
180 PRINT SUM(I)
190 NEXT I
200 'ЗАДАНИЕ 2
210 FOR J = 1 TO N%
220 MAX(J) = AR (1,J)
230 FOR I = 2 TO M%
240 IF MAX(J) > AR(I,J) THEN MAX(J) = AR(I,J)
250 NEXT I, J
260 PRINT "Максимальные значения в столбцах:"
270 FOR I = 1 TO N%
280 PRINT MAX(I)
290 NEXT I
300 'ЗАДАНИЕ 3
310 INPUT "Номер строки ="; K%: IF K% < 1 OR K% > M% THEN 310
320 P = 1: F = 0
330 FOR J = 1 TO N%
340 IF AR(K,J) > 0 AND AR(K,J) < 10 THEN P = P * AR(K,J): F = 1
350 NEXT J
360 IF F = 1 THEN PRINT "Произведение чисел равно"; P: GOTO 380
370 PRINT "Чисел, лежащих в указанном диапазоне, в строке нет."
380 END
```

Задачи на вычисление суммы, произведения, количества, среднего арифметического, максимума, минимума элементов на главной и побочной диагонали, а также выше и ниже их в задачах с квадратными матрицами. Задачи данного типа распространяются только на квадратные массивы (матрицы). Если в условии встретилось слово «диагональ», значит речь идет именно о

них. Вычисление указанных значений производится при помощи рассмотренных выше стандартных приемов. Специфика заключается в способе отбора нужных элементов и организации циклов.

Главная диагональ проходит из левого верхнего в правый нижний угол матрицы. Для расположенных на ней элементов справедливо равенство $I = J$, где I и J — индексы строки и столбца. Если элемент лежит ниже главной диагонали, то $I > J$, если выше то $I < J$.

Побочная диагональ проходит из правого верхнего в левый нижний угол матрицы. Для расположенных на ней элементов справедливо равенство $I = N-J+1$, где I и J — индексы строки и столбца, N — количество элементов в строках и столбцах матрицы. Если элемент лежит ниже побочной диагонали, то $I > N-J+1$, если выше то $I < N-J+1$.

Когда требуется обрабатывать элементы диагоналей, достаточно одного цикла FOR: ведь диагональ можно представить как одномерный массив. Один из индексов задается параметром цикла, другой вычисляется на его основе по одной из приведенных выше формул.

Если нас интересуют элементы над (под) диагональю, потребуются вложенные циклы FOR. Далее можно поступить двояко: либо заложить во внутреннем цикле условие проверки индексов (см. формулы), либо организовать циклы так, чтобы рассматривать только требуемые элементы. Нужно уметь пользоваться обоими приемами, поэтому реализуем в задаче оба.

С целью экономии места будем рассматривать несколько задач в одной программе.

Задачи 3.32–34. Вычисления в диагоналях квадратной матрицы.

Ввести двумерный массив $N \times N$. Определить:

1) минимальное значение для элементов, лежащих

на главной диагонали, и максимальные значения для элементов, лежащих на побочной диагонали;

2) произведение элементов, лежащих выше побочной диагонали;

3) среднее арифметическое элементов, лежащих ниже главной диагонали.

Решение.

Приемы, использованные при решении данной задачи, описаны в начале раздела. Добавим лишь, что квадратные матрицы вводят точно так же, как и обычные двумерные массивы.

В задании 2 для выделения нужных элементов производится проверка индексов, в задании 3 нужные элементы выделяются за счет специфической организации цикла.

```
10 INPUT "Количество строк и столбцов ="; N%: IF N% < 3 THEN 10
20 DIM AR(N%,N%)
30 FOR I = 1 TO N%
40 FOR J = 1 TO N%
50 PRINT "Элемент "; I; "-й строки, "; J; "-го столбца = ";
60 INPUT AR(I,J)
70 NEXT J, I
80 'ЗАДАНИЕ 1
90 MIN = AR(1,1): MAX = AR(1,N%)
100 FOR I = 2 TO N%
110 IF AR(I,I) < MIN THEN MIN = AR(I,I)
120 IF AR(I,N% - I + 1) < MAX THEN MAX = AR(I,N% - I + 1)
130 NEXT I
140 PRINT "Минимум на главной диагонали равен"; MIN
150 PRINT "Максимум на побочной диагонали равен"; MAX
160 'ЗАДАНИЕ 2
170 P = 1
180 FOR I = 1 TO N%
190 FOR J = 1 TO N%
200 IF I < N - J + 1 THEN P = P * AR(I,J)
210 NEXT J, I
220 PRINT "Произведение эл-тов над побочной диагональю равно"; P
230 'ЗАДАНИЕ 3
240 S = 0
250 FOR I = 2 TO N%
260 FOR J = 1 TO I - 1
270 S = S + AR(I,J)
280 NEXT J, I
290 PRINT "Среднее арифметическое эл-тов, лежащих ниже ";
300 PRINT "главной диагонали, равно"; S / N%
310 END
```

Работа с «контурами» квадратной матрицы. К этому типу задач относятся задачи на заполнение матриц по контурам, осуществление вычислений для элементов контуров и т. п. При решении таких задач надо подумать, как лучше перемещаться: от центра к периферии или наоборот. Далее необходимо определить, сколько будет контуров. Наконец, необходимо вывести закономерности для определения границ матриц.

Подобные задачи предполагают использование FOR-композиций с большой глубиной вложения.

Задача 3.35. Определение сумм элементов периметров матрицы.

Ввести числовую квадратную матрицу $N \times N$ элементов. Найти суммы элементов на периметрах всех вложенных квадратных подматриц, центр которых совпадает с центром исходной матрицы.

Решение.

Принцип решения строится на том, что сумма элементов любого контура может быть получена по формуле `SK-S`, где `SK` — сумма всех элементов матрицы, ограниченной данным контуром, включительно, а `S` — сумма всех элементов матрицы, ограниченной следующим (по направлению к центру) контуром (вычисление суммы элементов матрицы не составляет труда и рассмотрено ранее). Таким образом, если принять, что сумма элементов минимальной матрицы равна нулю (не содержит элементов), то, наращивая в цикле размер матрицы, можно определить сумму элементов всех контуров. Для работы потребуются две переменные для хранения сумм — `SK` и `S`.

Во внешнем цикле (параметр `K`) будем перебирать контуры. Передвигаться следует от центра к периферии, поэтому параметр цикла будет уменьшаться на единицу (шаг = –1). Число контуров в матрице определяется при помощи операции целочисленного де-

ления по формуле (N+1)\2, где N — число строк и столбцов матрицы. В теле внешнего цикла обнуляем переменную SK для вычисления суммы очередной матрицы, наибольшей на данный момент, и вычисляем верхний и нижний индексы, определяющие границы данной матрицы (строка 100). Далее производится вычисление суммы S ее элементов при помощи вложенных циклов FOR (строки 110–140). Потом на экран выводится разность SK-S, сумма очередного контура. Перед выходом из внешнего цикла переопределяется переменная S (строка 150).

```
10 'Вычисление сумм элементов на периметрах матрицы.
20 N = 6: DIM A(N, N)
30 FOR I = 1 TO N
40 FOR J = 1 TO N
50 PRINT "Элемент "; I; "-й строки, "; J; "-го столбца = ";
60 INPUT AR(I, J)
70 NEXT J, I
80 S = 0
90 FOR K = (N + 1) \ 2 TO 1 STEP -1
100 SK = 0: L = K: R = N - K + 1
110 FOR I = L TO R
120 FOR J = L TO R
130 SK = SK + A(I, J)
140 NEXT J, I
150 PRINT SK - S;
160 S = SK
170 NEXT K
180 END
```

Задачи на преобразования массива. Преобразование массива заключается, как правило, в изменении порядка следования его элементов без изменения значений самих элементов. Задачи данного типа не очень сложны, но весьма разнообразны. В основе их подавляющего числа лежат типовые алгоритмы. Рассмотрим эти алгоритмы.

Задача 3.36. Перестановка двух заданных строк (столбцов).

Поменять местами элементы двух строк в двумерном массиве из $M \times N$ элементов. Номера строк вводятся с клавиатуры.

Решение.

См. задачу 3.31.

Для решения поставленной задачи достаточно одного цикла FOR. При помощи оператора SWAP будем попарно менять элементы двух указанных строк, имеющие одинаковое значение второго индекса (номер столбца). Цикл организуем по числу элементов в строке, т. е. по числу столбцов.

После строки 130 следует обыкновенный вывод двумерного массива (результирующий массив должен быть распечатан). Вывод опущен с целью экономии места.

```
10 INPUT "Количество строк ="; M%: IF M% < 1 THEN 10
20 INPUT "Количество столбцов ="; N%: IF N% < 1 THEN 20
30 DIM AR(M%, N%)
40 FOR I = 1 TO M%
50 FOR J = 1 TO N%
60 PRINT "Элемент "; I; "-й строки, "; J; "-го столбца = ";
70 INPUT AR(I,J)
80 NEXT J, I
90 INPUT "Номера двух строк ="; X%, Y%
100 IF X% < 1 OR X% > M% OR Y% < 1 OR Y% > M% THEN 90
110 FOR J = 1 TO N%
120 SWAP AR(X%, J), AR(Y%, J)
130 NEXT J
.....
```

Подобным образом решается задача на перестановку двух строк.

Задачи 3.37–39. Поворот массива на 90 и 180 градусов.

Ввести массив из $M \times N$ элементов. Сформировать три новых массива, повернув исходный на:

1) 180°;

2) 90° по часовой стрелке;

3) 90° против часовой стрелки.

Решение.

Строго говоря, в данной задаче преобразования не происходит: вместо этого из элементов исходного массива формируются новые массивы с сохранением порядка следования элементов.

При объявлении результирующих массивов необходимо учесть, сколько строк и сколько столбцов получится при повороте на тот или иной угол (строка 100). Так повороты на 90 градусов приведут к тому, что строки и столбцы поменяются местами. С учетом этого следует проектировать вложенные циклы для формирования новых массивов и вывода их на экран.

Далее следует выявить зависимости между индексами исходного и результирующего массивов. Для этого рекомендуется рисовать исходный и результирующий массивы, указывая индекс и количество элементов. Эти закономерности будут использованы при определении значений элементов результирующих массивов (строки 200, 320 и 330).

Квадратную матрицу можно повернуть также относительно главной и побочной диагонали. Закономерности при этом будут следующие:

```
GL(I, J) = ISH(J, I)                          'Для главной диагонали
POB(I, J) = ISH(N% - J + 1, N% - I + 1) 'Для побочной диагонали
```

Тут N% — число строк и столбцов в квадратной матрице.

```
10 INPUT "Количество строк =";  M%: IF M% < 1 THEN 10
20 INPUT "Количество столбцов =";  N%: IF N% < 1 THEN 20
30 DIM AR(M%, N%)
40 FOR I = 1 TO M%
50 FOR J = 1 TO N%
60 PRINT "Элемент "; I; "-й строки, "; J; "-го столбца = ";
70 INPUT AR(I,J)
80 NEXT J, I
90 PRINT "Исходный массив:"
100 FOR I = 1 TO M%
110 FOR J = 1 TO N%
120 PRINT AR(I, J),
```

```
130 NEXT J
140 PRINT
150 NEXT I
160 DIM X(M%, N%), Y(N%, M%), Z(N%, M%)
170 'ЗАДАНИЕ 1
180 FOR I = 1 TO M%
190 FOR J = 1 TO N%
200 X(I, J) = AR(M% - I + 1, N% - J + 1)
210 NEXT J, I
220 PRINT "Поворот на 180 градусов:"
230 FOR I = 1 TO M%
240 FOR J = 1 TO N%
250 PRINT X(I, J),
260 NEXT J
270 PRINT
280 NEXT I
290 'ЗАДАНИЯ 2 и 3
300 FOR I = 1 TO N%
310 FOR J = 1 TO M%
320 Y(I, J) = AR(M% - J + 1, I) 'По часовой стрелке
330 Z(I, J) = AR(J, N% - I + 1) 'Против часовой стрелки
340 NEXT J, I
350 PRINT "Поворот на 90 градусов по часовой стрелке:"
360 FOR I = 1 TO N%
370 FOR J = 1 TO M%
380 PRINT Y(I, J),
390 NEXT J
400 PRINT
410 NEXT I
420 PRINT "Поворот на 90 градусов против часовой стрелки:"
430 FOR I = 1 TO N%
440 FOR J = 1 TO M%
450 PRINT Z(I, J),
460 NEXT J
470 PRINT
480 NEXT I
490 END
```

Замечания.

1. Поскольку квадратная матрица — частный случай двумерного массива, предложенное решение применимо и к квадратным матрицам.

2. В случае квадратных матриц поворот на 90° или 180° можно осуществить и другим способом, в основе которого лежит типовой алгоритм — кольцевой сдвиг одномерного массива (см. задачу 3.53). В этом случае во внешнем цикле осуществляется проход по контурам (периметрам) массива (см. задачу 3.35): элементы каждого контура — это одномерный массив, в ко-

тором производится кольцевой сдвиг (вложенные циклы). Способ позволяет обойтись без создания дополнительного двумерного массива, но промежуточный одномерный массив потребуется.

Задачи 3.40–41. Зеркальное отображение (поворот) массивов относительно горизонтальной и вертикальной осей.

Ввести массив из $M \times N$ элементов. Повернуть его относительно:

1) горизонтальной оси;

2) вертикальной оси.

Дополнительных массивов не создавать.

Решение.

В настоящей задаче создание дополнительных массивов нецелесообразно: любое зеркальное отображение происходит, если поменять местами соответствующие пары элементов: в Basic'е для этого существует оператор SWAP. Задача является развитием задачи 3.36: в первом задании следует попарно поменять местами все строки (1-ю с последней, 2-ю с предпоследней и т. д.), во втором — все столбцы (аналогично). Так как поменять надо не две фиксированные строки, а все строки (столбцы), потребуются вложенные циклы. В первом задании во внешнем цикле определяем пару строк, элементы которых следует поменять местами, во внутреннем производим собственно перемещение элементов строк. Очень важный момент: внешний цикл организуем только до горизонтальной оси массива: в противном случае смена произойдет дважды, а в результате все останется на прежних местах.

Во втором задании во внешнем цикле определяем пару столбцов, во внутреннем меняем местами их элементы. Следует иметь в виду, что в этом задании исходным является массив, полученный в результате поворота относительно горизонтальной оси.

Строки 10--150 в точности соответствуют аналогичным строкам предыдущей задачи, поэтому они опущены.

```
.................
160 'ЗАДАНИЕ 1
170 FOR I = 1 TO M% \ 2
180 FOR J = 1 TO N%
190 SWAP AR(I, J), AR(M% - I + 1, J)
200 NEXT J, I
210 PRINT "После поворота относительно горизонтальной оси:"
220 FOR I = 1 TO M%
230 FOR J = 1 TO N%
240 PRINT AR(I, J),
250 NEXT J
260 PRINT
270 NEXT I
280 'ЗАДАНИЕ 2
290 FOR J = 1 TO N% \ 2
300 FOR I = 1 TO M%
310 SWAP AR(I, J), AR(I, N% - J + 1)
320 NEXT I, J
330 PRINT "После поворота относительно вертикальной оси:"
340 FOR I = 1 TO M%
350 FOR J = 1 TO N%
360 PRINT AR(I, J),
370 NEXT J
380 PRINT
390 NEXT I
400 END
```

Замечание. Решение применимо и к квадратным матрицам.

Задачи 3.42--43. Зеркальное отображение (поворот) квадратных матриц относительно диагоналей.

Ввести квадратную матрицу из $N \times N$ элементов. Повернуть матрицу относительно: 1) главной диагонали; 2) побочной диагонали. Дополнительных массивов не создавать.

Решение.

В настоящей задаче создание дополнительных массивов нецелесообразно: любое зеркальное отображение происходит, если поменять местами соответствующие пары элементов при помощи оператора SWAP.

Так как в перемещении принимают участие все элементы, кроме тех, что расположены на соответствующих диагоналях, потребуются вложенные циклы. Во внешнем цикле задаем номер строки, во внутреннем — номер столбца. Элемент с этими индексами меняем местами с зеркально расположенным относительно указанной диагонали; координаты парного элемента вычисляются в зависимости от того, какая из диагоналей является осью симметрии (строки 190 и 310). Важный момент: и внешний, и внутренний циклы организуем так, чтобы проход элементов осуществлялся лишь до нужной диагонали (над или под нею — не важно): в противном случае смена произойдет дважды, а в результате все останется на прежних местах. В предлагаемой программе «первичными» являются элементы под главной и над побочной диагональю.

```
10-150 'Ввод и распечатка исходной квадратной матрицы AR(N%,N%)
160 'ЗАДАНИЕ 1
170 FOR I = 2 TO N%
180 FOR J = 1 TO I - 1
190 SWAP AR(I, J), AR(J, I)
200 NEXT J, I
210 PRINT "После поворота относительно главной диагонали:"
220-270 'Вывод результата - см. предыдущую задачу
280 'ЗАДАНИЕ 2
290 FOR I = 1 TO N% - 1
300 FOR J = 1 TO N% - I
310 SWAP AR(I, J), AR(N% - J + 1, N% - I + 1)
320 NEXT J, I
330 PRINT "После поворота относительно побочной диагонали:"
340-390 'Вывод результата - см. предыдущую задачу
400 END
```

Иногда по условию требуется формирование дополнительного массива: сущность решения описана в задачах 3.37–39.

Задача 3.44. Преобразование двумерного массива в одномерный.

Ввести числовой массив (M строк, N столбцов). Преобразовать его в одномерный массив из $M \times N$ элементов.

Решение.

Ввод и вывод массивов в этой задаче стандартные, поэтому опускаем эти этапы.

Собственно решение может быть осуществлено одним из двух способов. В первом варианте задаем независимый счетчик K для результирующего одномерного массива. Во втором варианте значение очередного элемента одномерного массива вычисляем по формуле `REZ(N%*(I-1)+J)=ISH(I,J)` при исходном массиве из M% строк и N% столбцов.

Вариант 1:

```
.................
100 DIM REZ(M% * N%)
110 K = 1
120 FOR I = 1 TO M%
130 FOR J = 1 TO N%
140 REZ(K) = ISH(I, J)
150 K = K + 1
160 NEXT J, I
.................
```

Вариант 2:

```
.................
100 DIM REZ(M% * N%)
110 FOR I = 1 TO M%
120 FOR J = 1 TO N%
130 REZ(N% * (I - 1) + J) = ISH(I, J)
140 NEXT J, I
.................
```

Обратной данной задаче является преобразование одномерного массива в двумерный (задача 3.49).

Типовые задачи с одномерными массивами

Сортировка. Среди алгоритмов обработки массивов немаловажную роль играют алгоритмы сортировки. Сортировка (упорядочение) — это изменение порядка следования элементов в зависимости от их значения. Простейшие случаи сортировки:

по возрастанию ($AR(I) < AR(I + 1)$);

по убыванию ($AR(I) > AR(I + 1)$);

по неубыванию ($AR(I) <= AR(I + 1)$);

по невозрастанию ($AR(I) >= AR(I + 1)$).

$AR(I)$ и $AR(I + 1)$ здесь любая пара соседних элементов массива AR.

Существует множество алгоритмов сортировки, но нет идеального: либо алгоритм прост, краток и понятен, но медленно работает, либо работает быстро, но сложен. В данном пособии ограничимся рассмотрением трех наиболее простых методов. Ввод и вывод массивов в приводимых ниже примерах стандартные, поэтому опускаем эти этапы. Во всех трех задачах требуется сделать одно и то же, но разными методами — для наглядности. Чтобы лучше разобраться в том, как осуществляется сортировка различными методами, рекомендуется проводить формальное исполнение алгоритмов.

Задача 3.45. Сортировка по методу прямого выбора.

Ввести одномерный числовой массив R из N элементов. Отсортировать его по возрастанию с использованием метода прямого выбора.

Решение.

Это наиболее простой для понимания метод. Сущность его заключается в следующем: ищем в массиве минимальный элемент по стандартной схеме, фиксируя его позицию. Затем ставим его на место, меняя его местами с элементом № 1 (используем для этого оператор SWAP). Итак, 1-й элемент нас больше не интересует: в этой позиции располагается то, что нужно. Просматриваем массив, начиная со 2-го элемента, чтобы найти минимальное значение из оставшихся элементов. Выявив его и его позицию, меняем это значение со значением 2-го элемента. И так далее — в цикле FOR. Внешний цикл предназначен для последовательной фиксации элементов массива, во внутреннем осуществляется поиск минимума и его позиции. После выхода из внутреннего цикла следует перестановка

элементов. Последний элемент во внешнем цикле не рассматривается: он сам встанет на свое место.

```
.....................
50 FOR I = 1 TO N - 1
60 MIN = R(I): MINPOS = I
70 FOR J = I + 1 TO N
80 IF R(J) < MIN THEN MIN = R(J): MINPOS = J
90 NEXT J
100 SWAP R(I), R(MINPOS)
110 NEXT I
.....................
```

Задача 3.46. Сортировка по методу прямого обмена. Ввести одномерный числовой массив *R* из *N* элементов. Отсортировать его по возрастанию с использованием метода прямого обмена.

Решение.

В данном методе во внешнем цикле поочередно фиксируются элементы массива, а во внутреннем осуществляется их сравнение с остальными элементами массива, имеющими порядковый номер выше, чем у зафиксированного во внешнем цикле. Например, на 3-ей итерации внешнего цикла элемент `R(3)` будет сравниваться с элементами `R(4)`, `R(5)`, `R(6)` и т.д. Если в результате сравнения будет выявлено, что зафиксированный во внешнем цикле элемент (`R(I)`) больше элемента с индексом, определенным во внутреннем цикле (`R(J)`), они поменяются местами. В результате первой итерации наименьший элемент окажется в позиции с индексом 1. И так далее — пока все элементы не обретут подобающие им места. О последнем элементе во внешнем цикле заботиться не надо.

```
.....................
50 FOR I = 1 TO N - 1
60 FOR J = I + 1 TO N
70 IF R(J) < R(I) THEN SWAP R(I), R(J)
80 NEXT J, I
.....................
```

Задача 3.47. Пузырьковая сортировка.

Ввести одномерный числовой массив *R* из *N* элементов. Отсортировать его по возрастанию с использованием пузырьковой сортировки.

Решение.

Этот метод основан на сравнении соседних элементов. «Неправильно» расположенные по отношению друг к другу элементы меняются местами. Во вложенных циклах поочередно фиксируем пару соседних элементов массива. В результате первого прохода элемент с минимальным значением оказывается в первой позиции массива (всплывает, подобно пузырьку).

Следует заметить, что данный метод асимметричен: если «легкие» элементы оказываются на месте за один проход, то «тяжелые» смещаются за проход лишь на одну позицию. Алгоритм будет работать быстрее, если чередовать направления проходов*.

```
.....................
50 FOR I = 1 TO N - 1
60 FOR J = N TO I + 1 STEP -1
70 IF R(J - 1) < R(J) THEN SWAP R(J - 1), R(J)
80 NEXT J, I
.....................
```

Слияние упорядоченных массивов. Рассмотренный ниже типовой алгоритм применим только к отсортированным массивам. С незначительными изменениями он используется при решении задач, требующих слияния двух упорядоченных массивов в третий, также упорядоченный. При этом исходные и результирующий массивы могут быть упорядочены по-разному. Главное условие — упорядоченность. В отдельных задачах формировать третий массив не требуется — достаточно распечатать элементы исходных отсортированных последовательностей в определенном порядке.

*Это усовершенствование реализовано в «шейкерном» методе сортировки.

Задача 3.48. Слияние массивов.

Ввести два одномерных числовых массива из M и N элементов, упорядоченных по возрастанию. Сформировать из элементов обоих массивов упорядоченный по возрастанию третий массив из $(M + N)$ элементов, не используя сортировку последнего.

Решение.

Поскольку алгоритм работает только на отсортированных массивах, при вводе элементов исходных массивов обязательна проверка на упорядоченность (строки 60 и 100).

Первый этап слияния: просматривая элементы исходных массивов, каждый раз пересылаем в третий наименьший из них. Соответственно приращиваем индекс того из исходных массивов, элемент которого помещен в массив-результат. В любом случае увеличиваем на единицу счетчик элементов в третьем массиве. Процесс закончим, как только будет исчерпан какой-либо из массивов. Какой из массивов подойдет к концу прежде, предугадать невозможно, поэтому используем итерационный цикл (это один из редких случаев, когда для работы с массивами используются циклы WHILE). Сформулируем условие выполнения цикла: пока не исчерпан ни один массив, т. е. текущие индексы обоих массивов не достигли своих предельных значений (строка 130).

Когда один из массивов заканчивается, в другом остаются элементы, еще не вошедшие в результирующий массив. Их надо туда дослать. Поскольку мы не знаем, в каком из исходных массивов остались неоприходованные элементы, прописываем этап досылки для обоих: будет исполнен только цикл для неисчерпанного массива.

```
10 INPUT "Размерность массива A ="; M%: IF M% < 1 THEN 10
20 INPUT "Размерность массива B="; N%: IF N% < 1 THEN 20
30 DIM A(M%), B(N%), C(M% + N%)
40 FOR I = 1 TO M%
```

```
50 INPUT "Очередной элемент массива A ="; A(I)
60 IF I > 1 AND A(I) < A(I - 1) THEN 50
70 NEXT I
80 FOR I = 1 TO N%
90 INPUT "Очередной элемент массива B ="; B(I)
100 IF I > 1 AND B(I) < B(I - 1) THEN 90
110 NEXT I
120 I = 1: J = 1: K = 1  'Индексы массивов A, B и C
130 WHILE I <= M% AND J <= N%
140 IF A(I)<B(J) THEN C(K)=A(I): I=I+1 ELSE C(K)=B(J): J=J+1
150 K = K + 1
160 WEND
170 WHILE I <= M%          'Пересылка "хвоста" массива A
180 C(K) = A(I): I = I + 1: K = K + 1
190 WEND
200 WHILE J <= N%          'Пересылка "хвоста" массива B
210 C(K) = B(J): J = J + 1: K = K + 1
220 WEND
230 PRINT "Результирующий массив:"
240 FOR I = 1 TO M% + N%: PRINT C(I);: NEXT
250 END
```

Задача 3.49. Преобразование одномерного массива в двумерный. Ввести числовой одномерный массив из $M \times N$ элементов. Преобразовать его в двумерный массив (M строк, N столбцов).

Решение.

Данная задача обратна задаче преобразования двумерного массива в одномерный (3.44). Аналогично существуют два варианта решения.

Ввод и вывод массивов в этой задаче стандартные, опустим эти этапы.

Вариант 1:

```
.................
100 DIM REZ(M%, N%)
110 K = 1
120 FOR I = 1 TO M%
130 FOR J = 1 TO N%
140 REZ(I, J) = ISH(K)
150 K = K + 1
160 NEXT J, I
.................
```

Вариант 2:

```
.................
100 DIM REZ(M%, N%)
110 FOR I = 1 TO M%
```

```
120 FOR J = 1 TO N%
130 REZ(I, J) = ISH(N% * (I - 1) + J)
140 NEXT J, I
.................
```

Уплотнение массива. Уплотнение массива — это удаление из него элементов, отвечающих тем или иным условиям. Образующиеся пустоты заполняются за счет сдвига всех оставшихся элементов*.

Если в задаче не наложено ограничение на использование дополнительного массива, то самый простой способ — создать дополнительный массив того же размера, что и исходный (возможно, в исходном массиве не окажется подлежащих удалению элементов). В цикле просматриваем исходный массив, переписывая в новый «ценные» элементы, согласно условию, и каждый раз наращивая счетчик: для нового массива он должен быть независимым (как в варианте 1 задачи 3.49). Конечное значение этого счетчика — реальное число элементов в новом массиве. Это простой, но не очень рациональный метод: расходуется место в памяти на дублирование массива. Предпочтительнее способ без формирования дополнительного массива (в задачах, как правило, это оговорено). Он заключается в следующем: просматриваем в цикле исходный массив, находя элемент на выброс, последовательно сдвигаем влево оставшиеся элементы. И считаем, сколько выбросили. Трудность в том, что в Basic'е в цикле FOR условия выполнения цикла определяются единожды — при входе в цикл. А массив-то укорачивается. Это значит, нужно использовать цикл WHILE. В нижеследующей задаче приводится алгоритм, соответствующий второму способу.

*Физически никаких пустот не образуется: когда в некоторый участок оперативной памяти заносится новое значение, прежнее утрачивается. На место «лишнего» элемента записывается следующий за ним.

Задача 3.50. Уплотнение массива.

Ввести одномерный массив из N% элементов. Удалить из него числа, кратные трем. Дополнительный массив не создавать.

Решение

Принцип решения изложен выше.

Ввод элементов в массив следует проводить с проверкой: они не должны содержать дробной части, иначе операция получения остатка от деления даст некорректный результат (см. задачу 3.24).

Во внешнем цикле (как пояснено выше, это должен быть не FOR, а WHILE) перебираем поочередно имеющиеся в массиве элементы. Изначально их N штук. В теле этого цикла проверяем, кратно ли число трем. Если нет, все остается на своих местах: следует увеличить на 1 параметр цикла WHILE (см. соотношение FOR и WHILE в разделе «Циклы с неизвестным числом повторений») и перейти к следующему числу массива, используя оператор безусловного перехода. Если число с индексом I подлежит удалению (кратно трем), нужно на его место записать значение элемента с индексом (I+1), на его место — следующее и т.д. Для этого смещения потребуется вложенный цикл. На данном этапе число N остается фиксированным, поэтому можно использовать FOR. Позиции до I в сдвиге не участвуют. Передвинув элементы, уменьшим на 1 значение N: удалили один элемент. И в этом случае необходимо увеличить на 1 параметр цикла WHILE.

Ниже приведен фрагмент программы (без ввода и вывода).

```
.................
100 I = 1
110 WHILE I <= N
120 IF A(I) MOD 3 <> 0 THEN I=I+1: GOTO 170
130 FOR J = I TO N - 1
140 A(J) = A(J + 1)
150 NEXT J
```

```
160 N = N - 1
170 WEND
```

Вставка элемента в массив. Эта задача обратна предыдущей, но обе они содержат один и тот же прием — смещение группы элементов на одну позицию. При уплотнении сдвиг производится влево, при вставке — вправо.

При вставке возникает проблема, с которой мы не сталкиваемся при уплотнении: как быть с последними элементами? Когда в массиве инициализированы (определены, участвуют в работе) все элементы, «хвост» остается только вытеснить: значения при этом будут утрачены. Иначе нужно создавать дополнительный массив, что не всегда допускается по условию. Если известно, сколько элементов будет вставлено, можно объявить массив соответствующего размера, инициализировав все элементы, кроме последних. Выбор типа цикла для просмотра элементов (FOR или WHILE) зависит от конкретного случая.

Задача 3.51. Введение элементов в массив.

Дан одномерный массив, упорядоченный по возрастанию. С клавиатуры вводится число. Вставить это число в массив, не нарушая упорядоченности. Последний элемент вытеснить.

Решение.

Работаем в цикле FOR: «хвост» подлежит вытеснению, поэтому число элементов в массиве остается неизменным. Перебираем исходный массив — ищем место для нового значения, сравнивая его с текущим элементом массива. Если введенное значение больше текущего элемента, переходим к следующему элементу. В противном случае место найдено. Теперь позицию надо освободить, сместив на единицу вправо элементы, стоящие в позициях, начиная с той, куда

планируется поместить новое значение. Тут сдвиг надо начинать от последнего элемента (шаг –1). Последний элемент вытесняется. После сдвига в нужное место можно поместить введенное значение.

```
.................
100 INPUT "Вставить в массив значение ="; X
110 FOR I = 1 TO N
120 IF X > A(I) THEN 180
130 FOR J = N TO I + 1 STEP -1
140 A(J) = A(J - 1)
150 NEXT J
160 A(I) = X
170 I = N
180 NEXT I
.................
```

Изменение положения элементов на некотором отрезке. Задача довольно проста, но используемый в ней прием нередко применяется в более сложных алгоритмах.

Задача 3.52. Перестановка элементов на некотором отрезке.

Ввести одномерный массив из *N* элементов. Изменить порядок следования значений элементов на обратный от позиции BEG до позиции FIN.

Решение.

Очень важен контроль ввода: границы диапазона должны быть целыми, не выходить за пределы индексов массива, а также нижняя граница должна быть меньше верхней (см. задачу 3.19).

В данной задаче есть две тонкости. Во-первых, надо поменять местами пары элементов лишь один раз, иначе ничего не изменится. Поэтому исполняем цикл только для половины диапазона: `(BEG%+FIN%)\2`. Во-вторых, индекс левого элемента на 1 увеличивается, а правого на столько же уменьшается — отсюда формула вычисления индекса парного элемента для `J`-го в строке 130.

```
.................
100 INPUT "Начальная и конечная позиции ="; BEG%, FIN%
110 IF BEG% < 1 OR FIN% > N% OR BEG% > FIN% THEN 100
120 FOR J = BEG% TO (BEG% + FIN%) \ 2
130 SWAP A(J), A(FIN% + BEG% - J)
140 NEXT J
.................
```

Замечание. Эту задачу удобно решать также и с помощью цикла WHILE.

Задачи на кольцевой сдвиг элементов массива. Кольцевой сдвиг — это смещение элементов массива вправо либо влево, причем вытесненные элементы занимают освободившиеся в результате смещения позиции в противоположном конце массива — так, словно массив представляет собой кольцо (первый и последний элементы смыкаются). Порядок следования элементов при этом сохраняется.

Задача 3.53. Кольцевой сдвиг.

Ввести одномерный массив из *N* элементов. Произвести кольцевой сдвиг его элементов на *K* позиций. *K* вводят с клавиатуры, оно может быть как положительным, так и отрицательным.

Решение.

Количество элементов задаем программно. Число K нуждается в проверке: оно должно лежать в диапазоне от -(N-1) до (N-1) и не иметь дробной части. Строго говоря, превышение диапазона на качестве работы программы не отразится: просто произойдет дополнительная циркуляция элементов (см. строки 80–90). Но при выходе за нижнюю границу не будет выполняться внешний цикл.

В цикле ввода массива с помощью генератора случайных чисел (см. раздел «Встроенные функции») получаем числа без дробной части, не превышающие 99, и сразу же распечатываем их на экране в строку.

Далее анализируем введенное число. Если это 0, ничего двигать не надо — сразу переходим к выводу результата. Если число отлично от 0, формируем на его основе переменную `sdvig` (строка 80). При отрицательном K происходит дополнение до положительного значения путем прибавления общего количества элементов (так, если массив из 20 элементов сдвинуть на +4 и на –16, получится одно и то же). Теперь все сдвиги положительны, что бы ни ввел пользователь, т. е. — вправо.

Сущность дальнейшего заключается в том, что в массиве во внешнем цикле осуществляется кольцевое смещение на 1 позицию вправо `sdvig` раз. Собственно смещение на 1 элемент осуществляется во внутреннем цикле (см. задачу 3.51). Смещение начинается с последнего элемента. Чтобы не потерялось его значение, его помещают во вспомогательную переменную `tmp`, откуда извлекают и после выхода из внутреннего цикла отправляют в первую позицию.

```
10 N = 20: DIM a(N)
20 INPUT "Сдвиг K ="; K
30 IF K <= -N OR K >= N OR K <> FIX(K) THEN 20
40 FOR i = 1 TO N
50 a(i) = FIX(RND(1) * 99): PRINT a(i);
60 NEXT i
70 IF K = 0 THEN 160
80 IF K > 0 THEN sdvig = K ELSE sdvig = N + K
90 FOR i = 1 TO sdvig
100 tmp = a(N)
110 FOR j = N - 1 TO 1 STEP -1
120 a(j + 1) = a(j)
130 NEXT j
140 a(1) = tmp
150 NEXT i
160 PRINT "Результат сдвига на"; K; "позиций:"
170 FOR i = 1 TO N: PRINT a(i); NEXT i
180 END
```

Задачи, требующие выбора одного или нескольких типовых алгоритмов

Как правило, предлагаемый здесь способ решения не является единственно возможным: чем сложнее задача, тем больше вариантов.

В конце раздела приведены задачи особого рода (3.61–3.62). В условии дается программа: следует указать, что будет выведено на экране в результате ее работы. Разбор подобных задач позволяет глубже прочувствовать особенности работы с массивами.

Задача 3.54.

Ввести одномерный массив из N случайных целых чисел в диапазоне от –5 до +5. Вычислить среднее арифметическое квадратов отрицательных чисел; положительные элементы уменьшить вдвое, отрицательные заменить значениями их индексов. Вывести результирующий массив.

Решение.

Имеем одномерный массив, следовательно, на всех стадиях работы будет нужен цикл FOR без вложений.

Алгоритм может быть разбит на три этапа: 1. Ввод и распечатка исходного массива. 2. Обработка массива. 3. Выдача результата, включающая распечатку среднего арифметического и вывод результирующего массива.

1. Ввод и распечатку можно объединить, поскольку ввод осуществляется при помощи датчика случайных чисел. Функция RND дает числа от 0 до 1, естественно, дробные. Для расширения диапазона будем домножать эти числа на 10. Чтобы числа получались целые, применим функцию FIX. Вычитание результата из числа 5 дает числа в указанном диапазоне — как положительные, так и отрицательные.

2. В задаче требуется вычислить среднее арифметическое, значит потребуется переменная для суммирования (S). Важный момент: среднее определяем не для всех чисел, поэтому удовлетворяющие условию следует считать: нужна переменная для подсчета количества (K). Обнулим эти переменные перед входом в цикл.

В цикле производим анализ значений элементов (+ или −) и указанные действия. Нули не трогаем. Важна последовательность действий: в первую очередь нужно проверить число на положительность, иначе мы не найдем в массиве отрицательных чисел: им будет присвоено значение их индекса, а оно положительно. Далее: если число отрицательно, сперва следует прибавить к сумме его квадрат, а потом уже менять.

3. При выводе среднего проверяем, были ли отрицательные числа, анализируя значение переменной K. Вывод результирующего массива стандартный.

```
10 N = 20: DIM A(N)
20 PRINT "Исходный массив:"
30 FOR I = 1 TO N
40 A(I) = 5 - FIX(RND(1) * 10): PRINT A(I);
50 NEXT I
60 S = 0: K = 0
70 FOR I = 1 TO N
80 IF A(I) > 0 THEN A(I) = A(I) / 2
90 IF A(I) < 0 THEN K = K + 1: S = S + A(I) ^ 2: A(I) = I
100 NEXT I
110 PRINT: PRINT "В исходном массиве ";
120 IF K = 0 THEN PRINT "отрицательных чисел не было": GOTO 140
130 PRINT "среднее квадратов отрицательных чисел ="; S / K
140 PRINT "Результирующий массив:"
150 FOR I = 1 TO N
160 PRINT A(I);
170 NEXT I
180 END
```

Задача 3.55.

В одномерном массиве из N произвольных чисел поменять местами элементы, стоящие равноудаленно от элемента с заданным индексом K. Вывести на экран исходный и результирующий массивы.

Решение.

Количество элементов задаем программно (строка 10).

Этапы решения: 1. Ввод исходного массива и его распечатка на экране. 2. Ввод с проверкой числа K 3. Перестановка элементов. 4. Вывод результирующего массива.

1. Числа произвольные, поэтому для заполнения массива используем датчик случайных чисел.

2. Число K вводится с клавиатуры и нуждается в проверке: оно должно укладываться в диапазон от 1 до N и не иметь дробной части.

3. Строки 80–120. Перестановку элементов производим симметрично относительно K до тех пор, пока не дойдем до одной или обеих границ массива. Удобно осуществлять ее в итерационном цикле, начиная от правого и левого соседей элемента с индексом K. Поменяв местами очередную пару элементов, на 1 уменьшаем левую и на 1 увеличиваем правую границы.

4. Вывод стандартный.

В этой задаче можно использовать для перестановки и цикл FOR.

```
10 N = 13: DIM A(N)
20 PRINT "Исходный массив:"
30 FOR i = 1 TO N
40 A(i) = RND(1): PRINT A(i);
50 NEXT i
60 PRINT : INPUT "Index ="; K
70 IF K < 1 OR K > N OR K <> FIX(K) THEN 60
80 L = K - 1: R = K + 1
90 WHILE L >= 1 AND R <= N
100 SWAP A(L), A(R)
110 L = L - 1: R = R + 1
120 WEND
130 PRINT : PRINT "Результирующий массив:"
140 FOR i = 1 TO N
150 PRINT A(i);
160 NEXT i
170 END
```

Задача 3.56.

В двумерном массиве случайных чисел (*M* строк, *N* столбцов) определить номера столбцов, среднее арифметическое элементов которых меньше среднего арифметического элементов во всем массиве.

Решение.

Количество строк и столбцов задаем программно (строка 10).

Выделим этапы решения: 1. Ввод исходного массива и его распечатка на экране. 2. Вычисление среднего по всему массиву (потребуется переменная для суммы — SAR. Количества в этой задаче считать не надо: средние значения находим для всех элементов). 3. Вычисление средних значений по столбцам. 4. Сравнение средних значений по столбцам с общим средним. 5. Вывод результатов по итогам сравнения.

Каждый из этих этапов — один из типовых алгоритмов, рассмотренных выше: никаких проблем. Но получим очень длинную программу. Кроме того, придется трижды проходить массив с помощью вложенных циклов FOR, создавать дополнительный одномерный массив для хранения средних значений по столбцам, а потом его тоже просматривать. Попробуем упростить.

Прежде всего объединим ввод и распечатку массива с суммированием всех его элементов в одном цикле. Не забудем обнулить значение переменной SAR перед входом в цикл (строка 30). Выйдя из цикла, вычислим общее среднее (строка 120).

Далее: можно объединить 3, 4 и 5 этапы. Вычисление и сравнение вполне логично осуществлять в одном цикле, тем более, что массив средних по столбцам более нигде в программе не используется. А чтобы не загромождать программу дополнительными промежуточными объектами, сразу же будем выводить результат. Не забудем, что внешний цикл должен быть по

столбцам! Для каждого очередного столбца обнуляем исходное значение суммы его элементов (строка 170), во внутреннем цикле суммируем элементы данного столбца; выйдя из внутреннего цикла, сравниваем общее среднее со средним по столбцу и, в зависимости от результата сравнения, распечатываем или не распечатываем индекс столбца и его среднее (распечатка столбиками — используем зонный формат вывода) — строка 210.

Строки 130–150 — распечатка необходимых пояснений к выводу. В данной задаче всегда найдется хотя бы один столбец, удовлетворяющий условию, поэтому не нужно предусматривать вывод для обратной ситуации.

```
10 M = 5: N = 4
20 DIM AR(M, N)
30 SAR = 0
40 PRINT "Исходный массив:"
50 FOR I = 1 TO M
60 FOR J = 1 TO N
70 AR(I, J) = RND(1)
80 SAR = SAR + AR(I, J): PRINT AR(I, J),
90 NEXT J
100 PRINT
110 NEXT I
120 SAR = SAR / M / N
130 PRINT "Ср.ар. во всем массиве ="; SAR
140 PRINT "Столбцы, где ср.ар. меньше общего ср.ар.:"
150 PRINT "Индекс", "Ср.ар."
160 FOR J = 1 TO N
170 S = 0
180 FOR I = 1 TO M
190 S = S + AR(I, J)
200 NEXT I
210 IF SAR > S / M THEN PRINT J, S / M
220 NEXT J
230 END
```

Замечание. Возможны и другие пути упрощения этой задачи.

Задача 3.57.

В двумерном массиве случайных чисел (M строк, N столбцов) переставить строки так, чтобы суммы их элементов возрастали.

Решение.

Количество строк и столбцов задаем программно.

Этапы решения: 1. Ввод исходного массива и его распечатка на экране. 2. Вычисление сумм элементов каждой строки (потребуется одномерный массив для сумм — `S(M)`: все эти значения впоследствии используются). 3. Контрольная распечатка значений сумм элементов строк. 4. Сортировка массива сумм. 5. Перестановка строк массива. 6. Вывод результирующего массива. 7. Контрольная распечатка значений сумм элементов строк.

Каждый из этапов выполняется с помощью типового алгоритма. Усовершенствуем решение.

Объединим 1, 2 и 3 этапы в общем вложенном цикле. Внешний цикл — по строкам: нам надо вычислить суммы элементов каждой строки. В строке 50 обнуляем исходное значение суммы для данной строки*. При выводе, закончив распечатку элементов текущей строки во внутреннем цикле, распечатываем сумму ее элементов — уже во внешнем.

Вывод результата (этапы 6 и 7) произведем аналогично в общем вложенном цикле (строки 220–270).

Сортировка реализуется с помощью любого известного способа. При этом стадии сортировки и перестановки строк не только можно, но нужно объединить. Ведь массив элементов и массив сумм его строк — связанные массивы: элементам строки с индексом *x* соответствует элемент массива сумм с тем же индексом. Значит при сортировке удобно менять местами два элемента массива сумм (строка 150) и тут же переставлять поэлементно две соответствующие строки исходного массива (строки 160–180). Следует обратить внимание на то, как выполняется сравнение двух сумм:

---

*Строго говоря, в Basic'е это делать необязательно: при объявлении массивы инициализируются нулями. Но культура программирования требует не опускать этот этап.

если они расположены «правильно», переходим к следующему элементу массива сумм с помощью оператора безусловного перехода (на строку 190). Строки 150–180 работают только при «неправильном» расположении сравниваемых сумм.

В строках 30 и 210 три запятые подряд нужны для того, чтобы пояснительный текст *Сумма* оказался при выводе именно над столбиком сумм.

```
10 M = 5: N = 4
20 DIM AR(M, N), S(M)
30 PRINT "Исходный массив", , , "Сумма"
40 FOR I = 1 TO M
50 S(I) = 0
60 FOR J = 1 TO N
70 AR(I, J) = RND(1)
80 S(I) = S(I) + AR(I, J): PRINT AR(I, J),
90 NEXT J
100 PRINT S(I)
110 NEXT I
120 FOR I = 1 TO M - 1
130 FOR J = I + 1 TO M
140 IF S(I) <= S(J) THEN 190
150 SWAP S(I), S(J)
160 FOR K = 1 TO N
170 SWAP AR(I, K), AR(J, K)
180 NEXT K
190 NEXT J
200 NEXT I
210 PRINT "Результирующий массив", , , "Сумма"
220 FOR I = 1 TO M
230 FOR J = 1 TO N
240 PRINT AR(I, J),
250 NEXT J
260 PRINT S(I)
270 NEXT I
280 END
```

Возможны и другие варианты решения.

Задача 3.58.

В одномерном массиве из N произвольных чисел изменить порядок следования элементов на обратный на участках слева и справа от элемента с заданным индексом K. Вывести на экран исходный и результирующий массивы.

Решение.

Количество элементов задаем программно (строка 10).

Этапы решения: 1. Ввод исходного массива и его распечатка на экране. 2. Ввод с проверкой числа K. 3. Перестановка элементов. 4. Вывод результирующего массива.

Этапы 1, 2 и 4 в точности соответствуют аналогичным этапам задачи 3.54.

Строки 80–150. Перестановку элементов производим в соответствии со стандартным алгоритмом, рассмотренным в задаче 3.52. Элементы на левом и правом отрезках массива следует переставлять отдельно, в обоих случаях определим начало и конец интервала перед входом в соответствующий цикл. В 80–110 строках перемещаем элементы слева, в 120–150 — справа от K-го элемента. Сам элемент остается на прежнем месте, отсюда — значения переменных `beg` и `fin`, в которые заносим начало и конец отрезка.

```
10 N = 11: DIM A(N)
20 PRINT "ISH ar"
30 FOR I = 1 TO N
40 A(I) = FIX(RND(1) * 5): PRINT A(I);
50 NEXT I
60 PRINT : INPUT "Index ="; k
70 IF k < 1 OR k > N OR k <> FIX(k) THEN 60
80 beg = 1: fin = k - 1
90 FOR I = beg TO (beg + fin) \ 2
100 SWAP A(I), A(beg + fin - I)
110 NEXT I
120 beg = k + 1: fin = N
130 FOR I = beg TO (beg + fin) \ 2
140 SWAP A(I), A(beg + fin - I)
150 NEXT I
160 PRINT : PRINT "Rez ar"
170 FOR I = 1 TO N
180 PRINT A(I);
190 NEXT I
200 END
```

Задача 3.59.

Сформировать матрицу 7 × 7 элементов, заполненную согласно схеме 24. Вывести ее на экран, развернув на 90° против часовой стрелки.

Решение.

Количество элементов задаем программно.

Алгоритм включает два этапа: 1. Программное заполнение матрицы. 2. Распечатка матрицы согласно условию.

1	14	15	28	29	42	43
2	12	16	27	30	41	44
3	12	17	26	31	40	45
4	11	18	25	32	39	46
5	10	19	24	33	38	47
6	9	20	23	34	37	48
7	8	21	22	35	36	49

Схема 24. Матрица

1. Матрица представляет собой двумерный массив, поэтому для заполнения (и распечатки) используем вложенные циклы FOR. Проанализировав схему 24, нетрудно заметить: элементы следуют от столбца к столбцу «змейкой», а также отличаются от соседей на единицу. Заполнение удобно проводить, организовав внешний цикл по столбцам (по вертикали), а внутренний — по строкам. Внешний цикл при этом будет обычный: монотонно продвигаемся от 1-го столбца к 7-му. Но в пределах столбца элементы расположены то по возрастанию, то по убыванию, поэтому заголовок внутреннего цикла предстоит менять от столбца к столбцу: изменяться будут начальное (`beg`) и конечное (`fin`) значения параметра цикла (то от 1 до 7, то от 7 до 1) и, естественно, шаг `d` (то +1, то −1). Зададим начальные значения указанных переменных перед входом в FOR-комбинацию; изменять их будем после прохода очередного столбца (строка 90).

43	44	45	46	47	48	49
42	41	40	39	38	37	36
29	30	31	32	33	34	35
28	27	26	25	24	23	22
15	16	17	18	19	20	21
14	13	12	11	10	9	8
1	2	3	4	5	6	7

Схема 25. Последовательность чисел при выводе на экран

Наименьшее число — один, с него начнем, присвоив перед входом в цикл это значение переменной `el`. Присвоив очередному элементу значение переменной `el` во внутреннем цикле, переопределяем значение `el`, увеличивая его на 1 (строки 60–70).

Существуют и другие способы заполнения матрицы согласно предложенному рисунку.

2. Следует иметь в виду, что на этой стадии преобразовывать саму матрицу не надо: нужно только «нетрадиционно» вывести ее. В результате вывода элементы должны быть расположены так, как показано на схеме 25. Как видно из обоих рисунков, элементы столбцов распечатываются по горизонтали, от большего номера к меньшему (отсюда — внешний цикл, строка 110), а строк — по вертикали, в порядке нарастания номера строки (внутренний цикл, строка 120).

```
10 N = 7
20 DIM AR(N, N)
30 beg = 1: fin = N: d = 1: el = 1
40 FOR J = 1 TO N
50 FOR I = beg TO fin STEP d
60 AR(I, J) = el
70 el = el + 1
80 NEXT I
90 SWAP beg, fin: d = -d
100 NEXT J
110 FOR J = N TO 1 STEP -1
120 FOR I = 1 TO N
130 PRINT AR(I, J);
140 NEXT I
150 PRINT
```

```
160 NEXT J
170 END
```

Задача 3.60.

В одномерный массив размера *N* (*N* задано: не менее 8 и не более 20) ввести натуральные числа. Создать новый массив, в который поместить только те элементы исходного массива, которые имеют с предыдущим элементом общий делитель, отличный от 1. Распечатать результирующий массив; если искомых элементов нет, вывести соответствующее сообщение.

Решение.

Количество элементов вводим с клавиатуры с проверкой (строки 10–20).

При объявлении количества элементов результирующего массива исходим из того, что в него войдут все элементы исходного, за исключением первого (см. ниже). Для формирования массива-результата потребуется независимый счетчик, обнулим его перед началом работы (строка 40).

Этапы решения: 1. Ввод в массив с клавиатуры натуральных чисел с проверкой (напомним, что натуральные числа — это целые числа больше 0). 2. Анализ элементов исходного массива с формированием массива-результата. 3. Вывод результата.

На первом этапе производится последовательный ввод элементов в цикле FOR. В аналогичном цикле идет последовательная обработка введенных элементов (этап 2), значит оба этапа можно объединить в общий цикл. Но есть одна тонкость: на первом этапе проходим все числа от первого до последнего, а во втором — начиная со второго (первое число предыдущего не имеет, поэтому из рассмотрения выпадает). Учтем это обстоятельство: введя первое число (`i=1`), минуя обработку, переходим к вводу очередного числа (строка 90).

Анализ производим во внутреннем цикле: начиная со второго элемента, ищем общий делитель для него и предыдущего элемента. Делители задает параметр внутреннего цикла. Поскольку делитель должен быть отличен от 1, внутренний цикл начинаем с `j=2`. Предельное значение делителя — текущий элемент массива: ведь один из элементов может оказаться единственным делителем другого, например, 25 и 5 (общий делитель 5).

Два элемента имеют общий делитель X, если остаток от деления каждого из них на X равен нулю. В этом случае на единицу увеличиваем счетчик элементов в результирующем массиве (`k=k+1`) и k-му элементу результирующего массива присваиваем значение текущего элемента исходного массива (`B(k)=A(i)`), после чего заканчиваем цикл поиска делителей (`j=A(i)`), чтобы несколько раз не записать текущий элемент в массив-результат (например, в случае, когда `A(i)=6`, а `A(i-1)=12`), и вводим очередное число в массив A. Все это можно записать в одном операторе IF (строка 110): напомним, что вся IF-конструкция в рассматриваемой версии Basic'а должна быть размещена на одной строке.

Вывод результата производится в зависимости от того, есть ли элементы в массиве B — используем условный оператор в сочетании с оператором условного перехода (строки 140–180).

```
10 INPUT "Количество элементов в массиве ="; N
20 IF N <> FIX(N) OR N < 8 OR N > 20 THEN 10
30 DIM A(N), B(N-1)
40 k = 0
50 FOR i = 1 TO N
60 PRINT "Элемент #"; i
70 INPUT A(i)
80 IF A(i) <> FIX(A(i)) OR A(i) < 1 THEN 60
90 IF i = 1 THEN 130
100 FOR j = 2 TO A(i)
110 IF A(i) MOD j = 0 AND A(i - 1) MOD j = 0 _ 'Перенос строки*
```

*Так переносить строки можно далеко не во всех версиях Basic'а.

```
    THEN k = k + 1: B(k) = A(i): j = A(i)
120 NEXT j
130 NEXT i: PRINT
140 IF k = 0 THEN PRINT "Искомых элементов нет": GOTO 190
150 PRINT "Результирующий массив:"
160 FOR i = 1 TO k
170 PRINT B(i);
180 NEXT i
190 END
```

Определение результата работы программы

Задача 3.61.

Представить результат, который программа выводит на экран.

```
10 N = 4: DIM A(N, N)
20 FOR i = 1 TO N
30 FOR j = 1 TO N
40 IF i + j > N + 1 THEN A(i, j) = N + j ELSE A(i, j) = N - i
50 NEXT j, i
60 FOR i = 1 TO N \ 2
70 FOR j = 1 TO N
80 SWAP A(i, j), A(N - i + 1, j)
90 NEXT j
100 PRINT A(i, N - i + 1);
110 NEXT i
120 FOR i = 1 TO N
130 PRINT
140 FOR j = 1 TO N - i
150 SWAP A(i, j), A(N - j + 1, N - i + 1)
160 PRINT A(j, i);
170 NEXT j, i
180 END
```

Решение.

Отметим, что `A(N)` — квадратная матрица.

Разобьем программу на этапы, определим, что в них происходит (разбивку следует осуществлять по циклам) и изобразим расположение элементов в матрице после каждого из этапов.

1. Строки 20–50. Заполнение матрицы. Значения элементов вычисляются в зависимости от условия.

2. Строки 60–110. Преобразование матрицы и распечатка некоторых ее элементов во внешнем цикле. Внешний цикл организован по строкам, причем только до середины матрицы (до ее горизонтальной оси).

После этапа 1				После этапа 2				После этапа 3				На экране
3	3	3	3	0	6	7	8а	3в	8е	8з	8	8 7
2	2	2	8	1г	1	7б	8	3	2ж	7	7	3 1 2
1	1	7	8	2д	2	2	8	3	2	1	6	8 2
0	6	7	8	3	3	3	3	3	2	1	0	8

Схема 26. Схема матрицы в результате преобразований

Во внутреннем цикле — проход всех элементов строки от начала до конца. В теле внутреннего цикла осуществляется перестановка элементов с одинаковым номером столбца (он задан во внешнем цикле), номера строк при этом расположены зеркально относительно горизонтальной оси. На основании перечисленных наблюдений делаем вывод: на этом этапе происходит поворот матрицы относительно горизонтальной оси.

Распечатка производится во внешнем цикле, значит распечатано будет два элемента: при `i=1` и `i=2`. Индексы выводимых элементов свидетельствуют о том, что выведены будут элементы побочной диагонали, начиная с правого верхнего. На первой итерации внешнего цикла будет выведено число 8 (отмечено буквой «а» на схеме 26), на второй — 7 («б» на схеме 26). Разделитель «точка с запятой» в строке 100 показывает, что распечатаются они в строку в уплотненном формате (через два пробела).

3. Строки 120–170. Преобразование матрицы и распечатка некоторых ее элементов во внутреннем цикле. Сразу отмечаем для себя «пустой» PRINT в строке 130 (первый оператор в теле внешнего цикла): значит каждая группа элементов с одинаковым индексом `i` будет выводиться с новой строки.

Внешний цикл 3-го этапа — по строкам, внутренний — по столбцам. Анализ заголовков циклов и индексы меняемых местами элементов указывают на то, что тут происходит поворот матрицы относительно побочной диагонали.

Во внутреннем цикле выводятся элементы, имеющие индексы A(j,i), т. е. по вертикали. И тут нужно быть очень внимательным: при выводе необходимо следить за тем, какое из значений на данном этапе располагается в указанной позиции — «новое», уже после перестановки, или еще прежнее. Выводимые элементы отмечены буквами в последовательности появления на экране.

Разделитель «точка с запятой» в строке 160 показывает, что в пределах одного индекса j элементы будут распечатываться в строку в уплотненном формате (через два пробела). Количество элементов в строке на экране (!) следует из заголовков циклов.

Задача 3.62.

Представить результат, который программа выводит на экран.

```
10 N = 4: P = 7
20 k = 0: S = 0
30 DIM A(N, N), B(N * N)
40 FOR x = 1 TO P
50 FOR y = x TO P + 1 - x
60 k = k + 1
70 B(k) = 2 * P - x - y
80 S = S + (y - x - 1)
90 NEXT y, x
100 PRINT S
110 FOR j = 1 TO N
120 FOR i = 1 TO N
130 A(i, j) = B(N * (i - 1) + j)
140 PRINT A(i, j);
150 NEXT i
160 PRINT
170 NEXT j
180 END
```

Решение.

Отметим, что A(N) — квадратная матрица 4 × 4, а B(N*N) — одномерный массив из 16 элементов.

Разобьем программу на этапы, определим их назначение и изобразим расположение элементов в массивах после каждого этапа.

Таблица 16. Значения параметров и переменных в цикле

x	y (от...до)	k (от...до)	B(k)	B(k) (от...до)	y-x-1	y-x-1 (от...до)	Слагаемые
1	1...7	1...7	13-y	12...6	y-2	−1...5	−1, 0, 1, 2, 3, 4, 5
2	2...6	8...12	12-y	10...6	y-3	−1...3	−1, 0, 1, 2, 3
3	3...5	13...15	11-y	8...6	y-4	−1...1	−1, 0, 1
4	4...4	16	10-y	6	y-5	−1	−1
5	5...3	—	—	—	—	—	—
6	6...2	—	—	—	—	—	—
7	7...1	—	—	—	—	—	—

Таблица 17. Массив B(N × N) после 2-го этапа

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B(i)	12	11	10	9	8	7	6	10	9	8	7	6	8	7	6	6

1. Строки 10–30. Заданы исходные значения переменных: N — определяет число элементов в массивах; S — значение, вычисляемое на следующем этапе; k — счетчик элементов для формирования массива B; P — переменная для формирования массива B.

2. Строки 40–90. Заполнение одномерного массива и вычисление S.

Счетчик k каждый раз увеличиваем на 1, значит идет последовательное заполнение массива B. Значение параметра внутреннего цикла (y) зависит от значения параметра внешнего цикла (x); в таблице 16 указаны диапазоны изменения y для каждого x.

Видно, что при значениях x с 5 по 7 операторы внутреннего цикла исполнены не будут, следовательно, заполнение и суммирование происходят в 4 стадии. Для каждой стадии определим формулы расчета элемента массива и очередного слагаемого (см. столбцы «B(k)» и «y-x-1» таблицы 16), а на их основе — диапазоны значений и сами значения. Получим значение переменной S, сложив слагаемые (S=18). Значения элементов массива занесены в таблицу 17.

Матрица $A(N, N)$ после 3-го этапа				На экране			
				18			
12а	8б	9в	8г	12	8	9	8
11д	7е	8ж	7з	11	7	8	7
10и	6к	7л	6м	10	6	7	6
9н	10о	6п	6р	9	10	6	6

Схема 27. Схема матрицы и выведенные значения

3. Строка 100. Распечатка суммы. После вывода значения на экран произойдет переход на следующую строку: завершающий символ — пробел.

4. Строки 110–170. Заполнение матрицы и распечатка всех ее элементов (см. схему 27).

Массив A заполняется элементами массива B. Последовательность элементов массива B — от меньшего индекса к большему, по порядку, как видно из формулы для расчета индекса: по такой схеме производится преобразование одномерного массива в двумерный. Внешний цикл — по столбцам, значит указанный порядок получим при чтении массива по вертикали. Между тем фактическое заполнение (последовательность указана на схеме 27 буквами) и распечатка происходят по горизонтали, как видно из соотношения индексов и параметров цикла.

Разделитель «точка с запятой» в строке 140 показывает, что в пределах одного индекса j элементы будут распечатываться в строку в уплотненном формате (через два пробела). «Пустой» PRINT в строке 160 (последний оператор в теле внешнего цикла) — перевод на новую строку экрана после распечатки очередной строки матрицы.

Использование данных символьного типа

Предварительные замечания

В реальных (неформализованных) задачах часто встречаются объекты символьного типа: названия городов и предприятий, наименования продукции, фамилии учеников и т. п. Символьная константа представляет собой цепочку допустимых символов языка, заключенную в кавычки. Например, в ее состав могут входить буквы латинского алфавита, кириллица, цифры, всевозможные знаки, скобки, пробел и др.

Напомним, что при идентификации таких объектов их тип указывается с помощью специального конечного символа — знака доллара ($). Например, идентификаторы объектов символьного типа: FIO$, NAZV$, ST$ и т. д.

Символьные данные, как и данные других типов, можно объединять в массивы, например:

```
20  DIM  F$(20)      ' F$ - массив фамилий абитуриентов
30  DIM  PROD$(K)    ' PROD$ - массив названий продукции
```

Символьные данные, подобно прочим, можно вводить с клавиатуры и выводить на экран, используя операторы INPUT и PRINT:

```
50  FOR  I = 1  TO  20
60  PRINT "Номер в списке:"; I,
70  INPUT  "Фамилия абитуриента: "; FAM$ (I)
80  NEXT I
```

При выводе на экран символьные данные распечатываются без кавычек. В тексте программы кавычки для символьных констант обязательны, если только они не находятся в списке оператора определения констант.

В Basic'е под каждый символ выделяется 1 байт (8 бит) памяти. Исходя из этого, формируются кодовые таблицы, содержащие 256 символов*. В зависимости от национального алфавита таблицы различаются, однако первые 128 символов (с 0 по 127) везде одинаковы: с 0 по 31 — управляющие символы, 32 — пробел, далее — знаки, 48–57 — цифры от 0 до 9, 65–90 — заглавные буквы латинского алфавита, 97–122 — малые буквы латинского алфавита. В промежутках — знаки.

Средства работы с данными символьного типа

Операции, позволяющие обрабатывать строки. Символьные данные можно сравнивать друг с другом. Для этого используют те же операции сравнения, что и при работе с данными арифметического типа. Сравнение выполняется посимвольно (побайтно) слева направо. Фактически, сравниваются коды, соответствующие символам. При сравнении строк разной длины недостающие символы в конце более короткого операнда считаются пробелами. Например:

```
20 IF "a" > "b" THEN PRINT "a>b "; ELSE PRINT "a<=b ";
30 IF "aa" > "b" THEN PRINT "aa>b "; ELSE PRINT "aa<=b ";
40 IF "ba" > "b" THEN PRINT "ba>b "; ELSE PRINT "ba<=b ";
50 IF "aa" > "a" THEN PRINT "aa>a "; ELSE PRINT "aa<=a ";
```

На экране при этом получим:

```
a<=b  aa<=b  ba>b  aa>a
```

Для обработки символьных данных можно использовать операцию сцепления операндов — конкатенацию (знак операции — символ «+»). Например:

```
10 PRINT "Видео" + "камера"    'На экране:    Видеокамера
20 PRINT "1" + "2" + "3"       'На экране:    123
30 PRINT " 1" + " 2" + " 3"    'На экране:     1 2 3
```

*В кодовой таблице каждому из символов приводится в соответствие некоторое число. Эти числа называются кодами.

Встроенные функции для работы с символьными строками. Для обработки символьных данных можно использовать специальные встроенные функции. Напомним, что всякая функция возвращает результат вполне определенного типа; если функция имеет аргументы, то эти аргументы тоже должны быть вполне определенного типа. Символ «$» после имени функции означает, что она возвращает значение символьного типа, иначе — арифметического типа.

В Basic'е имеется значительное количество функций для обработки символьных данных. Наиболее употребимые приведены в таблице 18.

Кроме того, при работе с символьными данными всегда можно преобразовать строку в одномерный массив отдельных символов, «расщепив» ее при помощи функции `MID$`. Работа с таким массивом практически ничем не будет отличаться от работы с обычным (числовым) массивом (см. раздел «Массивы»). После завершения работы символы следует слить в строку при помощи операции сцепления.

Типовые задачи на обработку символьных данных

Задачи, связанные с обработкой символьных данных, не слишком разнообразны. Как правило, они требуют проведения анализа и (или) преобразования строк. Иногда бывает необходимо сформировать массив из слов (лексем) строки. Решение подавляющего большинства задач требует владения встроенными функциями: без них не обойтись даже в случае преобразования строки в массив.

Таблица 18. Функции для работы с данными символьного типа

Функция	Выполняемые действия	Комментированный пример
1	2	3

LEN(x$)	Определяет длину указанной символьной переменной в байтах (количество символов).	`10 FIO$ = "Петров-Водкин"`
		`20 L = LEN(FIO$)`
		Дает результат: L = 13.

LEFT$(x$,k)	«Вырезает» из переменной x$ слева k символов. «Вырезанные» символы можно присвоить другой переменной.	`10 FIO$ = "Петров-Водкин"`
		`20 T$ = LEFT$(FIO$, 4)`
		Дает результат: T$ = "Петр"
		(FIO$ — остается без изменений)

RIGHT$(x$,k)	Аналогично предыдущей функции, только справа.	`10 FIO$ = "Петров-Водкин"`
		`20 P$ = RIGHT$(FIO$, 6)`
		Дает результат: P$ = "Водкин"
		(FIO$ — остается без изменений)

MID$(x$,pos,k)	«Вырезает» k символов из переменной x$ начиная с позиции pos, включительно.	`10 FIO$ = "Петров-Водкин"`
		`20 S$ = MID$(FIO$, 4, 3)`
		Дает результат: S$ = "ров"
		(FIO$ — остается без изменений)

VAL(x$)	Преобразует указанную символьную переменную x$ в десятичное число. Символьная переменная должна иметь правильное представление десятичного числа.	`20 P$ = "-243.65"`
		`30 Y = VAL(P$)`
		`40 PRINT Y + 100`
		На экране: -143.65

Продолжение таблицы 18

Функция	Выполняемые действия	Комментированный пример
1	2	3

STR$(y)	Преобразует число y в строку символов. С учетом знака. Если число положительное, вместо знака ставится пробел.	`20 a = 123.45`
		`30 b = -678`
		`40 c$ = STR$(b) + STR$(a)`
		Результат: c$ = "-678 123.45"

CHR$(nomer)	Результат — символ, соответствующий указанному номеру в таблице (списке) символов.	`20 FOR I = 128 TO 159`
		`30 PRINT CHR$(I): NEXT I`
		Это вывод заглавных русских букв по алфавиту

ASC(x$)	Определяет числовой код первого символа в строке x$.	`20 PRINT ASC("123");`
		`30 PRINT ASC(" 1a2m");`
		`40 PRINT ASC("ABC");`
		На экране: 49 32 65

INSTR(n,x$,y$)	Начиная с позиции n определяет позицию первого появления подстроки y$ в строке x$. Если не находит, то результат 0.	`20 x$="кукуруза": y$="ку"`
		`30 PRINT INSTR(1, x$, y$);`
		`40 PRINT INSTR(2, x$, y$);`
		На экране: 1 3

HEX$(y)
OCT$(y)
	Преобразует число y в строку 16-ричных (8-ричных) цифр. Не в число!!!	`20 y=255`
		`30 PRINT HEX$ (y);`
		`40 PRINT HEX$ (16)+ HEX$ (10);`
		На экране: FF 10A

Задача 3.63. Сколько слов содержат не более трех букв «z»?

Дана символьная строка длиной до 254 символов. В качестве разделителей слов в предложении используются пробелы. Вывести на экран сообщение: сколько слов в этом предложении содержат не более 3 букв «z».

Решение.

Просмотр строки осуществим в цикле с известным числом повторений: количество символов определяем при помощи функции LEN. Перед входом в цикл обнулим две переменные: k — количество слов предложения, содержащих не более 3 букв «z», и kolz — количество букв «z» в текущем слове. При помощи функции MID$ будем в цикле по одному выделять символы строки и анализировать их. При этом пробел является признаком окончания текущего слова (чтобы не пропало последнее слово, надо добавить к введенной строке дополнительный пробел — строка 20). Пока не дошли до пробела, считаем в текущем слове все обнаруженные в нем символы «z». Встретив пробел, подводим итог по текущему слову: проверяем, отвечает ли количество указанных символов поставленному условию. Если да, увеличиваем на 1 итоговый счетчик (k). Перед переходом к следующему слову нужно приготовить счетчик букв «z» — обнулить его; сделать это надо до выхода из цикла. Вывод результата производим после того как вышли из цикла, просмотрев все слова строки.

```
10 INPUT "Строка ="; x$
20 x$ = x$ + " "
30 kolz = 0: k = 0
40 FOR i = 1 TO LEN(x$)
50 c$ = MID$(x$, i, 1)
60 IF c$ = "z" THEN kolz = kolz + 1
80 IF c$ <> " " THEN 110
90 IF kolz <= 3 THEN k = k + 1
```

```
100 kolz = 0
110 NEXT i
120 PRINT "Не менее 3 'z' имеется в"; k; "словах"
130 END
```

Задача 3.64. Сколько цифр в строке?

Дана строка длиной до 254 символов. Посчитать, сколько в строке цифр.

Решение.

Вычленение символов из строки и определение их количества осуществляется так же, как в задаче 3.63.

Метод решения поставленной задачи основан на возможности сравнения строк и том соображении, что в кодовой таблице символы «родственного» типа следуют один за другим. Этот прием удобно использовать в тех случаях, когда надо проверить символ на принадлежность либо непринадлежность некоторой группе символов.

Аналогично можно определить количество символов, не являющихся цифрами, тех или иных букв и т. п.

```
10 INPUT "Строка ="; x$
20 kol = 0
30 FOR i = 1 TO LEN(x$)
40 c$ = MID$(x$, i, 1)
50 IF c$ >= "0" AND c$ <= "9" THEN kol = kol + 1
60 NEXT i
70 PRINT "Количество цифр в строке ="; kol
80 END
```

Задача 3.65. Вычислить среднее арифметическое цифр строки.

Дана строка длиной до 254 символов. Подсчитать среднее арифметическое входящих в него цифр.

Решение.

В основу решения положены приемы, описанные в задаче 3.64. Прокомментируем особенности данной задачи.

После того как мы убедились, что символ — цифра (проверка условия в строке 50), можно воспользоваться функцией VAL, которая возвращает десятичное число, соответствующее указанной в качестве аргумента строке, а именно — нашему символу. Это десятичное число складываем с текущим значением суммы и на 1 увеличиваем количество.

Просмотрев в цикле все символы строки, перейдем к выводу результата. Перед вычислением среднего арифметического надо проверить, были ли в строке цифры. Если не было, то среднее определить нельзя, что необходимо учесть при выводе.

```
10 INPUT "Строка ="; x$
20 k = 0: sum = 0
30 FOR i = 1 TO LEN(x$)
40 c$ = MID$(x$, i, 1)
50 IF c$ >= "0" AND c$ <= "9" THEN k = k + 1: sum = sum + VAL(c$)
60 NEXT i
70 IF k = 0 THEN PRINT "Цифр нет" ELSE PRINT "sred="; sum / k
80 END
```

Задача 3.66. Поиск одинаковых символов в строке.

Определить, есть ли в заданной строке одинаковые символы. По результатам проверки вывести на экран сообщение.

Решение.

Каждый символ следует сравнить со всеми предыдущими: для этого используем вложенные циклы FOR — во внешнем последовательно выделяем символы строки, во внутреннем перебираем всех его предшественников и сравниваем. Если обнаружилось значение, которое уже было, изменяем значение флажка FLAG и закрываем циклы.

Внешний цикл ограничиваем предпоследним символом. Это принципиально важно, поскольку при I=L произошел бы выход за пределы строки во внутреннем цикле.

Вывод результата производим в зависимости от значения флажка.

```
10 INPUT "Строка символов ="; F$
20 L = LEN(F$): FLAG = 0 'FLAG = 0 - нет одинаковых символов
30 FOR I = 1 TO L - 1
40 S1$ = MID$(F$, I, 1)
50 FOR J = I + 1 TO L
60 S2$ = MID$(F$, J, 1)
70 IF S1$ = S2$ THEN FLAG = 1: J = L + 1: I = L
80 NEXT J, I
90 PRINT "Одинаковые символы ";
100 IF FLAG = 0 THEN PRINT "отсутствуют." ELSE PRINT "есть."
110 END
```

Задача 3.67. Составление перечня имеющихся в строке гласных.

Составить перечень всех гласных латинских букв, имеющихся в строке, посчитать, сколько раз каждая из них встречается.

Решение.

Задачу можно решить несколькими способами. Рассмотрим один из них.

Определим символьную переменную `glas$`, присвоив ей значение строки, представляющей собой перечень латинских гласных букв. Введем с клавиатуры исходную строку. Следует также предусмотреть ситуацию, когда в строке гласных не окажется вовсе: в этом случае нужно выдать соответствующее сообщение. Для контроля наличия гласных введем флажок, который обнулим перед началом работы.

Решение построено на сравнении каждого из символов строки гласных букв с символами исходной строки (символы будем выделять из строк при помощи функции MID$). Потребуются вложенные циклы. Во внешнем будем перебирать гласные, выделяя их из строки `glas$`, и обнулять количество данной буквы в анализируемой последовательности символов (строка 50). Затем производится подсчет, сколько раз встретилась данная гласная. Для этого нужен внутренний

цикл, в котором осуществляется проход по строке и сравнение, по итогам которого наращивается (или нет) на 1 значение количества (строка 90). Просмотрев всю анализируемую строку, проверяем значение количества. Если оно отлично от нуля, распечатываем гласную букву и ее количество, а также присваиваем флажку значение 1: гласные в строке есть (флажок примет это значение при первой же гласной с k>0 и больше меняться не будет). На этом работа с текущей гласной заканчивается, осуществляется переход к следующей.

Просмотрев все гласные, проверяем значение флажка. Если он сохранил стартовое значение (0), гласных в строке нет, о чем и выдаем сообщение.

```
10 glas$ = "AaEeIiOoUuYy" 'Строка латинских гласных
20 f = 0
30 INPUT "Строка ="; x$
40 FOR i = 1 TO LEN(glas$)
50 k = 0
60 t$ = MID$(glas$, i, 1)
70 FOR j = 1 TO LEN(x$)
80 c$ = MID$(x$, j, 1)
90 IF t$ = c$ THEN k = k + 1
100 NEXT j
110 IF k > 0 THEN f = 1: PRINT t$, k
120 NEXT i
130 IF f = 0 THEN PRINT "Лат. гласные в строке отсутствуют."
140 END
```

Замечание. Если бы требовалось подсчитать, сколько раз встречается в строке каждая из гласных букв, задача была бы проще: не нужно было бы проводить проверку на наличие.

Задачи на изменение строк (замена, удаление, вставка символов и лексем). Если не преобразовывать строку в массив символов, а потом обратно, то любые изменения проще всего производить не в исходной строке, а создать дополнительную переменную символьного типа, куда и заносить нужные символы (результаты преобразований). Эту переменную перед началом работы

следует определить, присвоив ей значение "" (строка, не содержащая символов, пустая).

Если по условию создание дополнительной строки не предусмотрено, нужно «разрывать» строку на части с помощью функции MID$ и склеивать в преобразованном виде, используя операцию конкатенации (сцепления). Например, вставим в данную строку слово «не »:

```
10 x$ = "Это интересно."
20 x$ = MID(x$, 1, 4) + "не " + MID(x$, 5, LEN(x$))
30 PRINT x$            'На экране: Это не интересно.
```

В любом случае выбор того или иного способа решения определяется условием задачи.

Задача 3.68. Нормализация пробелов.

Ввести текст длиной до 254 символов в виде строки, состоящей из слов, разделенных пробелами (одним или более). Преобразовать строку так, чтобы между словами было ровно два пробела; вывести ее на экран. Выдать сообщение о количестве удаленных или добавившихся пробелов.

Решение.

Введем исходную строку `x$`. При вводе строки автоматически отсекаются ведущие и хвостовые пробелы, поэтому о них нам заботиться не надо. Будем вводить результат в дополнительную переменную `y$`, которую инициализируем в строке 20.

Просматриваем в цикле исходную строку посимвольно от начала и до окончания строки: не-пробелы просто переписываем в `y$` (строка 50). Из всех следующих подряд пробелов, независимо от их количества, выделяем только один, записывая вместо него в результирующую строку два (строка 60).

Выйдя из цикла, распечатаем результат преобразований. Форма вывода количества удаленных или добавленных символов зависит от того, какая из строк

длиннее: x$ или y$. Собственно количество берем по модулю (строка 100).

```
10 INPUT "Исходная строка ="; x$
20 y$ = ""
30 FOR i = 1 TO LEN(x$)
40 c$ = MID$(x$, i, 1)
50 IF c$ <> " " THEN y$ = y$ + c$
60 IF c$ = " " AND MID$(x$, i + 1, 1) <> " " THEN y$ = y$ + "  "
70 NEXT i
80 PRINT "Новая строка ="; y$
90 IF LEN(x$) > LEN(y$) THEN PRINT "Удал."; ELSE PRINT "Добавл.";
100 PRINT ABS(LEN(x$) - LEN(y$)); "символов."
110 END
```

Задача 3.69. Удаление символов из строки.

Дана строка длиной до 254 символов. Удалить все знаки «+» перед символами, не являющимися цифрами.

Решение.

Задача является развитием задачи 3.64.

Введем заготовку для результирующей строки и инициализируем ее пустой строкой; добавляем пробел к исходной строке — для работы с последним символом (строка 20).

В цикле выделяем поочередно пару символов строки — текущий и следующий (за исключением последнего: это тот пробел, который мы добавили к строке). Анализируем символы согласно условию (строка 50). Важный момент: по логике анализа скобки в условии необходимы. Если текущий символ подлежит удалению (см. условие), переходим к следующему символу, в противном случае помещаем его в результирующую строку.

При подсчете числа удаленных плюсов «искусственный» пробел вычитаем из общего количества удаленных символов.

```
10 INPUT "Исходная строка ="; x$
20 y$ = "": x$ = x$ + " "
30 FOR i = 1 TO LEN(x$) - 1
40 c$ = MID$(x$, i, 1): s1$ = MID$(x$, i + 1, 1)
```

```
50 IF c$ = "+" AND (ø1$ < "0" OR ø1$ > "9") THEN 70
60 y$ = y$ + c$
70 NEXT i
80 PRINT "Новая строка =: "; y$
90 PRINT "Удалено"; LEN(x$) - LEN(y$) - 1; "пробов."
100 END
```

Задача 3.70. Замена слов в строке.

В строке длиной до 254 символов заменить все слова «всегда» на «часто» и вывести результат.

Решение.

Введем с клавиатуры исходную строку (x$). Для удобства работы и повышения массовости алгоритма определим еще две символьные переменные: y$ — замещаемая подстрока, z$ — замещающая подстрока. Слова берем вместе с прилежащими к ним пробелами, иначе будет произведена замена не только отдельных слов, но и заданного набора символов в составе других слов (так «навсегда» превратится в «начасто»). Если подлежащее замене слово окажется в исходной строке первым или последним (с одним пробелом), оно не преобразуется. Чтобы этого не случилось, добавим программно по пробелу в начало и конец исходной строки.

Решение построено на использовании функции INSTR(i,x$,y$) — см. раздел «Встроенные функции для работы с символьными строками». Начиная с позиции i она определяет позицию первого появления подстроки y$ в строке x$; если подстроки в строке не находит, то возвращает результат 0. Определим позицию первого вхождения (n). Можно производить замену. Для этого разрываем исходную строку на 3 части: до искомой подстроки, подстрока, после подстроки. Первую и третью части вычленяем при помощи функции MID$, помещаем между ними замещающее слово, связываем их операцией конкатенации и присваиваем то, что получилось, переменной x$. Так как мы не знаем, сколько раз в строке встретится интересующая нас

подстрока, используем цикл WHILE. Условие выполнения цикла: -n <> 0, т.е. наличие в строке интересующих нас подстрок. Перед входом в цикл определим лишь позицию первого вхождения, — чтобы n было определено (здесь удобно было бы использовать и итерационный цикл с постусловием). На каждой итерации цикла происходит замена одного нужного слова.

Произведя все необходимые замещения, уберем из строки «искусственные» пробелы (строка 80) и выведем ее на экран.

```
10 INPUT "Исходная строка:"; x$
20 x$ = " " + x$ + " ": y$ = " всегда ": z$ = " часто "
30 n = INSTR(1, x$, y$)
40 WHILE n <> 0
50 x$ = MID$(x$, 1, n - 1) + z$ + MID$(x$, n + LEN(y$), LEN(x$))
60 n = INSTR(1, x$, y$)
70 WEND
80 x$ = MID$(x$, 2, LEN(x$) - 1)
90 PRINT "Преобразованная строка: "; x$
100 END
```

Задача 3.71. Преобразование строки в массив лексем.

Дана строка длиной до 254 символов, содержащая слова, разделенные пробелами. Преобразовать строку в массив слов (лексем).

Решение.

Для записи выделенных лексем потребуется массив. На сколько слов его объявлять? Исходим из максимально возможного, а именно: все лексемы состоят из одного символа, все разделители — в единственном экземпляре (строка 30).

Принцип посимвольного анализа исходной строки описан в задаче 3.63 — воспользуемся им. Перед началом формирования очередного слова наращиваем счетчик (это его индекс в массиве) и присваиваем элементу массива лексем стартовое значение " ", чтоб было к чему прибавлять очередные символы.

В результате имеем не k, а k-1 полноценных элементов в массиве лексем. Последний — всегда пробел.

предпоследний — всегда не-пробел, за счет этого и получается лишний, не имеющий наполнения элемент (нереализованная заготовка). Учтем это при выводе результата.

```
10 INPUT "Исходная строка ="; x$
20 IF x$ = "" then 10
30 DIM w$(LEN(x$) \ 2)
40 x$ = x$ + " ": k = 1: w$(k) = ""
50 FOR i = 1 TO LEN(x$)
60 c$ = MID$(x$, i, 1)
70 IF c$ <> " " THEN w$(k) = w$(k) + c$: GOTO 90
80 IF MID$(x$, i + 1, 1) <> " " THEN k = k + 1: w$(k) = ""
90 NEXT i
100 FOR i = 1 TO k - 1
110 PRINT "Лексема №"; i; ": "; w$(i)
120 NEXT i
130 END
```

Неформализованные задачи

В условиях рассмотренных ранее задач конкретно говорилось, какого типа объекты следует использовать в программе (простые переменные или массивы, символьные или арифметические данные) и что с ними надо сделать. Реальные задачи подобных подсказок, как правило, не содержат: выбор объектов и способов решения возлагается на программиста. Он должен уметь увидеть в объектах задачи объекты алгоритма — формализовать ее, а сам алгоритм «разложить» на составляющие — этапы, каждый из которых представляет собой типовой прием. Уяснив это для себя, можно переходить к оптимизации решения: объединить отдельные этапы, обыграть нюансы условия.

В настоящем разделе приведены трудно формализуемые задачи. Обычно чем сложнее задача, тем больше она имеет вариантов решения, поэтому, разбирая предложенные способы, следует иметь в виду, что это только примеры, демонстрирующие, каков может быть ход рассуждений.

Многие трудно формализуемые задачи требуют использования массивов, а также часто предполагают работу с несколькими массивами, характеризующими один объект (связанные массивы). Подобного рода задачи и собраны в данном разделе. Хотя говорить о типовых алгоритмах тут уже не приходится, можно выделить некоторые стандартные приемы, используемые при решении задач с теми или иными объектами.

Задачи с использованием массивов

Задача 3.72.

24 спортсмена (фамилии и рост ввести с клавиатуры) необходимо построить в колонну по 4 человека в ряд. В колонне спортсмены должны располагаться так, чтобы рост их увеличивался от ряда к ряду и от правого фланга к левому. Результат вывести в виде таблицы фамилий спортсменов; рядом с каждой фамилией в скобках указать рост.

Решение.

Для повышения массовости алгоритма заменим значения констант, соответствующих количеству спортсменов, числу рядов и людей в ряду переменными (`N`, `P`, `C`), значения которых определим в строке 10.

Этапы решения:

1. Строки 30–50. Ввод исходных одномерных массивов: `fam$(N)` — фамилия, `R(N)` — рост (это связанные массивы).

2. Строки 60–90. Сортировка массива `R` по убыванию значений элементов с параллельной перестановкой элементов массива фамилий. Перестановка элементов в обоих массивах при сортировке необходима: массивы связанные, и если отсортировать только `R`, связь будет утрачена. Сортировку можно производить любым известным способом.

3. Может возникнуть мысль преобразовать исходные массивы в двумерные. В этом нет необходимости.

Вполне достаточно построить спортсменов указанным образом на экране, пользуясь приемом преобразования одномерного массива в двумерный, а именно — соотношением индексов.

Строки 100—170. Вывод элементов одномерных массивов в форме таблицы. Чтобы не возникло сомнений, перед выводом таблицы укажем, что это голова колонны, а после — хвост. Фамилию и соответствующий рост выводим в уплотненном формате, данные по спортсменам распечатываем в зонном формате. После распечатки каждого ряда переходим на новую строку.

```
10 N = 24: P = 6: C = 4
20 DIM R(N), fam$(N)
30 FOR i = 1 TO N
40 INPUT "Фамилия и рост очередного спортсмена: "; fam$(i), R(i)
50 NEXT i
60 FOR i = 1 TO N - 1
70 FOR j = i + 1 TO N
80 IF R(j) < R(i) THEN SWAP R(i), R(j): SWAP fam$(i), fam$(j)
90 NEXT j, i
100 PRINT "Голова колонны"
110 FOR i = 1 TO P
120 FOR j = 1 TO C
130 PRINT fam$(C * (i - 1) + j); "("; R(C * (i - 1) + j); ")",
140 NEXT j
150 PRINT
160 NEXT i
170 PRINT "Хвост колонны"
180 END
```

Задача 3.73.

Известны годовые объемы добычи нефти каждой из N компаний за последние T лет, N и T заданы. Последний год — 2000. Для каждой компании найти максимальный период, в течение которого она непрерывно увеличивала объем добытой нефти, — определить с какого по какой год длился этот период. Учесть, что искомый период не может длиться менее года.

Решение.

Введем с клавиатуры с проверкой число компаний и лет наблюдения N и T (строки 10—20).

Названия компаний объединим в одномерный массив символьного типа. Объемы выпуска компаний по годам — арифметический двумерный массив. Объявим их (строка 30) и введем в них данные с клавиатуры с помощью вложенных циклов FOR: внешний — по компаниям, внутренний — по годам наблюдения (строки 40–100).

Обратимся к условию задачи: может случиться, что ни у одной из компаний не обнаружится ни одного периода прироста добычи. Значит нужно ввести переменную-флажок (F), которая будет фиксировать факт наличия периода нарастания хотя бы у одной компании. Ее следует обнулить перед началом анализа (строка 13) и присвоить ей значение 1, как только выявится период прироста у какой-либо из компаний.

Обработку данных также произведем во вложенных циклах FOR. Во внешнем будем перебирать поочередно компании.

Для каждой компании будем фиксировать периоды возрастаний (их может оказаться несколько или не быть вообще) и находить среди них наиболее продолжительный, значит необходимы две переменные: L — текущая длительность периода нарастания для данной компании и MAXL — максимальная длина периода увеличения для текущей компании. Им следует присвоить стартовые нулевые значения для каждой компании, следовательно, в теле внешнего цикла (строки 150–160). Далее просматриваем выпуски данной компании по годам во внутреннем цикле (строки 170–200), отслеживая периоды прироста: прирост имеет место, если выпуск текущего года превышает выпуск предыдущего (поэтому внутренний цикл начинаем с 2). Если условие в строке 180 истинно, присваиваем флажку значение 1 (период прироста хотя бы у одной компании есть) и на единицу увеличиваем значение переменной L; если условие не выполняется, текущий период при-

роста кончился, и надо обнулить переменную L — приготовить ее для следующего потенциального периода благоденствия.

В строке 190 производится сравнение длительности текущего периода нарастания с максимальным периодом нарастания для данной компании. Если текущий оказался больше, требуется переопределение максимума с фиксацией окончания данного периода на текущий (!!!) момент в переменной `fin`. Сделать это можно с учетом года или без (в зависимости от этого видоизменится вывод в строке 21): выберем первое.

Просмотрев для данной компании выпуск за все годы, выведем результат: но только в том случае, если у нее был хотя бы один период прироста. Чтобы это проверить, используем условный оператор — строка 210.

Завершив просмотр всех компаний, проверим состояние флажка: если он остался равен 0, то ни у одной компании не было ни одного периода прироста, о чем и выдадим сообщение (строка 230).

```
10 INPUT "Число компаний и период наблюдения:"; N, T
20 IF N < 1 OR T < 1 THEN 1
30 DIM NAZ$(N), V(N, T)
40 FOR i = 1 TO N
50 PRINT "Предприятие №"; i
60 INPUT "Название:"; NAZ$(i)
70 FOR j = 1 TO T
80 PRINT "Год:"; j
90 INPUT "Объем выпуска:"; V(i, j)
100 NEXT j, i
110 PRINT "Максимальные периоды увеличения выпуска"
120 PRINT "Компания", "Начало", "Конец"
130 F = 0
140 FOR i = 1 TO N
150 L = 0
160 MAXL = 0
170 FOR j = 2 TO T
180 IF V(i, j) > V(i, j - 1) THEN L = L + 1: F = 1 ELSE L = 0
190 IF L > MAXL THEN MAXL = L: fin = 2000 - T + j
200 NEXT j
210 IF MAXL > 0 THEN PRINT NAZ$(i), fin - MAXL, fin
220 NEXT i
230 IF F = 0 THEN PRINT "Таких компаний нет."
240 END
```

Замечание. В данной задаче ввод и обработку данных по каждой компании можно объединить в общем охватывающем цикле.

Задача 3.74.

Группа спелеологов планирует осмотреть N пещер. Маршрут начинается от пещеры K, туда же группа должна вернуться. Дважды посещать прочие пещеры не предполагается. Из еще не обследованных пещер в качестве следующей выбирается пещера, которая расположена ближе всего к той, где спелеологи находятся в данный момент. Расстояния между пещерами, количество пещер N и номер пещеры, где начался маршрут — K, известны. Требуется указать номера пещер в порядке посещения и общий путь.

Решение.

Задача может быть решена разными способами. Рассмотрим один из них.

Подготовительный этап.

Потребуются два массива: а) двумерный — `rst(N,N)` — массив расстояний между пещерами — это симметричная относительно главной диагонали квадратная матрица: расстояние от пещеры A до пещеры B такое же, как от B до A, расстояние от A до A равно нулю (главная диагональ); б) одномерный — `p(N)` — массив посещений. Этот массив нужен для фиксации факта посещения данной пещеры (его размерность равна числу пещер). Изначально этот массив обнуляем. При посещении пещеры соответствующему элементу массива будет присвоено значение 1 или число, указывающее, какой по счету посетили данную пещеру.

Строки 30–80. С использованием вложенных циклов FOR заполним квадратную матрицу. Вводим с клавиатуры только элементы, которые выше главной диагонали: матрица симметрична. Элементы под

диагональю вычислим: `rst(j,i)=rst(i,j)`. Главную диагональ пока не трогаем.

Строки 90–160. Во вложенных циклах FOR распечатаем массив расстояний. В строке 110 во внешнем цикле заполним нулями главную диагональ и — тоже нулями — массив посещений (строго говоря, в Basic'е этого можно было бы и не делать: при объявлении массивы инициализируются нулями, но не советуем привыкать к небрежному стилю программирования).

Строки 170–180. Ввод номера пещеры, откуда начинается маршрут. Проверка обязательна.

Строка 190. Подготовка к вычислениям: прежде всего обнулим переменную `sum`, куда будет записываться общий путь и элементу массива посещений с индексом `K` присвоим значение 1: тут уже побывали. Поскольку в пещеру `K` еще предстоит вернуться, переменную `K` использовать для записи текущей позиции нельзя; введем для этого дополнительную переменную — `ps`. Ее исходное значение равно `K`.

Этап вычислений. Строки 210–280.

Для обработки данных потребуется организовать вложенные циклы FOR. Внешний — по числу перемещений от пещеры к пещере минус один: последнее перемещение — в пещеру `K`, будем обрабатывать уже за пределами цикла. Во внутреннем цикле ищем ближайшую пещеру к данной (ее номер) и расстояние до нее (соответственно переменные `sled` и `min`), игнорируя нули и точки, где уже были (см. условие). Перед входом во внутренний цикл определяем исходное значение минимума `min=E+38` и его позицию `sled=0`. Позиция минимума и будет являться номером следующего пункта. Выйдя из внутреннего цикла, прибавляем к пройденному пути значение `min`, переопределяем значение переменной `ps`: на следующей итерации текущей будет пещера, найденная в качестве следующей

на данной итерации. Заносим эту пещеру в массив посещений.

Вывод результата.

Вывод будет состоять как бы из двух частей. Общий пройденный путь может быть выведен лишь после того, как все вычисления закончены, т. е. после выхода из циклов (строка 300). Номера осмотренных пещер в последовательности посещения удобно распечатывать в цикле, по мере поступления очередной текущей позиции (строка 270). При выводе следует учесть два момента: во-первых, в перечне посещенных пещер не отразится исходный (конечный) пункт. Распечатаем его отдельно, перед входом в цикл обработки (строка 200) и после него (строка 290). Во-вторых, расстояние от предпоследнего пункта до конечной точки не может быть вычислено в цикле, это не минимум. Но мы знаем номера обеих пещер, а значит, знаем расстояние между ними. Прибавим это расстояние к общей сумме при выводе (строка 300).

```
10 N = 6
20 DIM rst(N, N), p(N)
30 FOR i = 1 TO N - 1
40 FOR j = i + 1 TO N
50 PRINT "Расстояние между пунктами"; i; "и"; j;
60 INPUT rst(i, j)
70 rst(j, i) = rst(i, j)
80 NEXT j, i
90 CLS: PRINT "Массив расстояний между объектами"
100 FOR i = 1 TO N
110 rst(i, i) = 0: p(i) = 0
120 FOR j = 1 TO N
130 PRINT rst(i, j);
140 NEXT j
150 PRINT
160 NEXT i
170 INPUT "Номер начального пункта: "; K
180 IF K < 1 OR K > N OR K <> FIX(K) THEN 170
190 p(K) = 1: ps = K: sum = 0
200 PRINT "Маршрут: "; ps;
210 FOR i = 1 TO N - 1
220 min = N + 38: sled = 0
230 FOR j = 1 TO N
240 IF rst(ps, j) <= min AND rst(ps, j) <> 0 AND p(j) = 0 _
    THEN min = rst(ps, j): sled = j   'Весь IF одну строку!
250 NEXT j
```

```
260 sum = sum + min: ps = sled: p(ps) = i:
270 PRINT ps;
280 NEXT i
290 PRINT K
300 PRINT "Расстояние: "; sum + rst(K, ps)
310 END
```

Задача 3.75.

Известны очки, полученные каждым из *M* спортсменов-многоборцев в каждом из *N* видов соревнований. Для каждого спортсмена определить, в каких видах соревнований он получил результат не хуже прочих и каков этот результат. Фамилии спортсменов известны.

Решение.

Объявим все необходимое: `N` — количество видов в многоборье; `M` — количество спортсменов; `fam$(M)` — массив фамилий спортсменов; `ball(M,N)` — двумерный массив баллов каждого спортсмена по каждому виду; `max(N)` — массив максимальных баллов по каждому виду (строки 10–20).

Заполним массивы фамилий и баллов: внешний цикл — по спортсменам, внутренний — по видам (строки 30–80).

По каждому виду определим лучший результат и запишем в массив максимальных баллов (строки 90–130). Внешний цикл — по видам: в каждом виде просматриваем результаты каждого спортсмена.

Анализ результатов каждого спортсмена и распечатка итога (внешний цикл — по спортсменам) — строки 140–210. Поскольку этапы обработки и вывода объединяем, распечатаем фамилию текущего спортсмена. Может оказаться, что он не показал лучших результатов ни в одном из видов: это следует учесть — введем флаг для контроля наличия у многоборца хотя бы одной победы. Во внутреннем цикле будем просматривать его показатели по всем видам и сопоставлять

с массивом максимумов. Если показатель спортсмена равен максимуму, флагу присваиваем значение 1 (победа есть) и выводим сообщения о победе в этом виде. Проанализировав оценки текущего спортсмена по каждому виду, проверим флаг. Если он равен нулю (побед нет), следует соответствующее сообщение.

Замечание по строке 180. Напомним, что IF-конструкция в Basic'е должна размещаться на одной программной строке.

```
10 N = 4: M = 5
20 DIM fam$(M), ball(M, N), max(N)
30 FOR i = 1 TO M
40 INPUT "Очередной спортсмен"; fam$(i)
50 FOR j = 1 TO N
60 PRINT "Оценка по виду №"; j;
70 INPUT ball(i, j)
80 NEXT j, i
90 FOR j = 1 TO N
100 max(j) = ball(1, j)
110 FOR i = 2 TO M
120 IF max(j) < ball(i, j) THEN max(j) = ball(i, j)
130 NEXT i, j
140 FOR i = 1 TO M
150 PRINT fam$(i)
160 flag = 0
170 FOR j = 1 TO N
180 IF ball(i, j) = max(j) THEN flag = 1: _
    PRINT "Лидер в виде №"; j; "Результат"; ball(i, j)
190 NEXT j
200 IF flag = 0 THEN PRINT "Ни в одном виде лидером не является."
210 NEXT i
220 END
```

Задачи по обработке данных с заданным или формируемым списком

Согласно условию подобных задач, представители некоторой группы населения выбирают один или несколько вариантов из предложенного списка либо просто отвечают на поставленный вопрос, а потом на основе их ответов формируется список. Встречаются задачи, включающие и то, и другое. Списки, как правило, подразумевают работу со связанными массивами: например, один из массивов может содержать

фамилии политических деятелей, а другой — количество голосов, набранных ими в ходе опроса. Напомним, что перестановки в одном из таких массивов требуют перемещений соответствующих элементов в остальных, иначе хранимая в них информация лишится смысла.

Алгоритмы задач со списками обычно требуют выполнения ряда стандартных операций. В зависимости от того, задан список или его предстоит формировать, эти операции будут различны. Рассмотрим наиболее типичные приемы.

1. Опрашиваемым предложено дать один или несколько ответов (выбрать или назвать). Если ответ один, используется простая переменная, в которую по очереди вводят мнения всех по очереди отвечающих. Если ответов несколько, следует объявить массив по числу ответов каждого (он тоже будет многоразового использования). Это удобно, поскольку по условию почти всегда нужно следить за тем, чтобы не было двух одинаковых ответов. Пример подобной проверки для некоторого опрашиваемого:

```
100 FOR j = 1 TO B 'B - количество ответов
110 PRINT "Ответ №"; j
120 INPUT v$(j)
130 FOR t = 1 TO j - 1
140 IF v$(j) = v$(t) THEN PRINT "Уже было": GOTO 110
150 NEXT t
.....................
220 NEXT j
```

Эта проверка часто проводится в задачах со списками обоих типов.

Создание двумерных массивов ($N \times B$, где N — число опрошенных) оправдано лишь в том случае, если предполагается неоднократное использование данных опроса.

2. При решении задач с готовым списком следует проверить, принадлежит ли ответ списку. Рассмотрим пример. Пусть `spis$(M)` — предлагаемый список из M

названий (фамилий и т. п.), а rey(M) — связанный с ним массив, где фиксируется количество отданных голосов. В цикле просматриваем элементы списка. Найдя совпадение, наращиваем рейтинг и переходим к следующему ответу. Если ответ в списке не значится, потребуется повторный ввод данного ответа.

```
100 FOR j = 1 TO B 'B - количество ответов
110 PRINT "Ответ №"; j
120 INPUT v$(j)
.....................
190 FOR y = 1 TO M 'M - количество названий в готовом списке
200 IF v$(j) = spis$(y) THEN rey(y) = rey(y) + 1: GOTO 230
210 NEXT y
220 PRINT "Нет в списке": GOTO 110
230 NEXT j
```

3. При формировании списка нужно обеспечить два момента. Во-первых, следует объявить массив (или несколько), куда будут заноситься данные будущего списка. Количество элементов при этом указывают максимально возможное (по логике задачи). Во-вторых, определяют переменную-счетчик (до начала работы k=0). Значение этой переменной будет увеличиваться по мере пополнения списка. Конечное ее значение — реальное число элементов в полученном списке. Рассмотрим пример. Пусть form$(N) — формируемый список названий (фамилий), rey(N) — связанный с ним массив, где фиксируется количество отданных голосов, v$ — простая переменная (случай, когда дается лишь один ответ). В цикле просматриваем уже оказавшиеся в списке наименования, сравнивая с введенным ответом. В случае совпадения наращиваем рейтинг и переходим к следующему респонденту. Если предложенный ответ в списке не найден, увеличиваем на 1 реальное количество элементов в списке, заносим в список ответ и определяем рейтинг данного наименования (пока 1).

```
70 k = 0
80 FOR i = 1 TO N 'N - число респондентов
```

```
90 PRINT "Респондент №"; i
100 INPUT v$
.....................
160 FOR x = 1 TO k 'k - число элементов в списке на данный момент
170 IF v$ = form$(x) THEN rey(x) = rey(x) + 1: GOTO 220
180 NEXT x
190 k = k + 1
200 form$(k) = v$
210 rey(k) = 1
220 NEXT i
```

Когда список сформирован, с ним работают точно так же, как с обычными массивами, но при этом нельзя забывать о связанных массивах.

В предлагаемых решениях часто используется оператор безусловного перехода. Это вызвано, главным образом, требованием уместить всю IF-конструкцию на одной программной строке (желательно, чтобы она при этом умещалась на экране). Возможны другие варианты решения, где используются, в частности, переменные-флажки. Советуем попробовать разработать их самостоятельно.

Задача 3.76. Выбор объекта из списка.

Дан список 10 политических деятелей. Опросили *N* журналистов: кто, по их мнению, может победить на выборах? Каждый из опрошенных назвал не менее одной и не более трех разных фамилий из предложенного списка (ввести с клавиатуры фамилии). Выдать пронумерованный список деятелей, которых никто не назвал.

Решение.

Это задача с выбором объекта из имеющегося списка объектов. Объявим необходимые объекты: `N` — количество опрошенных, `M` — кандидатов в списке и `K` — вариантов ответа; `fam$(M)` и `g(M)` — массивы фамилий и набранных ими очков для кандидатов (связанные); `kand$(K)` — массив ответов. Создавать двумерный массив ответов журналистов смысла не имеет: ввод их мнений и обработку будем проводить

параллельно, организовав цикл FOR по числу журналистов. Это будет внешний цикл.

Строки 30–60. Ввод фамилий кандидатов в массив `fam$(M)` и обнуление массива очков, ими набранных `g(M)`.

Внешний цикл — строки 70–200 — по опрашиваемым журналистам (`i`). Внутри него цикл по фамилиям, названным данным журналистом (`j`) — строки 90–200. Каждому журналисту дается три попытки (массив `kand$` из 3 элементов), из которых хотя бы одна должна принадлежать списку кандидатов, причем фамилии последних в массиве повторяться не должны.

После ввода очередного объекта (фамилии кандидата) проверим его на повторяемость — сравним с фамилиями, уже названными этим журналистом — вложение цикла (`t`) — строки 110–130. Это проверка № 1.

Проверка № 2: опрошенный должен назвать хотя бы одну фамилию из списка. Эту проверку можно выполнить разными способами. Будем сравнивать введенную фамилию поочередно с каждой фамилией списка — вложение цикла с параметром `z` — строки 150–170. Если фамилия в списке есть, наращиваем рейтинг данного кандидата. Если нет, рассматриваем два варианта: если это первый из ответов — повтор ввода (строка 180). Если хотя бы один из нормальных ответов есть, допрос данного журналиста прекращаем (строка 190). Для этого понадобится флаг, который обнулим перед входом в цикл с параметром `z` (строка 140). Флагу присваиваем значение 1, когда появляется хотя бы один нормальный ответ (строка 160).

Вывод неназванных кандидатов (строки 210–250). Их перечень по условию требуется пронумеровать. Для этого заводим независимый счетчик (`j`): обнулим его перед началом просмотра рейтингов кандидатов и будем увеличивать его значение на 1 при нахождении не

упомянутого кандидата. Возможно, в перечне никого не окажется, каждый из кандидатов кем-то будет упомянут. Для этого проанализируем значение счетчика j, выйдя из цикла просмотра массива рейтингов. Если рейтинг равен нулю, выдаем соответствующее сообщение.

```
10 N = 20: M = 10: K = 3
20 DIM fam$(M), g(M), kand$(K)
30 FOR i = 1 TO M
40 PRINT "Фамилия кандидата №"; i; : INPUT fam$(i)
50 g(i) = 0  'См. задачу 73.
60 NEXT i
70 FOR i = 1 TO N
80 PRINT "Журналист №"; i
90 FOR j = 1 TO K
100 INPUT "Выберите кандидата из списка"; kand$(j)
110 FOR t = 1 TO j - 1
120 IF kand$(j) = kand$(t) THEN PRINT "Уже был": GOTO 100
130 NEXT t
140 flag = 0
150 FOR z = 1 TO M
160 IF kand$(j) = fam$(z) THEN g(z) = g(z) + 1: flag = 1
170 NEXT z
180 IF flag = 0 AND j = 1 THEN PRINT "Нет в списке": GOTO 100
190 IF flag = 0 AND j > 1 THEN j = K
200 NEXT j, i
210 j = 1
220 FOR i = 1 TO M
230 IF g(i) = 0 THEN j = j + 1: PRINT j; "."; fam$(i)
240 NEXT i
250 IF j = 0 THEN PRINT "Все кандидаты названы"
260 END
```

Задача 3.77.

Среди *N* учащихся старших классов провели опрос (*N* задано): каждый выбрал из предложенного списка три разных университета страны, представляющих для него интерес (список пронумерован и содержит 8 университетов). Определить количество учащихся, чьи предпочтения совпали (с учетом последовательности перечисления) и номера выбранных ими университетов.

Решение.

Эта задача имеет очень много вариантов решений. Предлагаемый способ основан на том, что университеты пронумерованы от 1 до 8. Для удобства проверки вводимых чисел будем заносить их в одномерный массив `v(B)`. Каждое число, вводимое очередным учащимся, проверяется (должно укладываться в указанный диапазон и не должно быть названо дважды), после чего оно входит в качестве цифры в трехзначное число (строка 120): так выбор университетов 8, 4 и 7 даст число 847. Заносим трехзначное число в массив предпочтений `vib(N)` — оно однозначно отражает мнение данного опрошенного. Если бы порядок перечисления роли не играл, потребовалось бы упорядоченье названных однозначных чисел перед формированием элемента массива `vib(N)`.

Отсортируем массив предпочтений `vib(N)`. Теперь одинаковые числа следуют подряд. Воспользуемся этим: будем просматривать элементы массива, сравнивая текущее число со следующим. Несовпадение означает переход к следующему значению: можно подвести итог по предыдущему (распечатать), но только в том случае, если количество больше единицы: нас интересуют лишь повторы. В случае совпадения наращиваем количество данных значений в массиве, в противном случае сбрасываем счетчик: `kol=1`. Поскольку в цикле текущий элемент сравниваем со следующим, последний элемент обрабатывается отдельно, уже за пределами цикла. Нужно учесть, что повторов может не оказаться: для этого введем флажок `f`.

```
10 N = 200: M = 8: B = 3
20 DIM vib(N), v(B)
30 FOR i = 1 TO N
40 PRINT "Посетитель №"; i
50 vib(i) = 0  'См. задачу 73.
60 FOR j = 1 TO B
70 INPUT "Номер университета"; v(j):
80 IF v(j) < 1 OR v(j) > M THEN 70
90 FOR z = 1 TO j - 1
```

```
100 IF v(j) = v(z) THEN 70
110 NEXT z
120 vib(i) = vib(i) * 10 + v(j)
130 NEXT j, i
140 FOR i = 1 TO N - 1
150 FOR j = i + 1 TO N
160 IF vib(i) < vib(j) THEN SWAP vib(i), vib(j)
170 NEXT j, i
180 kol = 1: f = 0
190 FOR i = 1 TO N - 1
200 IF vib(i) <> vib(i + 1) AND kol > 1 THEN PRINT vib(i); kol,
210 IF vib(i) = vib(i + 1) THEN kol = kol + 1: f = 1 ELSE kol = 1
220 NEXT i
230 IF kol > 1 THEN PRINT vib(N); kol
240 IF f = 0 THEN PRINT "Совпадений нет"
250 END
```

Замечание. Данное решение придется скорректировать, если университетов в списке будет более 10. Можно, в частности, перейти от цифр к буквам и работать с символьными данными, которые также поддаются сортировке.

Задача 3.78. Формирование списка.

Каждая из *N* ферм представила заказ на приобретение *M* машин (все машины в перечне разные). Составить общий перечень необходимых машин с указанием потребности в них, упорядочить список по убыванию потребности.

Решение.

Задача с формированием списка. Формируя список, исходим из максимально возможного числа элементов в нем. Объявим `N` — количество ферм, `M` — число заказов для всех ферм; массивы: `car$(N*M)` — список названий машин, `g(N*M)` — количество набранных ими очков (это связанные массивы) и `fcar(M)` — массив заказов (обновляемый для каждой фермы, один на всех). Количество элементов в массивах `car$(N*M)` и `g(N*M)` закладываем по максимуму: все фермы заказали совершенно разные машины.

Потребуется счетчик для формирования списка и фиксации реального числа машин, которое будет в списке, так как до начала работы список пуст, `k=0` (строка 30).

Строки 40–190 — формирование массивов нужных машин и рейтингов каждой их них (внешний цикл, по результатам опроса каждой из ферм). Ферма называет `M` машин, которые заносятся в массив заказов: это осуществляется во вложенном цикле с параметром `j` — строки 60–180. Мы следим, чтобы каждая машина была названа только один раз, сверяя свежевведенное название со всеми, которые уже занесены в массив заказов. Для этого организуем еще один вложенный цикл с параметром `t` — строки 90–110. Если обнаружен дубль — повтор ввода. Теперь приступаем к работе с общим списком. Во внутреннем цикле с параметром `x` — строки 120–140 — проводится сопоставление текущего элемента массива заказов `i`-й фермы, `fcar(j)`, с элементами общего списка, которые там уже есть на данный момент (их там `k` наименований). Если нашли совпадение, наращиваем рейтинг `x`-й марки и переходим к вводу следующей машины для данной фермы. Строки 150–170 работают лишь при условии, что названной марки в общем списке не обнаружено. В этом случае приращиваем количество элементов в общем списке (`k=k+1`) и вносим наименование в список. Рейтинг этой машины `g(k)=1`.

Отсортируем массив рейтингов по убыванию, попутно переставляя соответствующие названия машин (строки 200–230), и выведем результат (строки 240–260).

Замечание. Массив заказов нужен только для проверок, иначе следовало бы создавать двумерный массив (заказы, фермы).

```
10 N = 4: M = 5
20 DIM car$(N * M), g(N * M), fcar(M)
```

```
30 k = 0
40 FOR i = 1 TO N
50 PRINT "Ферма №"; i
60 FOR j = 1 TO M
70 PRINT "Машина №"; j;
80 INPUT fcar$(j)
90 FOR t = 1 TO j - 1
100 IF fcar$(j) = fcar$(t) THEN PRINT "Уже была": GOTO 70
110 NEXT t
120 FOR x = 1 TO k
130 IF fcar$(j) = car$(x) THEN g(x) = g(x) + 1: GOTO 180
140 NEXT x
150 k = k + 1
160 g(k) = 1
170 car$(k) = fcar$(j)
180 NEXT j
190 NEXT i
200 FOR i = 1 TO k - 1
210 FOR j = i + 1 TO k
220 IF g(i) < g(j) THEN SWAP g(i), g(j): SWAP car$(i), car$(j)
230 NEXT j, i
240 FOR i = 1 TO k
250 PRINT car$(i), g(i)
260 NEXT i
270 END
```

Задача 3.79.

В ряде городов провели опрос некоторого процента жителей на предмет доверия, недоверия или безразличия к правительству. Всего опросили *N* человек. Результаты опроса по городам не упорядочены. Напечатать список городов, где, согласно опросу, большая часть населения поддерживает правительство. При выводе указать также количество опрошенных из данного города и процент тех, кто доверяет правительству.

Решение.

Задача с формированием списка. Формируя список городов, исходим из максимально возможного числа элементов в нем (все опрошенные оказались из разных городов).

Объявим количество опрошенных `N` и массивы по городам (`G$(N)` — города, `kol(N)` — количества опрошенных и `itog(N)` — итоговое мнение жителей каждого города). Все три массива — связаны между собой. Поскольку по условию нас интересуют города,

где большая часть населения поддерживает правительство, будем исходить из принципа: «Кто не с нами, тот против нас», т. е. население делим на доверяющих (+1) и всех остальных (–1). Итог получим, суммируя все вводимые ответы. Вариант: если «все остальные» будут вводить не –1, а 0, то получим количество сторонников.

Список формируем по стандартной схеме — внешний цикл по числу опрошенных. Потребуются две переменные — для записи города (gorod$) и мнения данного опрошенного (otv). При вводе проверяем, чтобы мнение было +1 или –1 (скобки в строке 60 нужны!) и введена не пустая строка вместо названия города. Потом, как в предыдущей задаче, проверяем, есть ли уже этот город в списке или нет. В зависимости от этого работают разные группы операторов: строка 80 либо строки 100–110.

При выводе помним о том, что нас интересуют города, где доверяет большинство, а таких может не оказаться. Для контроля наличия/отсутствия нужных городов используем флажок. Перебираем в цикле все оказавшиеся в списке города и для каждого проверяем итог. Распечатываем только при положительном итоге.

```
10 N = 5
20 DIM G$(N), kol(N), itog(N)
30 k = 0
40 FOR i = 1 TO N
50 INPUT "Город? Довер/нет/безразл (1/-1/-1)"; gorod$, otv
60 IF gorod$ = "" OR (otv <> 1 AND otv <> -1) THEN 50
70 FOR j = 1 TO k
80 IF G$(j) = gorod$ _
   THEN kol(j) = kol(j) + 1: itog(j) = itog(j) + otv: GOTO 120
90 NEXT j
100 k = k + 1
110 G$(k) = gorod$: kol(k) = 1: itog(k) = otv
120 NEXT i
130 PRINT "Большинство доверяет правительству в городах:": f = 0
140 FOR i = 1 TO k
150 IF itog(i) > 0 _
   THEN PRINT G$(i), kol(i), itog(i) * 100 / kol: f = 1
160 NEXT i
```

```
170 IF f = 0 THEN PRINT "Таких городов нет"
180 END
```

Строго говоря, в условии упомянуто и третье мнение, хотя эта информация нигде далее не используется. Приведем фрагмент программы, где учтены три варианта ответа, а также запоминаются города и ответы опрошенных в связанных массивах `gorod$(N)` и `otv(N)`. В данном случае для каждого города будем фиксировать в связанных массивах название — `G$(N)`, число опрошенных — `kol(N)` и количество тех, кто доверяет правительству — `yes(N)`.

```
10 N = 10
20 DIM gorod$(N), otv(N)
30 DIM G$(N), kol(N), yes(N)
40 k = 0
50 FOR i = 1 TO N
60 INPUT "Город? Степень доверия (1/-1/0)"; gorod$(i), otv(i)
70 IF gorod$(i) = "" OR otv(1) > 1 OR otv(i) < -1 THEN 60
80 FOR j = 1 TO k
90 IF G$(j) = gorod$(i) AND otv(i) = 1 _
    THEN kol(j) = kol(j) + 1: yes(j) = yes(j) + 1: GOTO 150
100 IF G$(j) = gorod$(i) AND otv(i) < 1 _
    THEN kol(j) = kol(j) + 1: GOTO 150
110 NEXT j
120 k = k + 1
130 G$(k) = gorod$(i): kol(k) = 1
140 IF otv(i) = 1 THEN yes(k) = 1 ELSE yes(k) = 0
150 NEXT i
160 PRINT "Большинство доверяет правительству в городах:": f = 0
170 FOR i = 1 TO k
180 IF yes(i) > kol(i) - yes(i) _
    THEN PRINT G$(i), kol(i), yes(i): f = 1
190 NEXT i
200 IF f = 0 THEN PRINT "Таких городов нет"
210 END
```

Задача 3.80. Выбор из списка и формирование списка. Клуб собаководства на ежегодной выставке проводит опрос общественного мнения. Каждому из N посетителей предлагается назвать три породы из прошлогоднего списка (в списке M пород расположены в порядке убывания популярности). Сформировать новый список в порядке убывания популярности породы,

указав место каждой породы в прошлогоднем списке. Предусмотреть проверку: каждый из опрошенных должен назвать три разные породы и только из предложенного списка.

Решение.

Объявим:

`pr$(M)` — список пород прошлого года;

`nov$(M)`, `rey(M)`, `was(M)` — связанные массивы: список пород этого года (разупорядоченная выборка из прошлогоднего массива), количество голосов, отданных соответствующим породам, и места соответствующих пород в прошлогоднем списке (индексы в массиве `pr$(M)`); количество элементов в этих массивах указываем максимально возможное — в нынешний список вошли все породы из прошлогоднего;

`v$(B)` — массив пород, называемых каждым опрошенным (массив многоразового использования);

`k` — реальное количество пород в новом списке; изменяется по мере пополнения списка.

Для удобства отладки программы исходный (прошлогодний) список введем с помощью операторов определения и чтения констант (DATA, READ, см. раздел «Операторы определения и чтения констант»).

Эта задача как с выбором из имеющегося списка, так и с формированием нового. Значит нужно провести следующие проверки:

1. Три вводимые породы должны быть разными — строки 130–150.

2. Введенная порода уже может быть занесена в новый список: если она там, нужно только увеличить на 1 количество отданных за нее голосов и перейти к следующей породе (этого же опрошенного или уже следующего) — строки 160–180;

3. Если при второй проверке породы в новом списке не нашли, ее туда надо поместить, но только при том условии, что она имеется в прошлогоднем спис-

ке. Для этого просматриваем прошлогодний список, сопоставляя его элементы с введенной породой: если соответствие найдено, на единицу увеличиваем количество в списке этого года, заносим название породы в массив названий, определяем для этой породы рейтинг на данный момент (`rey(k)=1`: пока всего один голос), а также фиксируем прошлогоднюю популярность данной породы — это индекс в прошлогоднем списке. После этого нужно перейти к следующей породе. Эта проверка выполняется в строках 190–210. Если названная порода не обнаружена в прошлогоднем списке, просим повторить ввод — строка 220.

Проверки должны осуществляться именно в такой последовательности.

После формирования списка нынешнего года (три массива по k элементов в каждом) произведем сортировку массива рейтингов, попутно переставляя и соответствующие элементы двух других массивов.

Вывод результата — стандартный.

```
10 N = 200: M = 10: B = 3
20 DIM pr$(M), nov$(N), rey(M), was(M), v$(B)
30 DATA "Дог", "Доберман", "Боксер", "Бульдог", "Овчарка"
40 DATA "Ротвейлер", "Бассет", "Чау-чау", "Спаниель", "Такса"
50 FOR i = 1 TO M
60 READ pr$(i): PRINT i; pr$(i),
70 NEXT i
80 k = 0
90 FOR i = 1 TO N
100 PRINT "Посетитель №"; i
110 FOR j = 1 TO B
120 INPUT "Порода"; v$(j)
130 FOR t = 1 TO j - 1
140 IF v$(j) = v$(t) THEN PRINT "Уже было": GOTO 120
150 NEXT t
160 FOR x = 1 TO k
170 IF v$(j) = nov$(x) THEN rey(x) = rey(x) + 1: GOTO 230
180 NEXT x
190 FOR y = 1 TO M
200 IF v$(j) = pr$(y) THEN k = k + 1: _
    nov$(k) = v$(j): rey(k) = 1: was(k) = y: GOTO 230
210 NEXT y
220 PRINT "Нет в списке": GOTO 120
230 NEXT j, i
240 FOR i = 1 TO k - 1
250 FOR j = i + 1 TO k
```

```
260 IF rey(i) < rey(j) THEN SWAP rey(i), rey(j): _
    SWAP nov$(i), nov$(j): SWAP was(i), was(j)
270 NEXT j, i
280 PRINT "Порода", "Популярность", "Номер в старом списке"
290 FOR i = 1 TO k
300 PRINT nov$(i), rey(i), was(i)
310 NEXT i
320 END
```

Требования к оформлению текста программы

1. Все объекты программы должны быть описаны в начале программы с помощью операторов REM или знака комментария ' (апостроф).

Объектами программы являются:

— исходные данные;

— промежуточные результаты;

— вспомогательные переменные;

— окончательные результаты.

2. Все исходные данные должны быть введены в Оперативную Память машины одним из операторов ввода информации. Выбор конкретного оператора ввода определяется автором программы, исходя из целесообразности использования того или иного оператора в каждом конкретном случае.

3. Каждому оператору ввода исходных данных с клавиатуры должен предшествовать пояснительный текст, выводимый при выполнении программы на экран монитора.

4. Вывод результатов решения задачи должен сопровождаться текстом, поясняющим все выводимые данные.

5. В тексте программы не должно быть синтаксических ошибок, таких как:

— неправильная конструкция оператора;

— неверная запись ключевых слов языка программирования;

— нарушение правил назначения имен объектам алгоритма;

— использование символов, которых нет на клавиатуре ЭВМ.

6. Составленная программа должна однозначно соответствовать условию задачи и быть работоспособна.

7. Работоспособность программы должна быть доказана путем формального исполнения алгоритма.

Задачи для самостоятельного решения

1. Треугольник задан двумя сторонами и углом между ними (в градусах). Определить длину третьей стороны треугольника и его медианы.

2. Известны ребра прямоугольного параллелепипеда. Вычислить длину диагонали, объем и полную поверхность параллелепипеда.

3. Определить, принадлежит ли точка с координатами X, Y ромбу с вершинами в точках $(0, 1)$, $(1, 0)$, $(0, -1)$, $(-1, 0)$.

4. С клавиатуры вводятся координаты точки на плоскости (X и Y). Принадлежит ли точка фигуре, ограниченной линиями: а) $Y = |X|$ и $Y = X/3 + 2$; б) $X = 2 * Y$, $X + Y = 4$ и $Y = 3 * X$; в) $X + Y = 5$ и $Y = X^2$?

5. Определить наибольшее из трех чисел.

6. Найти два меньших числа из трех.

7. Даны действительные положительные числа a, b, c, x, y. Выяснить, пройдет ли кирпич с ребрами a, b и c в прямоугольное отверстие со сторонами x и y.

8. Определить, високосный или нет год, номер которого вводится с клавиатуры. Примечание: високосный год делится без остатка на 4, но не заканчивается на два нуля, либо делится без остатка на 400.

9. Даны 3 прямоугольника. Прямоугольники ограничены следующими прямыми: 1-й $x = 0$; $x = 3$; $y = 0$; $y = 3$; 2-й $x = 1$; $x = 4$; $y = 1$; $y = 4$; 3-й $x = 2$; $x = 5$; $y = 2$; $y = 5$. С клавиатуры вводятся координаты некоторой точки: *xt* и *yt*. Вывести сообщение о том, принадлежит или не принадлежит точка: а) одновременно всем трем квадратам; б) ни одному из трех квадратов; в) одновременно первому и второму квадратам, не принадлежа при этом третьему; г) либо второму, либо третьему квадрату.

10. С клавиатуры вводится натуральное трехзначное число *N*. Определить, кратно ли оно 19. Предусмотреть проверку вводимых значений.

11. Юрист принимает по понедельникам и четвергам с 9 до 14, по вторникам с 14 до 19, по пятницам с 15 до 20 часов. В прочие дни приема нет. Организовать вывод названия дня недели и приемных часов при вводе номера дня недели.

12. В простую переменную с клавиатуры последовательно ввести *N* чисел и посчитать их произведение

13. Вычислить факториал введенного с клавиатуры целого числа *M*.

14. Распечатывать квадраты чисел, находящихся в диапазоне от 0 до 1 с шагом 0,2.

15. Распечатывать произведение чисел *A* и *B*, изменяющихся от 11 и −25 с шагом −3 и 5 соответственно до тех пор, пока это произведение — отрицательное число.

16. Известен поквартальный выпуск продукции для 5 цехов. Вывести среднее арифметическое значение выпуска по каждому цеху и определить суммарный выпуск по всем цехам.

17. В простую переменную последовательно вводят и суммируют положительные числа. Ввод заканчивается, когда сумма превысит 120. Вывести сумму.

18. В переменную последовательно вводят числа,

отличные от нуля. Окончание ввода — ноль. Определить среднее арифметическое отрицательных чисел.

19. В простую переменную последовательно вводятся N вещественных чисел. Вычислить максимальное значение среди введенных чисел, превышающих заданное число P.

20. В простую переменную последовательно вводятся числа, не равные 0. Определить минимальное значение и количество чисел, равных минимуму.

21. В простую переменную последовательно вводятся N чисел. Найти среди нечетных чисел: а) минимальное значение и порядковый номер последнего числа, равного этому значению; б) максимальное значение и порядковый номер первого числа, равного этому значению.

22. Диапазон задан числами L и R. Вводится последовательность произвольных чисел; окончание ввода — по желанию пользователя. Посчитать: а) сумму чисел, лежащих вне диапазона; б) количество чисел, кратных 7 и попадающих в диапазон.

23. В простую переменную последовательно вводятся числа. Окончание ввода — 0. Сколько чисел больше своих соседей справа?

24. В простую переменную последовательно вводятся N чисел. Определить, сколько чисел больше своих соседей слева и справа.

25. В простую переменную последовательно вводятся числа. Окончание ввода — по желанию пользователя. Все ли числа попадают в заданный диапазон (начало и конец диапазона вводятся с клавиатуры)?

26. В простую переменную последовательно вводятся N чисел, отличных от нуля. Четные или нечетные числа завершают последовательность и сколько их?

27. В простую переменную вводят N чисел. Сколько раз в последовательности меняется знак?

28. Напечатать изображение всех простых несократимых правильных дробей, знаменатель которых не превышает 9.

29. Вводится положительное целое число $A < 10000$. Определить знакочередующуюся сумму его цифр S, считая самый старший разряд положительным слагаемым. Массивы не использовать.

30. Вводится положительное целое число $B < 10000$. Все ли его цифры разные? Массивы не использовать.

31. Среди натуральных трехзначных чисел определить количество простых и распечатать их.

32. Представить вводимое с клавиатуры число в виде эквивалентной ему суммы троек и пятерок так, чтобы количество троек было минимальным.

33. Найти наибольший общий делитель для трех заданных чисел.

34. Найти наименьшее общее кратное для трех заданных чисел.

35. Найти минимальное количество слагаемых в сумме ряда $1 \cdot 1 + 1 \cdot 2 \cdot 2 + 1 \cdot 2 \cdot 3 \cdot 3 + \ldots + 1 \cdot 2 \cdot 3 \cdot \ldots \cdot K \cdot K$, при котором эта сумма станет больше 120.

36. Для произвольного значения аргумента «x» (по модулю больше 1) вычислить сумму

$$S = 1 - 1/x + 1/x^2 - 1/x^3 + \ldots + 1/x^k.$$

37. Для произвольного значения аргумента «x» (по модулю больше 2) вычислить сумму:

$$S = 1 + 2/x + 4/x^2 + 8/x^3 + \ldots$$

Вычисления продолжать до тех пор, пока очередное слагаемое больше заданного значения Z. Вывести на экран число слагаемых K.

38. Вычислить последнюю сумму членов ряда, при которой модуль разности между текущим и предыдущим членами ряда остается меньше 1. Кроме суммы,

вывести на экран значение последнего слагаемого и его номер.

$$\frac{1}{2} + \frac{1\cdot 2}{4} + \frac{1\cdot 2\cdot 3}{8} + \frac{1\cdot 2\cdot 3\cdot 4}{16} + \ldots$$

39. Вычислить сумму членов ряда. Суммировать до тех пор, пока разность между текущей и предыдущей суммами больше 0,001. Вывести сумму, последнее слагаемое, вошедшее в сумму, и его номер. Ряд имеет следующий вид:

$$\frac{1^2\cdot 5}{2\cdot 4} + \frac{2^2\cdot 5}{2\cdot 4\cdot 6} + \frac{3^2\cdot 6}{2\cdot 4\cdot 6\cdot 8} + \frac{4^2\cdot 6}{2\cdot 4\cdot 6\cdot 8\cdot 10} + \frac{5^2\cdot 7}{2\cdot 4\cdot 6\cdot 8\cdot 10\cdot 12} + \ldots$$

40. Составить программу вычисления приближенного значения суммы элементов ряда

$$S = 1 - 2 * x/1! + x^3/3! - 2 * x^5/5! + x^7/7! - \ldots$$

для произвольного значения аргумента «x» (по модулю меньше 1). Суммирование продолжать до тех пор, пока разность между очередным и предыдущим слагаемыми не станет меньше заданной величины Z (точности вычислений).

41. Дан одномерный массив из N элементов. Посчитать произведение элементов, кратных пяти и стоящих в позициях с четными номерами.

42. Определить в двумерном массиве минимальное значение и координаты элементов (номера строк и столбцов), равных ему.

43. Проверить, есть ли в трехмерном массиве целых чисел элементы, кратные 29. Если есть, вывести их координаты, если нет, выдать соответствующее сообщение.

44. Проверить, упорядочены ли по убыванию элементы одномерного массива.

45. Целочисленный массив (M строк, N столбцов) содержит числа от 0 до 9. Сколько раз встречается в массиве каждое число?

46. Ввести двумерный массив $M \times N$. Определить. а) минимальное значение в каждой строке; б) среднее арифметическое отрицательных элементов для каждого столбца.

47. Ввести двумерный целочисленный массив, содержащий 7 строк и 6 столбцов. Посчитать количество двузначных чисел в 5-й строке и произведение чисел, стоящих в нечетных позициях 2-го столбца. Предусмотреть проверку вводимых данных.

48. Ввести квадратный массив $N \times N$. Определить последнее минимальное значение для элементов, лежащих на побочной диагонали, и первое минимальное значение для элементов, лежащих на главной диагонали.

49. В квадратном массиве случайных чисел определить сумму положительных элементов, лежащих выше главной и побочной диагоналей.

50. В квадратном массиве целых чисел определить среднее арифметическое элементов, лежащих ниже побочной и выше главной диагонали. Распечатать координаты тех элементов массива, значение которых равно найденному среднему арифметическому.

51. Заполнить числовую квадратную матрицу 7×7 элементов согласно схеме:

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	2	2	2	1	0
0	1	2	3	2	1	0
0	1	2	2	2	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Проделать аналогичное с матрицей 8×8 элементов.

52. В двумерном массиве из $M \times N$ элементов поменять местами элементы столбцов с максимальной и минимальной суммами элементов.

53. Ввести массив из $M \times N$ произвольных чисел. Посчитать сумму элементов массива. Если она отрицательна, повернуть массив на 180°; если лежит в диапазоне от 0 до 10, повернуть на 90° против часовой стрелки, если превышает 100, — на 90° по часовой стрелке. В остальных случаях массив не поворачивать.

54. Ввести массив из $M \times N$ элементов. Изменить значения элементов, лежащих ниже горизонтальной оси, таким образом, чтобы массив стал симметричен относительно указанной оси.

55. Квадратная матрица $M \times M$ заполняется случайными двузначными числами. Посчитать количество диагоналей, параллельных главной, и суммы их элементов. Проделать подобное для диагоналей, параллельных побочной.

56. Дана целочисленная квадратная матрица из $N \times N$ элементов. Заменить элементы, лежащие ниже побочной диагонали, остатками от деления этих элементов на элементы, расположенные зеркально относительно указанной диагонали. Проделать аналогичное для элементов, лежащих выше главной диагонали. Дополнительных массивов не создавать.

57. Заполнить массив (10 строк, 9 столбцов) положительными и отрицательными числами в диапазоне от –10 до +10. Преобразовать фрагмент массива, ограниченный строками №№ 3 и 8 и столбцами №№ 4 и 7, в одномерный массив из 24 элементов.

58. Ввести одномерный числовой массив R из 40 элементов. Упорядочить его по убыванию на отрезке, ограниченном числами L и R, вводимыми с клавиатуры. Предусмотреть проверку вводимых данных.

59. Дан одномерный массив произвольных чисел. Отсортировать его по неубыванию абсолютных значений элементов.

60. Составить два алгоритма «пузырьковой» сортировки одномерного массива по возрастанию: а) «тя-

желые» элементы перемещаются к концу массива; б) «легкие» элементы перемещаются к началу массива.

61. Ввести двумерный числовой массив (M строк, N столбцов). Отсортировать все его строки по возрастанию. Отсортировать все столбцы полученного массива по возрастанию.

62. Вводятся две числовые последовательности из M и N элементов, упорядоченных по возрастанию. Сформировать из них массив, упорядоченный по убыванию, не используя сортировку последнего.

63. Дан одномерный массив, содержащий следующие элементы: 11, 28, 73, 44, 51, 99, 39, 62, 92, 87, 26, 41. Распечатать его четырьмя способами:

```
1.11 28 73 44    2.11 28 73    3.11 51 92   4.11 44 39 87
  51 99 39 62      44 51 99      28 99 87     28 51 62 26
  92 87 26 41      39 62 92      73 39 26     73 99 92 41
                   87 26 41      44 62 4
```

64. Ввести одномерный массив из N элементов. Удалить из него числа, кратные пяти, а кратные трем разделить на три. Посчитать количество удаленных и измененных элементов. Дополнительный массив не создавать.

65. Ввести целочисленный одномерный массив, содержащий числа от 0 до 20. Удалить повторяющиеся элементы. Предусмотреть проверку вводимых данных.

66. Дан одномерный массив, упорядоченный по убыванию. С клавиатуры вводят 5 чисел. Вставить их в массив, не нарушая упорядоченности. Вытесненные элементы сохранить в отдельном массиве, упорядоченном по убыванию.

67. Ввести одномерный массив из N элементов. Изменить порядок следования значений элементов на обратный на отрезках от начала массива до позиции A и от позиции B до последнего элемента. Числа A и B вводятся с клавиатуры. Отрезки пересекаться не должны.

68. Ввести одномерный массив из N элементов. Производить кольцевой сдвиг его элементов до тех

пор, пока минимальный элемент (считаем его единственным) не окажется в позиции с индексом K. Число K вводят с клавиатуры. Предусмотреть проверку вводимых значений.

69. Ввести одномерный массив из N случайных целых чисел в диапазоне от -5 до $+25$. Вычислить среднее арифметическое остатков от деления значений элементов на их индексы.

70. Дан одномерный массив из N произвольных чисел и некоторое целое число K, не превышающее $N/2$. Присвоить элементам с индексами, превышающими K, значения элементов, расположенных симметрично относительно K, а при отсутствии таковых — обнулить. Вывести на экран исходный и результирующий массивы. Предусмотреть проверку вводимой информации.

71. В трехмерном массиве случайных чисел ($X \times Y \times Z$) определить номера горизонтальных плоскостей, сумма элементов которых минимальна.

72. В двумерном массиве случайных чисел (M строк, N столбцов) переставить столбцы так, чтобы суммы их элементов убывали.

73. В массиве целых чисел, инициализированном случайными числами, необходимо все отрицательные элементы переставить в конец массива, сохраняя их исходную последовательность.

74. В двумерный массив размером $N \times N$ ввести числа, модуль которых меньше 9. Найти в данном массиве количество квадратов размером $K \times K$, сумма элементов которых максимальна. N и K заданы. Вывести на экран количество таких квадратов и значение максимальной суммы.

75. В массив размером $N \times N$ (N — заданное число не более 9) ввести числа. Каждый из элементов массива заменить на суммы тех элементов, которые находятся выше и левее его. Элементы первого столбца и пер-

вой строки не менять. Вывести на экран измененный массив в виде таблицы.

76. Ввести текст длиной до 254 символов в виде строки, а затем определить, правильно ли в нем расставлены круглые скобки, т. е. находится ли справа от каждой открывающей скобки соответствующая ей закрывающая скобка, а слева от каждой закрывающей — соответствующая ей открывающая.

77. Ввести 16 цифр прямого машинного кода целого положительного числа. Напечатать число в 16-й системе счисления, соответствующее этому машинному коду, без левых незначащих нулей. Предусмотреть проверку правильности ввода информации.

78. Ввести зашифрованный текст, не содержащий знаков препинания и строчных букв. Зашифрованные слова записаны задом наперед и отделены друг от друга пробелами. В начале, в конце текста и между словами могут быть лишние пробелы. Напечатать только те расшифрованные слова, которые могут начинаться с той же буквы, что и первое слово текста, в порядке убывания их длины.

79. Среди простых трехзначных чисел найти самое большое, запись которого в двоичной системе счисления содержит максимальное число единиц. Кроме того, найти количество чисел, двоичное представление которых имеет такое же количество единиц.

80. Дана строка длиной до 254 символов. В качестве разделителей слов в предложении используются пробелы. Вывести на экран сообщение: сколько слов в этом предложении не содержат ни одной буквы «W».

81. Дана строка длиной до 254 символов. Удалить все пробелы перед знаками препинания.

82. В строке длиной до 254 символов заменить все слова «да» на «возможно», а «возможно» на «нет» и вывести результат.

83. На причале ждут посадки N беженцев. О каждом из них известна фамилия и вес вместе с багажом. Капитан хочет посадить на борт как можно больше пассажиров. Определить максимально возможное количество пассажиров и составить их список для посадки, если известна грузоподъемность судна. (Сортировку исходных массивов не производить.)

84. Ежедневно в течение апреля измеряли уровень шума в каждом из N районов города (N — задано). Найти первый номер района, в котором наблюдался максимальный уровень, первое число месяца, когда наблюдался этот уровень, а также определить, каким днем недели является это число, если известно, с какого дня недели начинался апрель.

85. Известны средние температуры каждого из 365 дней года. Для каждого месяца года определить даты начала и конца первого наибольшего периода с постоянной средней температурой и значение этой температуры. Учесть, что искомый период не может длиться менее суток.

86. Известны названия N предприятий, а также объемы выпуска продукции каждого из них за каждый год из последних M (N и M заданы). Напечатать список предприятий, ежегодно увеличивавших объем выпуска продукции, указав общий выпуск продукции каждого предприятия за все годы, а также общее число таких предприятий.

87. В театральную кассу поступило N заявок (N — заданное число) от школ. Каждая заявка содержит название одного спектакля и необходимое количество билетов. На основе этого определить популярность каждого из спектаклей, перечислив их названия в порядке убывания количества заказанных на них билетов.

88. Каждому из N опрошенных граждан предлагали выбрать вероятного кандидата в президенты из предложенного списка M политиков (N и M заданы).

Опрошенные или называли одну фамилию из списка, или отказывались отвечать. Напечатать перечень фамилий тех политиков, которых никто не назвал, или сообщить, что таких нет. Проверку ввода списка политиков не делать.

89. Известны фамилии N шахматистов и результаты их встреч на шахматном турнире (N задано). Результат встречи каждой пары участников турнира оценивался следующим образом: 2 — победа, 1 — ничья, 0 — поражение. Определить сумму очков, которую набрал шахматист с заданным номером, а также фамилии других участников, набравших такое же количество очков.

90. Известно название дня недели 1-го января. Напечатать все числа, соответствующие последним четвергам каждого месяца с указанием названия месяца. В году 365 дней.

91. Каждый из N (N — задано) опрошенных членов жюри кинофестиваля назвал 1, 2 или 3 фильма, как возможных претендентов на первое место. Определить название фильма, который является, по мнению опрошенных, наиболее вероятным победителем. Напечатать список таких фильмов в случае, если они получили одинаковое количество голосов.

Информационные технологии

Технология обработки текстовой информации

Системы подготовки текстовых документов

Редактор текстов (text editor) — это компьютерный продукт, обеспечивающий ввод, изменение и сохранение любого символьного текста. Исторически первым с помощью компьютера стал обрабатываться именно программный текст, т. е. текст, записанный на том или ином искусственном языке программирования.

Существующие в настоящее время системы подготовки текстовых документов можно классифицировать как по объему их функциональных возможностей, так и по их предназначению. Среди программных продуктов, относящихся к данным системам, можно выделить: обычные экранные текстовые редакторы и развитые системы подготовки текстов на естественных языках.

Среди систем подготовки текстов на естественных языках можно выделить три больших класса: простые форматеры, текстовые процессоры и настольные издательские системы. Исходя из внутримашинной структуры подготавливаемого документа, можно предложить следующий подход к классификации систем подготовки текстов.

Форматер — система подготовки текстов, которая не использует для внутреннего представления текста

никаких дополнительных кодов, кроме стандартных ASCII символов (конец строки, перевод каретки, конец страницы и т. п.).

Текстовый процессор — система подготовки текстов, которая во внутреннем представлении снабжает текст специальными кодами, т. е. разметкой.

В основном, экранные редакторы и текстовые процессоры различаются по назначению: первые создают ASCII-файлы, которые используются затем компиляторами или форматерами, а вторые — предназначены для подготовки текстов для последующей печати на бумаге, когда форма представления текста имеет большое значение.

Обычные текстовые редакторы. Обычный текстовый редактор, как правило, предназначен для подготовки на компьютере текстов, которые в конечном итоге являются программами. А так как текст программы не требует особого форматирования, т. е. автоматического преобразования расположения элементов текста, изменения шрифта и т. п., то и набор операций такого текстового редактора определяют только особенности построчной записи текстов на языке программирования.

Результатом работы экранного редактора является файл, в котором все знаки являются печатными символами кодовой таблицы ASCII и который не содержит особых, специфичных для данного редактора, знаков. Такие файлы называются ASCII-файлами.

Различаясь способами управления и набором сервисных возможностей, все они в том или ином виде позволяют:

- набирать текст на экране, используя до 200 символов;
- исправлять ошибочные символы в режиме замены;
- вставлять и удалять группы символов (слова) в пределах строки, сдвигая вправо/влево в режиме

вставки или замещения символов не изменяющуюся часть строки целиком;

- удалять одну или несколько строк, размножать их или перемещать в другое место текста;
- раздвигать строки существующего текста, чтобы вставить туда новый фрагмент;
- вставлять группы строк из других текстов;
- обнаруживать все вхождения определенной группы символов (контекста);
- заменять один контекст другим, возможно, разной длины;
- сохранять набранный текст для последующих корректировок;
- печатать текст на разных типах принтеров стандартными программами печати одним шрифтом в пределах документа.

К группе обычных текстовых редакторов относятся такие как Norton Editor, SideKick, Brief и многофункциональный многооконный редактор Multi-Edit.

Редакторы текста для подготовки документов на естественном языке. Когда целью пользователя является подготовка текстов на естественном языке, набор операций редактора должен быть существенно расширен, и программный продукт переходит в новое качество — становится настоящей системой подготовки текстов. Компьютерные программы таких систем ориентированы на работу с текстами, имеющими структуру документа, т. е. состоящими из абзацев, страниц и разделов.

Текстовые процессоры имеют специальные функции, которые предназначены для облегчения ввода текста и представления его в напечатанном виде. Набор этих функций предполагает обеспечение следующих возможностей.

- Ввод текста под контролем функций форматирования, обеспечивающих немедленное изменение вида страницы текста на экране и расположение слов на ней. Данная возможность, давая приближенное представление о действительном расположении текста на бумаге, обычно обеспечивается предварительной настройкой текстового процессора на принтер, на котором предполагается печатать текст.
- Возможность предварительного описания структуры будущего документа с помощью специального языка. В этом описании задаются такие параметры, как величина абзацных отступов, тип и размер шрифта для различных элементов текста, расположение заголовков, междустрочные расстояния, число колонок текста, расположение и способ нумерации сносок (в конце текста или на той же странице) и тому подобное.
- Возможность автоматической проверки орфографии и получения подсказки при выборе синонимов.
- Возможность ввода и редактирования таблиц и формул с отображением их на экране в том виде, в каком они будут напечатаны.
- Возможность объединения документов в процессе подготовки текста к печати.
- Возможность автоматического составления оглавления и алфавитного справочника.

Практически все текстовые процессоры имеют уникальную структуру данных для представления текста, что объясняется необходимостью включения в текст дополнительной информации, описывающей структуру документа, шрифты и тому подобное, поскольку каждое слово или даже символ могут иметь свои особые характеристики. Поэтому текст, подготовленный с помощью одного текстового процессора,

как правило, не может быть прочитан другими текстовыми процессорами и, следовательно, не может быть отредактирован и напечатан.

В целях совместимости текстовых документов при переносе их из среды одного текстового процессора в другой существует особый вид программного обеспечения — *конвертеры*, обеспечивающие получение выходного файла в формате текстового процессора — получателя документа. Программа-конвертер на входе получает информацию в одном файловом формате, а как результат своей работы выдает информацию в виде файла в требуемом формате. Дальнейшее усовершенствование систем обработки текстов привело к тому, что автономные программы-конвертеры практически прекратили свое существование и вошли составной частью в систему подготовки текстов. Сегодня наиболее яркие представители программ текстовой обработки поддерживают популярные файловые форматы за счет встроенных модулей конвертации.

Существующие в настоящее время текстовые процессоры значительно отличаются друг от друга характеристиками, возможностями по вводу и редактированию текста, его форматированию и выводу на печать, а также по степени сложности освоения пользователем. Наиболее мощными текстовыми процессорами, позволяющими подготовить и напечатать большие и сложные документы, включая книги, относятся WinWord, WordPerfect, ChiWriter, WordStar 2000. AmiPRo и T3.

Современные текстовые процессоры имеют очень широкие вспомогательные возможности, обеспечивающие удобную и эффективную работу пользователя. Основными свойствами текстовых процессоров, определяющими наличие этих возможностей, являются следующие.

1. *Многовариантность выполнения операций.* Практически все операции могут быть выполнены одним из трех-четырех способов, пользователь выбирает наиболее удобный.

2. *Справочная система.* Формирование справки в виде гипертекста позволяет легко и быстро осуществлять поиск нужной темы.

3. *Контекстное меню.* Разворачивается по щелчку кнопки (обычно правой) мыши на выбранном объекте. Речь идет, например, о месте таблицы, где в данный момент хочет работать пользователь. Наиболее часто используемые функции обработки, доступные в данной ситуации, собраны в контекстном меню.

4. *Контекстная подсказка.* Вызывается из контекстного меню или нажатием соответствующей кнопки в пиктографическом меню.

5. *Пиктографическое меню.* Наиболее часто используемым командам соответствуют пиктограммы, расположенные под строкой меню. Они образуют пиктографическое меню. Вследствие щелчка мышью на пиктограмме выполняется связанная с ней команда. Пиктографические меню могут быть составлены индивидуально.

6. *Средства для оформления и модификации экрана и документов.* Внешний вид рабочего окна и прочих элементов экранного интерфейса может быть определен в соответствии с требованиями пользователя, что позволяет сделать работу максимально удобной. Среди таких возможностей — разбиение экрана на несколько окон, использование различных форм представления и масштабирования документов, наличие множества панелей инструментов и т. п.

7. *Средства оформления и вывода на печать документа.* Для удобства пользователя предусмотрены все функции, обеспечивающие форматирование и печать документа, такие как выбор шрифта, цвета и стиля, вы-

бор размера страницы, разбиение на страницы, установка размера полей страниц, оформление колонтитулов, а также предварительный просмотр получившейся страницы.

8. *Шаблоны и другие средства автоматизации.* Текстовые процессоры позволяют автоматизировать рутинные операции оформления документа с помощью использования шаблонов, средств автокоррекции, грамматической и стилистической проверки текста и т. п.

9. *Средства структурирования, связывания и встраивания данных.* Пользователь имеет возможность вставить в текст документа в виде элементов различные рисунки, таблицы, графические изображения и т. п. Такие элементы могут создаваться как средствами самого текстового процессора, так и другими программами. Текстовые процессоры позволяют выполнять разнообразные действия по встраиванию и связыванию этих элементов в документе.

Настольные издательские системы. *Настольные издательские системы* (desktop publishing) готовят тексты по правилам полиграфии и с типографским качеством. Подобно тому как текстовые процессоры не являются «развитием» форматеров, настольные издательства не являются более совершенным продолжением текстовых процессоров, так как у них совсем иная предметная область.

Пакеты программ настольного издательства, по сути являются инструментом верстальщика. Предназначены программы этого класса не столько для создания больших документов, сколько для реализации различного рода полиграфических эффектов. Иначе говоря, программа настольного издательства позволяет легко манипулировать текстом, менять форматы страниц, размер отступов, дает возможность комбинировать различные шрифты, работать с материалом до

получения полного удовлетворения от внешнего вида как отдельных страниц (полос издания), так и всего издания.

Среди систем подготовки текстовых документов в этом классе можно также предложить деление на две подгруппы: настольные издательства профессионального уровня и издательские системы начального уровня. Системы первой подгруппы предназначены для работы над изданиями документов со сложной структурой или типа иллюстрированного журнала. К системам профессионального уровня можно отнести такие продукты, как QuarkXPress for Windows, FrameMaker for Windows, PageMaker for Windows. Данная книга сделана с помощью издательской системы LaTeX. Системы второй группы обычно не предназначаются для получения промышленной полиграфической продукции. Все пакеты данной категории ориентируются на новичка и пользователя, который отдает издательской деятельности лишь часть своего рабочего времени. Наиболее распространенными продуктами этой группы являются Microsoft Publisher и Pageplus for Windows.

Технология подготовки документа с помощью текстового процессора

Основные этапы подготовки текстового документа. Подготовка текстов с использованием текстового процессора заключается в последовательном выполнении ряда этапов. С некоторой долей условности можно выделить:

- набор текста;
- редактирование введенной информации;
- форматирование (оформление) отдельных структурных элементов будущего документа;
- печать документа;

- сохранение текста документа и ведение архива текстов.

Каждый этап состоит из множества операций. Набор операций определяется конкретной программой, выбранной для подготовки документа.

Операции редактирования текстового документа. Созданный на этапе набора текст документа в дальнейшем может подвергаться изменениям. К основным операциям редактирования принято относить следующие:

- добавление;
- удаление;
- перемещение;
- копирование фрагмента текста.

К числу операций редактирования можно также отнести операцию поиска и контекстной замены.

Под *фрагментом* понимается область текста, указанная (выделенная, маркированная) пользователем. Минимальный размер фрагмента — один символ, максимальный — весь текст документа. Выделение текста является одним из основополагающих принципов работы системы подготовки текстов. Основная концепция большинства систем этого назначения — «выдели и обработай». Выделение фрагмента документа может производиться с помощью мыши или клавиатуры. Выделенный фрагмент в окне редактирования отмечается либо цветом, либо негативным изображением.

Размеры редактируемого текстового документа обычно превышают размер области экрана дисплея для ввода, предоставляемой системой подготовки текста. Для того чтобы пользователь мог работать с нужным ему фрагментом, система подготовки текста обеспечивает возможность перемещения текстового курсора к тому месту документа, где в дальнейшем будет произведена любая операция с текстом. Обычно

для этой цели используются клавиши клавиатуры, либо перемещение по тексту осуществляется с помощью специальных графических компонентов интерфейса — линеек прокрутки с бегунками.

Для *добавления* одного или нескольких символов система подготовки текстов должна находиться в режиме вставки, а текстовый курсор — в том месте документа, где производится дополнительный набор текста. Индикация режима замены или вставки производится в статусной строке служебной области окна программы редактирования. При наборе очередного добавляемого символа часть строки справа (включая курсор) сдвигается на одну позицию вправо, а введенный символ появляется в позиции курсора. Если включен режим замены, то вновь набираемые символы замещают присутствующие в тексте редактируемого документа символы.

Для *удаления* одного или нескольких символов используется клавиши **Delete** или **Backspase**. При нажатии клавиши **Delete** удаляется символ в позиции курсора, правая часть строки сдвигается влево, сам курсор остается на месте.

При нажатии клавиши **Backspase** удаляется символ в позиции слева от курсора, курсор и правая часть строки сдвигаются влево. Эта клавиша используется, в основном, для удаления одного или нескольких только что набранных символов. Технология удаления больших фрагментов текста предполагает предварительное выделение фрагмента для редактирования. Как правило, в текстовом окне может быть выделен только один фрагмент.

Удаление может быть произведено в двух вариантах:

- выделенный фрагмент изымается из текста, оставшийся текст смыкается;

- выделенный фрагмент удаляется в специальный буфер временного хранения, откуда может быть извлечен для вставки в другое место редактируемого документа, либо использован в текстах других документов (если система подготовки текстов поддерживает многооконный режим для одновременной работы с несколькими документами). Содержимое временного буфера сохраняется в течение сеанса работы или до помещения в него новой порции информации.

Для *копирования* информации используется технология, во многом похожая на предыдущую:

- предварительно копируемый текст должен быть выделен, а затем специальной командой «Копировать» системы подготовки текстов помещен во временный буфер хранения. При этом в буфер попадает копия фрагмента, сам он по-прежнему располагается в тексте документа;
- текстовый курсор помещается в новую позицию для вставки;
- копия фрагмента извлекается из буфера и располагается, начиная с указанной курсором позиции, существующий справа от курсора текст сдвигается вправо.

Для выполнения *перемещения* фрагмента текста с использованием временного буфера хранения технологические операции следующие:

- выделение нужного фрагмента;
- удаление в буфер временного хранения;
- перемещение курсора в нужное место документа;
- вставка содержимого буфера в документ.

Проблема использования содержимого временного буфера хранения решается в настоящее время не только за счет возможности многократного использования, но и предоставлением возможности хранения нескольких фрагментов текста одновременно. Напри-

мер, текстовый процессор Word из комплекса Office 2000 позволяет сохранять до 12 фрагментов текста одновременно и использовать их для редактирования выборочно или всех сразу.

Существующие на сегодняшний день графические оболочки (например, Windows) поддерживают технику drag-and-drop работы манипулятора «мышь» в среде системы подготовки текста. Эта техника предполагает, что для операций перемещения и копирования временный буфер не задействуется. Однако в этом случае копирование или перемещение фрагмента возможно только один раз, тогда как временный буфер хранения предлагает использовать помещенную в него информацию столько раз, сколько это необходимо пользователю.

Операция *поиска* в среде программы обработки текста может выполняться в нескольких вариантах. Это может быть поиск по образцу, например, для последующей замены найденного словосочетания на другое. Действия пользователя системы подготовки текстового документа сводятся к следующей цепочке операций:

- задается некоторый образец (символ, слово или цепочка символов);
- указывается направление поиска (вперед от текущей позиции курсора либо назад);
- система подготовки текстов начинает поиск заданного фрагмента; при обнаружении последнего просмотр приостанавливается, курсор позиционируется перед искомым фрагментом и пользователь имеет возможность произвести нужную коррекцию.

Другой вариант поиска предполагает, что текст предварительно размечается специальными служебными метками (закладками или bookmarks), а затем

система подготовки текста осуществляет перевод курсора к метке, заданной пользователем по ее имени.

Команда *замены* производит замещение одного заданного контекста на другой. Замена может производиться в рамках выделенного фрагмента, по всему тексту либо после подтверждения пользователем каждого варианта замещения.

Операции форматирования (оформления) текстового документа. Современные средства подготовки текстовых документов используют два типа оформления структурных элементов текста. Это *непосредственное оформление*, когда форматирование применяется к предварительно выделенному фрагменту через команды меню и *оформление с помощью стиля*.

Рассмотрим подробнее процесс непосредственного форматирования. Каждый документ, создаваемый средствами текстового процессора, имеет в качестве основы некоторое оформление по умолчанию. Набор параметров (или атрибутов оформления), а также их конкретные величины определяются программой текстового процессора.

Например, текстовый процессор Word для Windows предлагает следующие параметры оформления документа:

- символы — нормальной насыщенности, кегль 10 пунктов;
- абзацы — без отступов, выровнены влево, через один интервал;
- величина табуляции — через 0,5 дюйма (или 1,27 см);
- размер печатной страницы документа — формат А4 (210 мм на 297 мм);
- границы текста на странице — левое и правое 3,17 см, верхнее и нижнее 1,5 см.

Таким образом, каждый документ создается по некоторому подобию или по шаблону уже существующего документа.

Различают три типа форматирования прозаических документов:

- *символьное* (или шрифтовое оформление);
- форматирование *абзаца* документа;
- *оформление (верстка) страниц* (или *разделов*) документа.

Стандартными параметрами символьного оформления являются:

- тип (гарнитура) шрифта;
- кегль (высота) символов шрифта;
- начертание литер (полужирный, курсив, полужирный курсив, обычный);
- подчеркивание;
- цвет символов;
- расположение символов относительно опорной линии строки (верхний и нижний индекс).

Внешний вид документа в большой степени зависит не только от форматирования символов текста, но и от оформления абзацев текста. *Абзац* является одним из основных структурных элементов прозаического документа. Обычно новый абзац в тексте образуется при нажатии клавиши **Enter** на клавиатуре при наборе текста. При этом курсор ввода переходит на новую строку и устанавливается в позицию левого отступа следующего абзаца. Позиция отступа зависит от параметров настройки конкретной системы текстовой обработки. Набор параметров абзацного форматирования, аналогично набору атрибутов символьного форматирования, зависит от конкретной программной среды, в которой изготавливается текстовый документ. К наиболее общим атрибутам можно отнести задание:

- типа выравнивания границ строк;
- отступов для строк;
- межстрочных интервалов;
- обрамления и цвета фона текста;
- расположения текста абзаца на смежных страницах документа.

Форматирование многостраничного текстового документа. Если система подготовки текста используется для создания и оформления многостраничного документа, то в тексте могут появиться новые структурные элементы: колонтитулы, сноски, закладки, перекрестные ссылки.

Под *закладкой* или *меткой* понимается определенное место в тексте документа, которому пользователь присваивает имя. В дальнейшем закладка в многостраничном документе может использоваться для:

- быстрого перехода к месту документа, обозначенному закладкой;
- создания перекрестных ссылок в документе.

Иногда по ходу чтения документа необходимы дополнения к основному тексту, подстрочные примечания. Подстрочные примечания оформляют *сносками*. В состав подстрочного примечания входят два неразрывно связанных элемента: знак сноски и текст собственно примечания. Знак сноски располагают в основном тексте у того места, к которому относится примечание и в начале самого примечания. Рекомендуется в текстовом материале использовать знак сноски в виде арабских цифр, а в цифровом — в виде букв или знаков.

Перекрестная ссылка — это текст, предлагающий читателю документа обратиться к другому фрагменту текста или рисунку, содержащемуся в тексте. Например:

«Вернитесь к разделу «Базовые функции редактирования текста» (страница ###)».

В случае изменения названия раздела или расположения его на другой странице в результате коррекции документа подобный текст необходимо изменить вручную. Наиболее мощные системы подготовки текстов позволяют автоматически отслеживать процесс изменения за счет организации перекрестных ссылок на элементы текста, отмеченные специальным образом как закладки, заголовки, сноски, рисунки или формулы.

Колонтитулом (Running head) называется одинаковый для группы страниц текст (графическое изображение), расположенный вне основного текста документа на полях печатной страницы. Различают верхний (Header) колонтитул, который обычно расположен над текстом документа и нижний (Footer), располагаемый ниже основного текста. Порядковые номера страниц входят в колонтитул. Их называют *колонцифрами.*

Стандартными параметрами оформления страниц документа являются:

- поля страниц;
- размер печатного листа и ориентация текста на бумаге;
- расположение колонтитулов;
- количество колонок текста (газетный стиль).

Подготовка текстовых документов с использованием шаблонов и стилей. Современные информационные технологии активно используют идею сохранения не только конкретного результата работы в виде документа, но и совокупности действий, с помощью которых этот результат был достигнут. Эта идея в системах подготовки текстовых документов реализована в виде механизма шаблонов.

Шаблон — это документ специального типа, который содержит разнообразную информацию о стилях форматирования частей документа, вставленных

полях и т. д. В шаблоне также хранятся макрокоманды, элементы глоссария, кнопки панели инструментов, нестандартные меню и установки клавиш сокращения, облегчающие работу с документами. Будучи один раз подготовленным и сохраненным в памяти компьютера, шаблон позволяет быстро изготавливать аналогичные по форме (но не по содержанию) документы без затрат времени на форматирование.

Специальным инструментом для описания используемых правил форматирования фрагментов разрабатываемого документа является стиль. Точнее *стиль* — это описание оформления текста, которое именуется и запоминается в шаблоне, на котором базируется документ. Стиль состоит из двух частей: его имени и инструкции форматирования. Имя стиля служит для его идентификации, а инструкция форматирования описывает оформление, которое использует текстовый процессор при применении данного стиля к фрагменту текста.

Главное преимущество стилей перед «непосредственным» форматированием заключается в том, что вы можете изменять стандартные атрибуты форматирования встроенных стилей, а также создавать свои собственные стили. Еще одно преимущество стилей — особенно в офисной среде — заключается в том, что используется стандартное оформление документов на основе ранее созданных стилей. При каждом изменении атрибутов форматирования, связанных с данным стилем, все абзацы, к которым он применен, будут автоматически переформатированы.

В комплектах поставки текстовых процессоров всегда содержатся обширные библиотеки готовых стилей, однако необходимо заметить, что большинство готовых шаблонов не соответствуют требованиям государственного стандарта по оформлению документов.

Подготовка «составных» документов и «внедрение» в них объектов. Создание «большого» документа должно начинаться с разработки его структуры, подчиненности и иерархии заголовков.

Под *структурой документа* понимается схема, определяющая взаиморасположение и связь его составных частей. На самом верхнем уровне иерархии находится название всего документа, а на нижних уровнях располагаются названия его отдельных структурных элементов, содержание которых и составляет смысловую часть документа. Для эффективной работы с большими документами, пользователь текстового процессора имеет в своем распоряжении следующий стандартный набор операций:

- создание структурированного документа и реорганизация его структуры, как то: повышение или понижение уровня иерархии некоторых заголовков;
- просмотр структуры документа с выводом на экран только заголовков определенного уровня иерархии;
- создание сносок, указателей, оглавления, ссылок, списка иллюстраций, закладок.

Большинство текстовых процессоров поддерживает концепцию составного документа — *контейнера*, включающего в себя объекты различных форматов. Пользователь имеет возможность вставить в текст документа различные рисунки, таблицы, графические изображения, подготовленные в других программных средах. Для выполнения таких задач используется технология связи и внедрения объектов (Object Linking and Embedding — OLE). Согласно этой технологии, любая вставляемая в документ или связываемая с документом информация обозначается абстрактным понятием «объект», а в качестве объекта могут выступать текст, рисунок, звук и даже видеоинформация.

Встраивание в документ графических объектов может сделать его более привлекательным. При добавлении рисунка в документ он присоединяется к находящемуся вокруг тексту. Если абзац, содержащий рисунок, перемещается вверх или вниз по странице, рисунок перемещается вместе с ним. Кроме того, большинство развитых систем компьютерной обработки текста имеет встроенные инструменты рисования для создания графических примитивов, с помощью которых легко изображать линии, стрелки, эллипсы, прямоугольники, окружности, дуги, сектора и различные кривые. После создания графического объекта его можно залить цветом или узором, изменить цвет и тип линий, увеличить или уменьшить, переместить, повернуть или зеркально отобразить. Объекты в документе можно комбинировать, создавая единый рисунок. Объекты можно копировать и вставлять в другое место документа или другого документа.

Графическое изображение можно вставить в документ и из файла. Импорт различных графических объектов возможен благодаря встроенным или внешним преобразователям форматов (графическим фильтрам). Графика может быть закодирована двумя принципиально разными способами: растровыми изображениями (bitmap images) и векторными рисунками.

Файлы растровой (или битовой) графики содержат в определенной последовательности совокупности отдельных точек изображения, называемых пикселами (от англ. picture element). Существует несколько форматов файлов растровой графики, и каждый формат предусматривает собственный способ кодирования информации о пикселах и другой присущей компьютерным изображениям информации. Среди наиболее распространенных форматов можно отметить BMP, PCX, GIF, TIFF и JPEG. Недостатком растрового изо-

бражения является потеря качества при увеличении масштаба рисунка.

Файлы с векторным изображением каждый отдельный элемент рисунка описывают и хранят в виде массива параметров — математического описания элементарных геометрических фигур. При каждом отображении векторное изображение перерисовывается компьютером, что несколько замедляет работу, но в то же время позволяет получить изображение с высоким разрешением. Наиболее популярны векторные форматы WMF, CDR, DXF.

Размещение графических фрагментов в текстовом документе производится с использованием кадров. *Кадр* — это хранилище для размещения объектов в области страницы, не управляемой параметрами полей печатной страницы (например, между колонками текста или в области полей страницы). Кадр появляется как окно вокруг заключаемого объекта и обладает особыми свойствами. Одно из самых важных свойств кадра — возможность окружать объект текстом. Развитые системы компьютерной подготовки текстов позволяют использовать кадры как прямоугольной, так и неправильной формы. Другое важное свойство — способность изменять размер и местоположение на странице.

Технология обработки табличных данных

Общая характеристика современных табличных процессоров

Современные текстовые процессоры, как правило, содержат минимальный набор средств работы с табличными данными. Однако наиболее эффективным

способом обработки такого рода информации является использование специализированных программных продуктов — табличных процессоров. Наряду с общими вспомогательными средствами обеспечения удобной и эффективной работы пользователя, табличные процессоры также предоставляют следующие специализированные функциональные возможности.

1. *Рабочие папки и рабочие листы.* Табличные документы можно объединять в рабочие папки, так что они могут рассматриваться как одно единое целое, если речь идет о копировании, загрузке, изменении или других процедурах. В нижней части электронной таблицы расположен указатель, который обеспечивает доступ к отдельным рабочим листам папки. Пользователь может задавать название листам, что делает наглядным содержимое папки и облегчает переход от документа к документу.

2. *Средства для оформления и модификации экрана и таблиц.* Среди средств обеспечения удобства работы пользователя имеются не только стандартные средства разбиения экрана на отдельные окна, но и средства фиксации на экране заголовков строк и столбцов.

3. *Средства оформления и вывода на печать таблиц.* Для удобства пользователя предусмотрены все функции, обеспечивающие печать таблиц, такие как: выбор размера страницы, разбиение на страницы, установка размера полей страниц, оформление колонтитулов, а также предварительный просмотр получившейся страницы.

4. *Средства оформления рабочих листов.* Современные табличные процессоры предоставляют широкие возможности по форматированию таблиц, такие как: выбор шрифта и стиля, выравнивание данных внутри клетки, выбор цвета фона клетки и шрифта, изменение высоты строк и ширины колонок, черчение рамок различного вида, определение формата данных внутри

клетки (например: числовой, текстовый, финансовый, дата и т. д.), а также обеспечение автоматического форматирования, когда в систему уже встроены различные варианты оформления таблиц, и пользователь может выбрать наиболее подходящий формат из уже имеющихся.

5. *Связывание данных.* Абсолютная и относительная адресации являются характерной чертой всех табличных процессоров, в современных системах они дают возможность работать одновременно с несколькими таблицами, которые могут быть тем или иным образом связаны друг с другом.

6. *Вычисления.* Для удобства вычисления в табличных процессорах имеются встроенные функции, а именно: математические, статистические, финансовые, функции даты и времени, логические и другие. Менеджер функций позволяет выбрать нужную функцию и, подставив значения, получить результат.

7. *Деловая графика.* Трудно представить современный табличный процессор без возможности построения различного типа двумерных, трехмерных и смешанных диаграмм.

8. *Выполнение табличными процессорами функций баз данных.* Эта возможность обеспечивает заполнение таблиц аналогично заполнению базы данных, т. е. через экранную форму; защиту данных, сортировку по ключу или по нескольким ключам, обработку запросов к базе данных, создание сводных таблиц. Кроме этого, все современные программы работы с электронными таблицами включают средства обработки внешних баз данных, которые позволяют работать с файлами, созданными, например, в формате dBase или PARADOX или других форматах.

9. *Моделирование.* Подбор параметров и моделирование — одни из самых важных возможностей табличных процессоров. С помощью простых приемов можно

находить оптимальные решения для многих задач. Методы оптимизации варьируются от простого подбора до метода линейной оптимизации со многими переменными и ограничениями.

10. *Программирование.* В простейшем случае для автоматизации выполнения часто повторяемых действий можно воспользоваться встроенным языком программирования макрокоманд. Разделяют макрокоманды и макрофункции. Применяя макрокоманды, можно упростить работу с табличным процессором и расширить список его собственных команд. При помощи макрофункций можно определять собственные формулы и функции, расширив таким образом набор функций, предоставляемый системой. Для создания приложений табличные процессоры содержат в качестве дополнительной компоненты язык программирования высокого уровня (например, компоненты языка Visual Basic для составляющих Microsoft Office).

Элементы рабочего окна табличного процессора. При загрузке табличного процессора обычно открывается рабочее окно, отображающее инструментальные панели, информационные поля и собственно электронную таблицу. Отображаемая электронная таблица, как правило, является одной из многих, объединенных в книгу или блокнот, являющихся объектом хранения в отдельном файле. В связи с этим отдельная таблица именуется рабочим листом.

Рабочее окно современного табличного процессора состоит из следующих элементов:

1. *Строка заголовка.*

Строка заголовка содержит:

- имя программы (табличного процессора);
- имя текущего файла (книги или блокнота);
- кнопки управления окном, состав и назначение которых определяются операционной средой, в которой работает табличный процессор (обычно

это кнопки преобразования окна в пиктограмму, изменения размера окна, закрытия окна).

2. *Строка меню.*

В строке меню расположены имена основных групп команд и инструментальных средств табличного процессора, каждой из которых соответствует собственное выпадающее меню, детализирующее возможности управления электронными таблицами. Обычно сюда помещают следующие основные группы команд:

- управление рабочим окном в целом (сворачивание и восстановление окна, перемещение и изменение размеров окна, закрытие окна);
- управление файлами (открытие и закрытие файлов с электронными таблицами, создание новых файлов, сохранение электронных таблиц в различных форматах, управление печатью электронных таблиц, управление передачей данных электронных таблиц в другие приложения, завершение работы табличного процессора);
- редактирование элементов электронной таблицы (удаление, перемещение, копирование и очистка фрагментов электронных таблиц, поиск и замена данных в электронных таблицах, заполнение фрагментов электронных таблиц);
- управление представлением элементов рабочего окна (определение размера отдельных компонентов рабочего окна, определение масштаба отображения данных электронной таблицы, определение количества и состава инструментальных панелей и т. п.);
- управление режимом и содержанием вставки (определение места и содержания вставляемых объектов: ячеек, строк, столбцов, рабочих листов, диаграмм, гиперссылок и т. п.);

- форматирование элементов электронной таблицы (оформление вида представления содержания ячеек, определение вида выравнивания, управление шрифтовым оформлением, представлением границ, цветовым оформлением и т. п.;
- сервисные команды (управление макросами, режимами и настройками организации работы с табличным процессором и др.);
- управление базой данных (управление данными электронной таблицы как базой данных с выполнением операций поиска по запросу, сортировки, формирования шаблонов и т. п.);
- управление межоконными взаимодействиями (переход от одного окна к другому, управление порядком расположения окон, закрепление и освобождение областей электронных таблиц);
- справочная система.

3. *Пиктографические меню (инструментальные панели).*

Пиктограммы, объединенные в инструментальные панели, предназначены для вызова наиболее часто используемых команд. Количество и состав пиктографических меню определяются пользователем как путем выбора из предлагаемого набора, так и оригинальным формированием. Обычно это можно реализовать в группе команд главного меню, управляющими видом представления элементов рабочего окна.

4. *Строка ввода (редактирования).*

Строка ввода (редактирования) предназначена для ввода и изменения данных в ячейках электронной таблицы. Она содержит имя рабочего листа и адрес активной ячейки, в которой расположен указатель ячеек в виде рамки, окружающей ячейку. Указатель ячеек можно перемещать по рабочему листу с помощью мыши или клавиш управления курсором.

5. *Рабочий лист.*

Рабочий лист отображает собственно электронную таблицу и разбит на ячейки, которые образуют прямоугольный массив и координаты которых определяются путем задания их позиции по вертикали (в столбцах) и по горизонтали (в строках). Столбцы обозначаются буквами латинского алфавита (*A*, *B*, *C*, ..., *Z*, *AA*, *AB*, *AC*, ..., *AZ*, *BA*, *BB*, ...), а строки — числами натурального ряда. Так, *D*14 обозначает ячейку, находящуюся на пересечении столбца *D* и строки 14, а *CD*99 — ячейку, находящуюся на пересечении столбца *CD* и строки 99. Имена столбцов всегда отображаются в верхней строке рабочего листа, а номера строк — на его левой границе.

6. *Линейки прокрутки.*

На экране, как в окне, всегда виден лишь фрагмент активного рабочего листа. Это окно можно передвигать по рабочему листу с помощью ползунков линеек прокрутки, которые расположены в правой (вертикальная линейка прокрутки) и нижней части листа справа (горизонтальная линейка прокрутки). Еще одна маленькая линейка, расположенная слева в нижней части рабочего листа, предназначена для перехода от одного рабочего листа к другому. Каждый рабочий лист имеет корешок с именем. Выбрав и активизировав конкретный корешок, можно продолжить работу на соответствующем этому корешку рабочем листе документа.

7. *Делители окна.*

Они расположены в концах линеек прокрутки и с их помощью можно разделить активный рабочий лист по горизонтали или по вертикали. Это позволяет одновременно увидеть на экране несколько фрагментов рабочего листа и осуществлять их совместную обработку.

8. *Строка сообщений.*

В строке сообщений отображается информация о текущем состоянии таблицы и программы и о результатах выполняемых операций. При выборе какой-либо команды в строке сообщений появляются краткие сведения о ее назначении.

Технология подготовки табличных документов

Ввод и редактирование данных в электронной таблице. После запуска табличного процессора и появления рабочего окна обычно устанавливается режим ввода данных в ячейки таблицы (рабочего листа). Как уже указывалось, одна из ячеек является текущей или активной.

При необходимости редактирования данных в процессе ввода (до нажатия клавиши **Enter**) следует использовать клавиши **Del** и **Backspace**. Если же возникает необходимость изменения данных, уже имеющихся в ячейке, или после нажатия клавиши **Enter**, следует перейти в режим редактирования. Это может быть осуществлено двумя способами: либо нажатием соответствующей функциональной клавиши (обычно **F2**), либо установкой и активизацией указателя мыши на строке ввода.

Помимо редактирования данных на уровне ячейки в электронной таблице реализуется редактирование на уровне объектов таблицы. К объектам таблицы помимо уже упомянутых столбцов, строк и ячеек относятся диапазоны столбцов и строк, блоки ячеек, таблица в целом.

Диапазоном столбцов (*строк*) называется последовательность нескольких подряд идущих столбцов (строк) таблицы. Обычно диапазон обозначается именами (номерами) первого и последнего элементов с

двоеточием между ними (например, **C:F** для диапазона столбцов и **6:8** для диапазона строк).

Блок клеток представляет собой прямоугольный фрагмент таблицы, образованный пересечением нескольких подряд идущих столбцов с несколькими подряд идущими строками. Обозначается блок клеток адресами ячеек, стоящих в верхнем левом и правом нижнем углах прямоугольного фрагмента, с двоеточием между ними (например, **H12:J15**).

Для объектов электронной таблицы определены следующие операции редактирования, объединенные в одну группу: удаление, очистка, вставка, копирование. Операция перемещения фрагмента сводится к последовательному выполнению операций удаления и вставки. Перед выполнением конкретной операции редактирования необходимо определить объект, над которым выполняется действие. По умолчанию таким объектом является текущая ячейка. Остальные объекты должны быть выбраны (выделены). Это обычно выполняется с помощью мыши или клавиатуры.

Операция *очистки* содержимого фрагмента электронной таблицы удаляет данные из его ячеек, оставляя на месте сами ячейки.

В отличие от очистки операция *удаления* приводит не только к очистке содержимого ячеек фрагмента, но и к удалению из электронной таблицы самих ячеек. Но при этом надо указать направление сдвига соседних с удаляемым фрагментом ячеек для заполнения освободившегося места. Обычно указываются направления влево и вверх.

Удаление строк и столбцов (равно как и их диапазонов) приводит к смыканию соответствующих соседних строк и столбцов. При удалении фрагментов электронной таблицы происходит присвоение новых адресов ячейкам по результатам сдвига на место удаленных фрагментов.

Операция *копирования* фрагмента электронной таблицы предполагает указание фрагмента-оригинала и фрагмента-копии. Необходимо иметь в виду, что после выполнения копирования старое содержимое ячеек фрагмента-копии будет уничтожено. При этом должно соблюдаться определенное соответствие между указанными фрагментами.

В самом простом случае имеет место однозначное соответствие вида и размеров фрагмента-оригинала и фрагмента-копии (ячейка в ячейку, строка в строку, столбец в столбец, блок в блок того же размера). В этом случае для фрагмента-оригинала достаточно указать его начальную ячейку (для блока — это ячейка в правом верхнем углу).

При указании в качестве фрагмента-копии компонента электронной таблицы больших размеров необходимо, чтобы его длина и высота были кратны соответствующим размерам фрагмента копии.

Операция *вставки* фрагмента предполагает указание места вставки (ячейки, определяющей начало фрагмента) и направление сдвига ячеек для освобождения места вставляемому компоненту электронной таблицы. Обычно указывают направления вправо и вниз.

Форматирование фрагментов электронной таблицы. Для правильного оформления таблицы в соответствии с требованиями, изложенными в п. 5.1, для отдельных элементов (объектов) таблицы могут быть установлены различные параметры формата.

Формат ячейки (группы ячеек) представляет собой совокупность значений следующих параметров:

- формат представления значений;
- выравнивание значений внутри клетки;
- шрифтовое оформление;
- оформление границ ячейки;
- оформление фона ячейки.

Дополнительно для столбцов и строк таблицы устанавливаются соответственно ширина и высота.

Все указанные параметры форматирования устанавливаются либо с помощью операций из соответствующего меню, либо с помощью соответствующих кнопок-пиктограмм с предшествующим выделением объекта форматирования.

Вычисления в электронной таблице. Каждая ячейка электронной таблицы характеризуется следующими параметрами:

- адрес ячейки;
- содержание ячейки;
- значение ячейки;
- формат ячейки.

Обычно при выполнении операций копирования фрагментов фрагменту-копии передаются все свойства соответствующих ячеек фрагмента-оригинала, но возможна передача только содержания, значения или формата.

В качестве *содержания* ячейки выступают числовые и текстовые константы, а также выражения (формулы).

В качестве *значения* ячейки рассматриваются выводимые на экран представления числовых и текстовых констант, а также результатов вычисления выражений (формул).

Под *выражением* понимается совокупность операндов, соединенных знаками операций. В качестве операндов используются числовые и текстовые константы, адреса ячеек и встроенные функции. При этом числовые и текстовые константы используются непосредственно, вместо адресов ячеек используются значения соответствующих клеток таблицы, а вместо встроенных функций используются возвращаемые ими значения.

Адреса ячеек в роли операндов и аргументов встроенных функций выступают в двух формах: относительной и абсолютной. Относительный адрес указывает на положение адресуемой ячейки относительно той ячейки, в содержании которой он используется и записывается как обычно (имя столбца и номер строки, например **F7**). Абсолютный адрес указывает на точное положение адресуемой ячейки в таблице и записывается со знаком $ перед именем столбца и номером строки (например F7). Возможна абсолютная адресация только столбца или строки ($F7 или F$7). При редактировании объектов электронной таблицы относительные адреса соответствующим образом корректируются, а абсолютные адреса не изменяются.

Состав и назначение встроенных функций в табличных процессорах. Встроенные функции имеют тот же смысл, что и в языках программирования высокого уровня, но в табличных процессорах их набор существенно больше. Существуют следующие группы встроенных функций:

- функции для работы с базами данных и списками;
- функции для работы с датами и временными значениями;
- функции для инженерных расчетов;
- функции проверки свойств и значений;
- логические функции;
- функции для работы со ссылками и массивами;
- математические функции;
- функции для статистических расчетов;
- текстовые функции;
- финансовые функции.

Встроенная функция как операнд выражения записывается в виде

FUNCTION(список аргументов).

Здесь **FUNCTION** представляет собой имя встроенной функции (зарезервированное слово табличного процессора), а список аргументов задается в виде перечня объектов (числовых и текстовых констант, адресов ячеек, диапазонов строк и столбцов, блоков ячеек, имен встроенных функций), разделенных принятым в конкретной операционной среде символом-разделителем.

Для встроенных функций современных табличных процессоров характерны вложенность (задание одной встроенной функции как аргумента другой) и рекурсивность (задание в качестве аргумента встроенной функции имени такой же функции).

Иллюстрации деловой графики на основе данных электронной таблицы. Представление числовых данных в виде таблиц значительно облегчает их восприятие при анализе конкретных ситуаций и принятии управленческих решений. Но простота и наглядность табличной формы представления данных утрачивается по мере увеличения размеров таблиц. Поэтому важную роль играют иллюстрации деловой графики, подготавливаемые на основе табличных данных и существенно упрощающие качественную оценку управленческих ситуаций.

Табличные процессоры предлагают различные виды иллюстраций деловой графики (диаграмм), причем их построение существенным образом облегчено за счет использования *мастеров диаграмм* — встроенных автоматизированных пошаговых процедур, позволяющих соответствующим образом для выбранного типа диаграммы выполнить все необходимые операции оформления ее различных компонентов.

Выбор конкретного вида диаграмм осуществляется на основе содержательного анализа табличных данных и преимущественной ориентации конкретного

вида диаграмм на отображение определенных явлений и процессов, причем в процессе построения конкретной диаграммы возможно уточнение за счет использования тех или иных разновидностей в рамках отдельного вида.

Гистограмма показывает изменение данных за определенный период времени и иллюстрирует соотношение отдельных значений данных. Категории располагаются по горизонтали, а значения по вертикали. Таким образом, уделяется большее внимание изменениям во времени.

Гистограмма с накоплением демонстрирует вклад отдельных элементов в общую сумму.

Линейчатая диаграмма отражает соотношение отдельных компонентов. Категории расположены по горизонтали, а значения по вертикали. Таким образом, уделяется большее внимание сопоставлению значений и меньшее — изменениям во времени. *Линейчатая диаграмма с накоплением* показывает вклад отдельных элементов в общую сумму.

График отражает тенденции изменения данных за равные промежутки времени.

Круговая диаграмма показывает как абсолютную величину каждого элемента ряда данных, так и его вклад в общую сумму. На круговой диаграмме может быть представлен только один ряд данных. Такую диаграмму рекомендуется использовать, когда необходимо подчеркнуть какой-либо значительный элемент.

Точечная диаграмма отображает взаимосвязь между числовыми значениями в нескольких рядах и представляет две группы чисел в виде одного ряда точек в координатах *x* и *y*. Эта диаграмма отображает нечетные интервалы (или кластеры) данных и часто используется для представления данных научного характера. При подготовке данных следует расположить в одной строке или столбце все значения переменной *x*, а соответ-

ствующие значения *y* — в смежных строках или столбцах.

Диаграмма с областями подчеркивает величину изменения в течение определенного периода времени, показывая сумму введенных значений. Она также отображает вклад отдельных значений в общую сумму.

Как и круговая диаграмма, *кольцевая диаграмма* показывает вклад каждого элемента в общую сумму, но в отличие от круговой диаграммы она может содержать несколько рядов данных. Каждое кольцо в кольцевой диаграмме представляет отдельный ряд данных.

В *лепестковой диаграмме* каждая категория имеет собственную ось координат, исходящую из начала координат. Линиями соединяются все значения из определенной серии. Лепестковая диаграмма позволяет сравнить общие значения из нескольких наборов данных.

Поверхностная диаграмма используется для поиска наилучшего сочетания двух наборов данных. Как на топографической карте, области с одним значением выделяются одинаковым узором и цветом. *Пузырьковая диаграмма* является разновидностью точечной диаграммы. Размер маркера данных указывает значение третьей переменной. При подготовке данных следует расположить в одних строке или столбце все значения переменной *x*, а соответствующие значения *y* — в смежных строках или столбцах.

Биржевая диаграмма часто используется для демонстрации цен на акции. Этот тип диаграммы также может быть использован для научных данных, например, для определения изменения температуры. Для построения этой и других биржевых диаграмм необходимо правильно организовать данные. Биржевая диаграмма для наборов из трех и пяти значений может иметь две оси: одна для столбцов, представляющих интервал колебаний, другая для цен на акции.

Маркеры данных в виде *конуса*, *цилиндра* и *пирамиды* могут придавать впечатляющий вид объемным гистограммам и объемным линейчатым диаграммам.

Технология использования систем управления базами данных

Система управления базами данных (СУБД)

Системой управления базами данных (*СУБД*) называют программную систему, предназначенную для создания на ЭВМ общей базы данных для множества приложений; поддержания ее в актуальном состоянии и обеспечения эффективного доступа пользователей к содержащимся в ней данным в рамках предоставленных им полномочий. СУБД предназначена, таким образом, для централизованного управления базой данных (БД) как информационным ресурсом в интересах всей совокупности пользователей.

В настоящее время практически невозможно представить информационную поддержку без применения СУБД. Однако существующий сегодня уровень возможностей программных продуктов данного направления был достигнут не сразу: эволюция СУБД прошла путь от систем, опиравшихся на иерархическую и сетевую модель данных, до систем так называемого третьего поколения, для которых характерны идеи объектно-ориентированного подхода.

СУБД первого поколения имели ряд существенных недостатков: отсутствие стандарта внешних интерфейсов и обеспечиваемости переносимости прикладных программ. Однако эти СУБД оказались весьма долговечны: разработанное на их основе программное обеспечение используется и сегодня.

Разработка Е. Коддом реляционной теории подтолкнула к созданию следующего класса СУБД. Осо-

бенностями второго поколения являются применение реляционной модели данных и развитый язык запросов SQL. Простота и гибкость модели данных позволили стать ей доминирующей и занять лидирующие позиции на соответствующем секторе рынка.

Многие разработчики сегодня выделяют ряд негативных моментов в реляционной модели, среди которых можно выделить невозможность представления и манипулирования данными сложной структуры (тексты, пространственные данные). Это заставляет вести работы по совершенствованию систем второго поколения или создания новой модели данных. Для СУБД третьего поколения характерны использование предложений, касающихся управления объектами и правилами, управления распределенными данными, языков программирования четвертого поколения (4GL), технологии тиражирования данных и других достижений в области обработки данных.

Состав и назначение языковых средств СУБД

Для работы с БД используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, обеспечивающий базовый пользовательский интерфейс с базами данных.

Реализация языковых средств пользователей для работы с БД может быть осуществлена различными способами:

- для высококвалифицированных пользователей языковые средства представляются в их явной синтаксической форме;

- в других случаях функции языков могут быть доступны косвенным образом, когда они реализуются в форме различного рода меню, диалоговых сценариев или заполняемых пользователем таблиц. По таким входным данным интерфейсные средства формируют адекватные синтаксические конструкции языка интерфейса и передают их на исполнение или включают в генерируемый код языка приложения.

Языковые средства используются для выполнения двух основных функций:

- для описания представления базы данных на управляемых уровнях архитектуры системы;
- для инициирования выполнения операций манипулирования данными.

Первая из этих функций обеспечивается *языком описания данных* (*ЯОД*). Его часто называют языком определения данных. Описание данных средствами ЯОД называют *схемой базы данных*. Оно включает описание логической структуры данных и налагаемых на нее ограничений целостности в рамках тех правил, которые регламентированы моделью данных используемой СУБД. Помимо указанных функций, ЯОД некоторых СУБД обеспечивает возможности задания ограничения доступа к данным или полномочий пользователей.

Язык манипулирования данными (*ЯМД*) позволяет запрашивать предусмотренные в системе операции над данными из базы данных, т. е. содержит набор операторов манипулирования данными, позволяющий заносить данные, удалять, модифицировать или выбирать их. Аналогично ЯОД, ЯМД не обязательно выступает в качестве синтаксически самостоятельного языка СУБД.

В настоящее время имеются многочисленные примеры языков СУБД, объединяющих возможности опи-

сания данных и манипулирования данными в единых синтаксических рамках. Более того, в современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с базой данных, начиная от ее создания и обеспечивающий базовый пользовательский интерфейс с базами данных. Наиболее популярным и стандартным для реляционных СУБД является язык SQL (Structured Query Language), разработанный фирмой IBM.

Некоторые СУБД располагают такими языками, которые не только реализуют функции определения и манипулирования данными, но и обладают управляющими структурами и другими средствами, свойственными традиционным языкам программирования. Благодаря этому они могут использоваться как функционально полное средство для создания прикладных программ и для формулировки запросов пользователей к БД. Такие языки называют *автономными* (*языки запросов*).

В настоящее время акцент разработки приложений все более переносится с профессиональных программистов на конечных пользователей. Поэтому в составе СУБД все чаще входят *языки конечных пользователей*, которые относят к классу автономных непроцедурных языков высокого уровня. Эти языки позволяют осуществлять выборку данных, формировать отчеты, разрабатывать новые приложения и запрашивать выполнение уже разработанных. В некоторых языках такого типа обеспечены также возможности обновления баз данных и реализация достаточно сложной их обработки.

Гипертекстовая технология и технология гипермедиа

Основные понятия

Гипертекст (нелинейный текст) — это организация текстовой информации, при которой текст представляет собой множество фрагментов с явно указанными ассоциативными связями между этими фрагментами. Основная идея гипертекстовых технологий состоит в том, что поиск информации происходит с учетом множества взаимосвязей, имеющихся между документами, а значит более эффективно, чем при традиционных методах поиска.

Формально гипертекст можно представить в виде сети или графа, где узлами являются фрагменты текста, а дуги отображают отношения, связывающие эти фрагменты. Доступ к информации осуществляется не путем последовательного просмотра текста, как в обычных информационно-поисковых системах, а путем движения от одного фрагмента к другому.

В самом общем виде взаимодействие пользователя с гипертекстовой системой заключается в следующем. Пользователь читает на экране компьютера некоторый текст и имеет возможность выполнять ряд определенных в системе действий в зависимости от того, какие у него возникают ассоциации от чтения текста на экране.

Гипертекст можно рассматривать как своеобразную базу данных, организованной в виде открытой, свободно наращиваемой и изменяемой сети, узлы которой (линейные тексты) соединяются самим пользователем. От обычной базы данных гипертекст отличается прежде всего тем, что в нем отсутствуют априорно заданные ограничения на характер связей (как, например, в иерархических структурах).

Элементы гипертекста (текстовые фрагменты) называются узлами. Узлы, между которыми возможен переход, считаются смежными, а сама возможность перехода называется «связь». Совокупность смежных узлов образует «окрестность» данного узла.

Последовательно соединенные связями узлы образует «цепь». Расстояние между узлами, что соответствует «близости» или «неблизости» их содержания, равно минимальному количеству промежуточных узлов.

В общем случае, в качестве узла могут выступать: слово; словосочетание; предложение; абзац; параграф; документ; собрание документов, относящихся к одной теме; отдельные сообщения и т. п.

Характер связей между узлами может быть различным. Переход может осуществляться между: текстом и комментарием к нему, между разными редакциями текста, между текстом и его возможными продолжениями, между текстами, отвечающими или возражающими друг другу, между текстами, пересекающимися по содержанию и т. д.

Создание гипертекста состоит прежде всего в формировании системы переходов от узла к узлу (системы ссылок). В зависимости от типа гипертекстовой системы такая система может задаваться как разработчиками, так и пользователем в процессе работы с гипертекстом.

Движение в гипертекстовой сети, совершаемое в процессе чтения гипертекста, называется «навигацией».

Если гиперсеть имеет сложную, разветвленную структуру, возникает проблема ориентации пользователя, т. е. определения в каком месте сети в данный момент он находится. Проблема ориентации присутствует и при работе с традиционным линейным текстом большого объема, но в этом случае пользователь имеет

только два направления поиска — «выше» или «ниже». Гипертекст предлагает больше возможностей в выборе направлений движения, поэтому в этом смысле работать с гипертекстом сложнее. Многие гипертекстовые системы облегчают проблему ориентации в гипертексте, предоставляя наглядное изображение структуры связей. В некоторых современных гипертекстовых системах существует возможность запоминания направлений поиска пользователя в процессе навигации.

Гипертекстовая технология реализуется в конкретной гипертекстовой системе, которая состоит из двух частей: гипертекста (базы данных) и гипертекстовой оболочки. Гипертекстовая оболочка осуществляет следующие основные функции:

- поддержка ссылочных связей;
- создание, редактирование и наращивание гипертекста;
- прямой доступ;
- поддержка ссылочных связей;
- просмотр;
- выделение виртуальных структур.

Поддержка ссылочных связей позволяет поддерживать ранее зафиксированные связи между узлами сети.

Функция создания, редактирования и наращивания гипертекста принципиально отличает технологию гипертекста от технологии баз данных (в которых концептуальная схема данных заранее задана) позволяет вводить новые узлы, редактировать содержание узлов, устанавливать связи между узлами.

Прямой доступ позволяет осуществлять прямой доступ к узлам сети по их именам.

Просмотр (browsing — браузинг) — операция, характерная только для гипертекста. Означает поиск информации посредством просмотра гипертекстовой сети, при этом возможно запоминание пути следования

с тем, чтобы при последующем аналогичном запросе поиск происходил по зафиксированному пути следования.

Реальные гипертекстовые системы в зависимости от специализации могут обладать различным набором вышеперечисленных функций.

Гипертекстовые технологии широко используются в различных прикладных системах:

- в настольных издательских системах — для создания документов большого объема со свойствами гипертекста (т. е. с системой ссылок);
- в системах управления документами (СУД) — например, для сведения в один итоговый документ информации, содержащейся в разнородных документах;
- в системах подготовки электронных документов, позволяющих составлять гипертекстовые документы с возможностью осуществления навигации.

Одним из перспективных направлений развития гипертекстовых систем является *технология гипермедиа* — соединение технологии гипертекста и технологии мультимедиа (интеграция текста, графики, звука, видео). Примерами разработки гипермедийных приложений являются различные электронные издания — справочники, энциклопедии, обучающие программы.

Применение гипертекстовой технологии в Internet

Гипертекстовые технологии нашли широкое применение и при организации хранения и представления информации в сети Internet, например в сервисе World Wide-Web (WWW). В состав Web-системы входят следующие составляющие:

- язык гипертекстовой разметки документов HTML (Hyper Text Markup Language);

- универсальный способ адресации ресурсов в сети URL (Universal Resource Locator);
- протокол обмена данными (гипертекстовой информацией) HTTP (Hyper Text Transfer Protocol);
- средства просмотра Web-страниц (браузеры).

Язык HTML — это средство для формирования гипертекстовых документов. Гипертекстовые ссылки встроены в текст документа и хранятся как его часть. Благодаря этому языку можно не только формировать гипертекстовые документы, но и осуществлять связь текста и изображения с документами, расположенными на другом сервере Web.

Универсальный способ адресации применяется для организации гипертекстовых ссылок и обеспечивает доступ к распределенным ресурсам сети. Адрес URL состоит из трех элементов: используемого протокола доступа, логического имени сервера, имени файла. Например, сервер Государственной публичной научно-технической библиотеки России имеет адрес: http://gpntb.ippi.ras.ru/.

Протокол обмена данными служит для установления связи с документами формата HTML независимо от его местонахождения

В настоящее время гипертекстовые технологии развиваются в нескольких направлениях. Одно из них концентрируется на представлении в узлах гипертекста разнородной, но семантически связанной информации — текста, рисунков, графики, фотографий, видео, звука.

Важным направлением развития гипертекстовых технологий является аналитическая обработка информации. Например, смысловое упорядочивание документов, обеспечивающих решение многоэтапной задачи или разработку сложных проектов.

Наиболее перспективным направлением является технология единой организации всех информацион-

ных ресурсов, распределенных в сетях различных типов (локальных, корпоративных, глобальных) и прежде всего Web-технология.

Компьютерные коммуникации

Локальные и крупномасштабные вычислительные сети

Глобальная (крупномасштабная) вычислительная сеть WAN (Wide Area Network) представляет собой множество географически удаленных друг от друга компьютеров, совместное взаимодействие которых обеспечивается коммуникационной сетью передачи данных и сетевым программным обеспечением. Основу WAN составляют мощные вычислительные системы, являющиеся различного рода серверами, а также специализированные компьютеры, выполняющие функции коммуникационных узлов. Пользователи персональных компьютеров становятся абонентами сети посредством подключения своих ПК именно к этим вычислительным или коммуникационным узлам.

WAN может носить как ведомственный, так общенациональный и даже интернациональный характер. Общими признаками WAN являются, во-первых, значительный масштаб сети (как по площади, так и по числу узлов), а во-вторых, неоднородность сети (т. е. различный тип архитектуры и программного обеспечения узлов), что и определяет дополнительные сложности организации взаимодействия сетевых элементов. В частности, масштаб WAN требует решения проблем общей адресации сетевых узлов и маршрутизации передачи данных между ними.

Internet — вычислительная сеть, объединяющая миллионы компьютеров по всему миру, фактически

является конгломератом многих глобальных, региональных, университетских и учрежденческих сетей, а также сетей коммерческих провайдеров, которые предоставляют доступ к Internet индивидуальным клиентам. В Internet нет центрального управляющего органа, а, следовательно, выход из строя любого из существующих узлов или появление новых узлов не оказывают никакого влияния на общую работоспособность сети. Однако архитектура коммуникационной системы Internet имеет вполне определенный иерархический характер. В этой иерархической архитектуре ограниченный набор дорогостоящих магистральных каналов с высокой пропускной способностью, составляющих так называемую опорную или базовую сеть, соединяет между собой сети со средней пропускной способностью, к которым в свою очередь подключаются отдельные организации со своими клиентами.

Локальные вычислительные сети ЛВС или LAN (Local Area Network), обеспечивая взаимодействие небольшого количества однородных компьютеров на небольшой территории, имеют по сравнению с WAN менее развитую архитектуру и используют более простые методы управления взаимодействием узлов сети. При этом небольшие расстояния между узлами сети и простота управления системой связи позволяют обеспечивать в LAN более высокие скорости передачи данных.

Термин *internet* (со строчной буквы) обозначает локальную или региональную сетевую среду, объединенную с помощью средств маршрутизации, которые управляют пересылкой данных на основе общего пространства логических адресов узлов.

Термин *intranet* обозначает изолированное пределами одной организации обеспечение сетевого доступа к общим данным при поддержке их разделения между отдельными подразделениями. Часто под intranet под-

разумевается обеспечение основных сетевых сервисов Internet в пределах корпоративной ЛВС.

Термин *extranet* обозначает сетевое объединение нескольких организаций, обеспечивающее прямой доступ к приложениям каждой из сторон. Первоначально такое объединение осуществлялось за счет выделенных сетевых соединений. По мере развития в Internet средств ведения электронной коммерции и стандартов шифрования данных необходимость использования выделенных соединений, по всей видимости, полностью исчезнет.

Основные службы Internet

Пользователи, подключенные к Internet, получают доступ ко всем ресурсам сети. Они могут с помощью программных средств *telnet*, *rlogin* и т. п. осуществить регистрацию и выполнить свою работу на одном из удаленных многопользовательских компьютеров; совместно с другими пользователями объединять свои файловые системы в рамках распределенной в пространстве сетевой файловой системы NFS (Network File System) или воспользоваться услугами доступной почти в любой точке земного шара электронной почты *E-mail*, которая практически по всем параметрам превосходит обыкновенную почту.

В Internet существует множество так называемых *ftp-серверов*, на которых хранится огромное количество файлов. Пользователь, соединившись одним из таких сервером с помощью сетевой службы FTP (File Transfer Protocol), получает возможность поиска на сервере и переноса на собственный компьютер необходимой ему информации. Правда, иногда, для того чтобы копировать файлы, необходимо иметь пользовательский бюджет на данном сервере. Однако при этом многие ftp-серверы позволяют регистрироваться под пользовательским именем *anonymous* и с адресом электронной

почты в качестве пароля (такие серверы называются анонимными ftp-серверами).

Для облегчения поиска необходимой информации в Internet существует отдельная *сетевая служба Archie*. Данная служба обеспечивает поиск по ключевым словам в специальной регулярно обновляемой базе данных о файлах, доступных по анонимному ftp.

Служба WAIS (Wide Area Information Server) аналогична Archie, однако позволяет проводить более глубокий поиск, не только по именам и общим характеристикам файлов, но и по их содержанию.

Сервисная система *Gopher* связывает все три вышеназванные службы воедино. Средства поиска Gopher хорошо совмещаются с Archie и WAIS, а средства ее пользовательского интерфейса позволяют просматривать и копировать документы, найденные в результате поиска.

Для представления хранимой в Internet информации в удобной для пользователя форме существует специальная *сетевая служба WWW* (World Wide Web), которая представляет собой своего рода распределенную по множеству узлов базу различного рода данных, построенную на гипертекстовой технологии. Для поиска в этой базе используются различные поисковые серверы, например Rambler, Lycos, Yahoo и др.

Помимо названных сетевых служб в Internet существуют и другие службы, в частности IRC и ICQ, которые обеспечивают возможность интерактивного общения удаленных пользователей сети. С помощью IRC (Internet Relay Chat) множество пользователей могут заходить на так называемые «каналы» (комнаты, виртуальные места, как правило, имеющие тематическую направленность), чтобы «поговорить» с группой людей или с конкретным человеком. Служба ICQ (I Seek You) — очень популярный в последнее время Internet-пейджер, позволяющий в любое время узнать, нахо-

дится ли некоторый пользователь в сети, «поговорить» с ним, обменяться файлами и т. д.

Воспользоваться услугами всех перечисленных выше сетевых служб можно при наличии у пользователя специальной программы-клиента. Отметим, что некоторые из таких программ-клиентов носят интегральный характер, обеспечивая взаимодействие пользователя сразу с несколькими сетевыми службами. Например, большинство Web-браузеров позволяет работать не только с WWW, но и с FTP и даже с некоторыми другими сетевыми службами.

Поиск информации в Internet

Internet представляет собой огромное хранилище распределенной информации различных форматов и видов:

- Web-страницы;
- онлайновые электронные библиотеки;
- виртуальные музеи;
- каталоги по продуктам и услугам;
- открытая правительственная информация;
- научно-исследовательские публикации;
- документы различных сервисов Internet: Gopher, FTP, и др.;
- коммерческая и финансовая информация.

Одна из основных проблем пользователя Internet — это эффективный поиск информации. Очевидно, что актуальность этой проблемы будет возрастать, так как объем документальной информации в Internet возрастает экспоненциально.

Основным инструментом поиска в Internet являются поисковые системы, которых в настоящее время насчитывается около 200. Существующие поисковые системы Internet можно классифицировать по нескольким критериям.

1. Объем поискового индекса.

Поисковые системы периодически просматривают узлы Internet и формируют постоянно обновляемые индексы документов. Из-за экспоненциального расширения Всемирной сети исчерпывающее индексирование всего содержимого Web и создание одного огромного индекса практически невозможно. В настоящее время даже лучшие поисковые системы индексируют не более трети всего содержимого сети Internet.

2. Метод выбора серверов для просмотра (опроса).

Генерация поискового индекса требует систематического обхода Web-узлов и определения местонахождения каждого из документов. Структура Web аналогична структуре ориентированного графа, поэтому здесь может использоваться любой из алгоритмов обхода графа.

3. Используемые поисковые технологии.

По этому критерию поисковые системы можно разбить на 4 категории.

Тематические каталоги. Технология предусматривает обработку документов и отнесение их к одной из нескольких категорий, перечень которых заранее задан. Фактически — это индексирование на основе классификации. Индексирование может проводиться либо автоматически, либо вручную с помощью специалистов, которые просматривают популярные Web-узлы и составляют краткое описание документов-резюме (ключевые слова, аннотация, реферат).

Специализированные каталоги или справочники. В этом случае каталоги создаются по отдельным отраслям и темам. Например, каталог по новостям, каталог по городам, каталог по адресам электронной почты и т. п.

Поисковые машины (самое развитое средство поиска). Технология реализуется индексными средствами поиска. При этом автоматически индексируются со-

держание всех текстов, расположенных на опрашиваемых серверах.

Средства мета поиска. В данной технологии запрос одновременно осуществляется несколькими поисковыми системами, а результат поиска объединяется в общий, упорядоченный по степени релевантности список. Поскольку каждая система обрабатывает только часть узлов сети, это позволяет значительно расширить базу поиска. К этому классу можно также отнести так называемые «персональные программы поиска», которые позволяют формировать свои собственные инструменты мета поиска (например, автоматически опрашивать часто посещаемые узлы).

Содержание

ДОМАШНИЙ РЕПЕТИТОР

ГОТОВИМСЯ К ЭКЗАМЕНУ ПО ИНФОРМАТИКЕ

Соответствует школьной программе «Информатика»

Краткая теория и решение задач

Типовые алгоритмы и нестандартные задачи.

Начальные сведения о современных информационных технологиях

ISBN 5-8112-0037-4