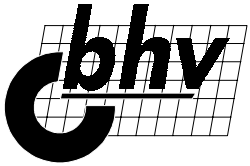
Юлий Кетков
Александр Кетков

САМОУЧИТЕЛЬ
Практика программирования: Бейсик, Си, Паскаль



Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Содержится более 130 готовых к исполнению программ, большинство из которых представлено на трех алгоритмических языках — Бейсике, Си и Паскале. Все разделы предваряются описанием соответствующих конструкций каждого алгоритмического языка. При этом особое внимание обращается на общность языковых средств рассматриваемых систем программирования — QBasic, Turbo C (Borland C++) и Turbo Pascal. Текстам программ предшествуют советы по их разработке с учетом специфики того или иного алгоритмического языка и описание наиболее характерных особенностей.

Для учащихся старших классов и студентов вузов,
а также для преподавателей информатики

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Редактор	*Ольга Михайлова*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Кетков Ю. Л., Кетков А. Ю.
Практика программирования: Бейсик, Си, Паскаль. Самоучитель. — СПб.: БХВ-Петербург, 2001. — 480 с.: ил.

ISBN 5-94157-104-6

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.07.01.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 38,7.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9-я линия, 12.

Содержание

Предисловие

Почему была написана эта книга?

Сегодня грех жаловаться на недостаток литературы по компьютерной тематике. Однако полки в специализированных отделах книжных магазинов заполнены, в основном, многочисленными руководствами, обещающими в немыслимо короткие сроки обучить пользователя навыкам работы с наиболее популярными программными продуктами. На фоне этого довольно поверхностного изобилия не так часто встречаются хорошие книги, посвященные глубокому изучению алгоритмических языков и методам их использования для решения различных задач.

Столь же безрадостная участь постигла и серию книг по практике программирования. Во-первых, таких книг просто мало. Во-вторых, набор рассмотренных в них программ обычно относится к двум противоположным полюсам. Это либо тексты программ, занимающих 5—10 строк и демонстрирующих самые поверхностные возможности того или иного алгоритмического языка, либо задачи повышенной сложности, входившие в программы международных олимпиад по информатике, где наиболее важным моментом является выбор оптимального алгоритма, а вопросы конструирования, оформления и тестирования программ отходят на второй план или просто не рассматриваются. Наконец, нам кажется абсолютно непродуктивной идея многочисленных курсов по информатике, базирующихся на последовательном изучении двух-трех универсальных языков программирования. Такие алгоритмические языки имеют слишком много общего и просто нерационально отводить дефицитные учебные часы на изучение каждого из них с самого начала по стандартной схеме. Параллельное изучение эквивалентных конструкций в разных алгоритмических языках позволяет выработать навыки автоматического преобразования программ, что может оказаться полезным в практической работе.

Для кого написана эта книга?

Для тех, кто начинает изучение основ информатики и пытается освоить технику программирования на примерах сравнительно несложных задач. Для их понимания вполне достаточно элементов математики, изучаемых в средней школе.

Для тех, кто хочет научиться отличать хорошую программу от плохой и не собирается ограничивать свои познания техникой программирования. Последнее подразумевает знание допустимых типов данных и их внутреннего представления в памяти ЭВМ, умение использовать конструкции алгоритмического языка, функциональные возможности системы программирования и операционной системы.

Для тех, кто владеет основами программирования на одном из алгоритмических языков и хочет познакомиться со спецификой других достаточно распространенных алгоритмических языков. Современное общее образование вряд ли может считаться полноценным без изучения хотя бы одного иностранного языка. Точно так же профессиональное образование программиста не может ограничиваться рамками единственного алгоритмического языка. Мы постарались донести до читателей идею о том, что все достаточно универсальные алгоритмические языки очень похожи друг на друга и хорошее знание одного из них позволяет сравнительно просто разобраться с изобразительными средствами другого.

Для начинающих преподавателей информатики, которые хотели бы быстро пополнить арсенал своих учебных программ. Мы были бы крайне признательны вам за любые предложения по совершенствованию предлагаемых программ и расширение состава полезных задач.

Что такое "хорошая" программа?

Если отвлечься от конкретного содержания той или иной задачи, то основные этапы ее решения с помощью компьютера, преобразующего исходные данные в выходные, приведены в таблице ниже.

Номер этапа	Содержание	Исполнитель
1	Формулировка задачи	Человек
2	Выбор алгоритма	Человек
3	Составление исходной программы на алгоритмическом языке	Человек
4	Перевод исходной программы в коды машинных команд	Компьютер
5	Исполнение машинной программы	Компьютер

Эта схема достаточно условна. В ней скрыты довольно важные моменты, связанные с использованием готовых к употреблению библиотечных программ, с устранением синтаксических и алгоритмических ошибок в тексте исходной программы. В некоторых системах программирования (к ним, в частности, относится и QBasic) этапы 4 и 5 совмещены.

Каждому из этапов присущи свои особенности. Формулировка задачи должна исключать какую-либо неопределенность в задании исходных данных и устанавливать область их допустимых значений. Состав исходной информации должен быть достаточен для решения поставленной задачи. Так, например, нельзя построить треугольник, зная только два его параметра. Задание трех его углов тоже не позволяет найти однозначное решение. Алгоритм, как правило, должен приводить к решению задачи за конечное чис-

ло шагов или предусматривать прекращение бесконечных циклов при выполнении определенных условий. Программа на алгоритмическом языке должна иметь четко выраженную структуру и быть понятной не только ее автору. В случае необходимости она должна допускать участие человека в процессе принятия решений. Машинная программа должна располагать удобным интерфейсом и предоставлять в распоряжение пользователя интуитивно понятные средства ввода, управления вычислительным процессом, визуализации промежуточных и окончательных результатов. Это — далеко не полный перечень требований, выработанных многолетней практикой. Но остановимся чуть подробнее на деталях, связанных с организацией исходной программы на алгоритмическом языке.

Во-первых, программа должна выполнять свое главное функциональное назначение — правильно решать поставленную задачу при любом допустимом наборе исходных данных. Без этого любая программа теряет свой смысл.

Во-вторых, программа должна быть, по возможности, эффективной и не тратить на решение задачи лишнее время и ресурсы компьютера. Это особенно важно, когда предполагается многократное использование программы или ее включение в состав более сложного программного комплекса. Конечно, эффективность программы в первую очередь зависит от выбранного алгоритма. Но и реализация последнего может внести свою лепту. Программа может быть идеальной с точки зрения использования конструкций алгоритмического языка, но далеко не самой эффективной из-за неудачного алгоритма. Представим себе, что потребовалось сложить натуральные числа от k1 до k2. Такая процедура реализуется простым циклом, например, на Паскале:

```
s:=0;
for j:=k1 to k2 do s:=s+j;
```

Однако сумму членов арифметической прогрессии можно найти вспомнив или элементарно получив нужную формулу. Выпишем друг под другом ее элементы в прямом и обратном порядках:

k1	k1+1	k1+2	.....	k1+(k2−k1)
k1+(k2−k1)	k2−1	k2−2	.....	k1

Сумма каждой пары равна (k1+k2), и таких пар (k2−k1+1). Поэтому искомая сумма может быть найдена по формуле

```
s:=(k1+k2)*(k2-k1+1)/2;
```

И для ее нахождения потребуется всего пять операций, независимо от числа слагаемых. Из этого примера можно сделать довольно тривиальный вывод — знание техники программирования является необходимым, но не всегда достаточным фактором для эффективного решения задачи.

В-третьих, программа не должна быть очень замысловатой по своей реализации и не должна допускать модификацию или расширение возможностей другими программистами. Поэтому такие моменты, как простота и наличие полноценного комментария способствуют продлению жизненного цикла программы. Очень вредит простоте программы неумеренное использование оператора безусловного перехода. Приводимый ниже пример программы-

спагетти на языке Бейсик предназначен для поиска максимального среди значений трех переменных — max (a,b,c).

```
10 INPUT A,B,C
20 IF A>B THEN GOTO 80
30 IF B>C THEN GOTO 60
40 PRINT "Наибольшее число = "; C
50 GOTO 100
60 PRINT "Наибольшее число = "; B
70 GOTO 100
80 IF A<C THEN GOTO 40
90 PRINT "Наибольшее число = "; A
100 END
```

Представляете, как разрастется подобный монстр, если количество переменных, среди которых ищется максимум, достигнет 10. Насколько изящнее выглядит следующая программа:

```
10 INPUT A,B,C
20 MAX=A
30 IF MAX<B THEN MAX=B
40 IF MAX<C THEN MAX=C
50 PRINT "Наибольшее число = ";MAX
```

Эта программа значительно короче, и в ней нет ни одного оператора перехода. Кроме того, она допускает простое обобщение на поиск максимума среди элементов массива:

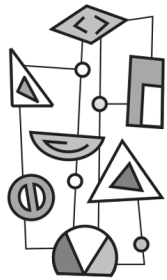
```
MAX=A(1)
FOR K=2 TO N
  IF MAX<A(K) THEN MAX=A(K)
NEXT K
```

Красивую программу можно сравнить со скульптурой, гениальный творец которой отсек от камня все лишнее.

Благодарности

Авторы выражают свою искреннюю признательность всем сотрудникам издательства "БХВ-Петербург", принимавшим участие в подготовке рукописи к печати, за оперативный выход книги в свет. Отдельно хотелось бы поблагодарить менеджера Адаменко Анатолия Николаевича, усилиями которого эта книга нашла своего читателя, и редактора Михайлову Ольгу Викторовну, которая помогла более точно донести до читателя идею книги, за ее внимательное, добросовестное отношение к работе. Большое спасибо.

Глава 1



Три богатыря на рубеже столетий

История алгоритмических языков, насчитывающая немногим менее полувека, успела зафиксировать создание и разрушение своеобразной Вавилонской башни, в стенах которой было погребено более 3000 оригинальных и не очень оригинальных версий универсальных и специализированных языков программирования.

Почти одновременно с появлением первых ЭВМ системные программисты стремились переложить на плечи ЭВМ наиболее рутинную работу, сопровождавшуюся многочисленными ошибками и опиской. Первые элементы автоматизации процесса написания программ были связаны с заменой числовых кодов машинных операций их мнемоническими символьными обозначениями. Например, команда сложения содержимого двух ячеек памяти вместо сугубо числового кода `01 0100 0101 0102` превращалась в более осмысленное действие типа `ADD 0100,0101,0102`. Почти сразу же стало ясно, что использование естественной числовой нумерации ячеек памяти становится неразумной преградой между обозначениями переменных решаемой задачи и их эквивалентами в виде числовых адресов. Почему бы не возложить на специальную программу чисто механическую работу по замене символьных обозначений исходных и промежуточных данных задачи на их машинные адреса? И тогда очередной пункт алгоритма, выражавшийся простой формулой z = x + y, превращался в достаточно наглядную и близкую по смыслу команду `ADD X,Y,Z`. На первом этапе развитие этих идей сдерживало отсутствие устройств ввода/вывода, которые могли бы обрабатывать алфавитно-цифровую информацию. Как только аппаратные средства позволили преодолеть это препятствие, неотъемлемой частью программного обеспечения ЭВМ стали системы, получившие название Автокодов или Ассемблеров.

Следующий бастион, который был взят сторонниками автоматизации тяжелого ручного труда по составлению программ, — укрупнение и стандартизация операций, встречавшихся при решении различных задач. Операции, входившие в состав машинных команд, представляли собой слишком микроскопические действия, на которые приходилось разлагать более понятные человеку процедуры. Так появились алгоритмические языки, каждая строка которых могла быть автоматически трансформирована в цепочку эквивалентных машинных команд. И этот труд по сию пору с честью выполняют

многочисленные "переводчики" (трансляторы, компиляторы) и "исполнители" (интерпретаторы).

К числу первых алгоритмических языков, получивших достаточно широкое распространение, относятся Фортран (FORTRAN — от FORmula TRANslation, "трансляция формул") и Алгол (ALGOL — от ALGOrthmic Language, "алгоритмический язык"). Первый из них родился в недрах фирмы IBM в 1954 г. и активно поддерживался этим наиболее могущественным концерном по производству средств вычислительной техники. В нашей стране он стал широко известен в связи с переходом на выпуск ЭВМ Единой Системы (ЕС ЭВМ), программно совместимых с ранее появившимися моделями IBM/360, IBM/370. История развития Фортрана насчитывает довольно много версий и диалектов, среди которых наиболее известны Fortran-II, Fortran-IV, Fortran-77 и Fortran-90.

Алгол-60 появился как противовес Фортрану в результате объединенных усилий европейских ученых, представлявших, по большей части, академическую и вузовскую науку. Первые сведения о нем были опубликованы в 1958 г., но как стандарт языка он был утвержден в 1960 г. Алгол получил довольно широкое признание в нашей стране. Для него отечественными учеными были разработаны несколько систем программирования — ТА-1М (Вычислительный центр фирмы С. П. Королева), ТА-2М (Институт прикладной математики АН СССР), Альфа (Вычислительный центр СО АН СССР). Однако следующий стандарт языка Алгол-68 практически остался академическим проектом и большого распространения не получил, т. к. в срочном порядке был вытеснен Фортраном.

Несмотря на совершенно разную структуру и наличие несоизмеримых по финансовой мощи спонсоров, оба языка оказали существенное влияние на последующее развитие систем программирования. Так, например, язык Бейсик многое заимствовал из Фортрана, тогда как Паскаль продолжил линию алголоподобных языков. Несколько особняком стоит язык Си, впитавший в себя элементы как низкоуровневого программирования (наследие от языков типа Ассемблер), так и типы данных и синтаксические конструкции алгоритмических языков высокого уровня. Он не только объединил эти две кажущиеся противоположными компоненты, но и внес значительный вклад в развитие объектно-ориентированного программирования.

Далеко немногие алгоритмические языки выдержали испытание временем. Однако языки Бейсик, Си и Паскаль, включенные в нашу книгу, сродни трем былинным богатырям. Их возраст либо приближается к заветному сказочному рубежу в 33 года, либо уже превысил его, что мы и постарались отметить в названии первой главы. На сегодняшний день они являются наиболее популярными как у начинающих, так и у профессиональных программистов.

Самый почтенный среди них — Бейсик, днем рождения которого считается 1 мая 1964 г. В этот день в Дартмутском колледже (США) проявил первые

признаки жизни интерпретатор, созданный студенческим коллективом во главе с профессорами Джоном Кемени (J. G. Kemeny) и Томасом Куртцем (T. E. Kurtz). Своим названием BASIC обязан сокращению фразы Beginner's All-purpose Symbolic Instruction Code, которая дословно переводится как "многоцелевой код (язык) символических инструкций для начинающих".

Бейсик открыл эру диалогового программирования. До него культивировался пакетный режим, при котором бумажные или магнитные носители с программами сдавались дежурному оператору и упорядочивались в соответствии с приоритетами их владельцев. Составленный таким образом пакет программ поступал в ЭВМ на последовательную обработку. При этом достигалась максимальная загрузка оборудования, но каждая программа выполнялась либо до первой автоматически обнаруженной ошибки, либо до истечения лимита заказанного времени. Информация о результатах прохождения программ выдавалась их авторам 2—3 раза в сутки. Поэтому календарные сроки создания программ затягивались на многие месяцы.

Системы коллективного доступа, работавшие в диалоговом режиме, обеспечивали одновременное обслуживание нескольких пользователей, запускавших свои программы с электромеханических или электронных терминалов. Оперативно получив сообщение об очередной ошибке, пользователь имел возможность тут же исправить текст исходной программы и снова выполнить ее.

Диалоговый режим, естественно, был связан с дополнительными накладными расходами на многотерминальное обслуживание. Загрузка ЭВМ при этом снижалась, но оперативность в отладке программ приводила к существенному сокращению календарных сроков их разработки.

Бейсик был одним из первых алгоритмических языков, в составе которого изначально присутствовали операторы общения пользователя с пошагово выполняющейся программой. Одновременно с текстом сообщения об ошибке Бейсик-система сообщала номер строки программы, нарушившей синтаксис языка или приведшей к аварийной ситуации. Первые Бейсик-системы, совмещавшие в себе возможности ввода, редактирования, исполнения и отладки программ, послужили прототипами современных интегрированных сред.

Вторая особенность, привлекающая массового потребителя к Бейсику, кроется в простоте начального освоения и краткости его изобразительных средств. Попробуйте найти хотя бы еще один язык, на котором программа, отвечающая на вопрос, чему равно дважды два, состоит всего из четырех символов:

```
?2*2
```

Аналогичная программа на Паскале содержит, минимум, три строки, а ее длина превышает 20 символов:

```
begin
```

```
(writeln 2 *2);
end.
```

Примерно вдвое большего по числу символов требует ее аналог на Си:

```
#include <stdio.h>
main()
{ printf ("%d",2*2); }
```

Становлению и распространению Бейсика в нашей стране способствовали первые в мире пошаговые компиляторы с этого языка, созданные в 1969 г. для отечественных ЭВМ типа М-20 в Горьковском университете под руководством одного из авторов этой книги.

Второе поколение Бейсик-систем ведет свой отсчет от появления первых ПК на базе 8-разрядных микропроцессоров Intel-8080 и Z-80, для которых в середине 70-х годов был разработан компактный интерпретатор BASIC-80. Именно с него началась карьера самого молодого американского миллиардера Билла Гейтса, основавшего корпорацию Microsoft.

Появление 16-разрядных IBM-совместимых ПК ознаменовалось конкурентной борьбой между компаниями Borland International и Microsoft Corp. Первая из них выпустила на рынок удобную интегрированную среду с компилятором Turbo BASIC, которая быстро привлекла на свою сторону многочисленных любителей Бейсика. Однако более мощная компания, постоянный президент которой не упускает случая прибавить к своей профессии приставку "программист Бейсика", не могла смириться с таким положением. На смену тихоходному интерпретатору GW-BASIC пришла целая серия скоростных систем Quick BASIC, в составе которых наряду с интегрированной средой поставлялись автономные компиляторы и достаточно мощные библиотеки программ. Соревнование Бейсик-систем третьего поколения закончилось поражением фирмы Borland, прекратившей сопровождение своей разработки и передавшей права на Turbo BASIC одному из авторов, вышедшему из состава компании. Одна из последующих его разработок известна под названием Power Basic (Мощный Бейсик).

А фирма Microsoft совершенствовала свое любимое детище и выпустила в конце 80-х годов мощные системы для профессиональных разработок (Professional Development System) BASIC PDS-6 и PDS-7.

Авторы языка Кемени и Куртц спустя 25 лет попытались разработать новую версию под названием True Basic (Истинный Бейсик), однако из-за отсутствия серьезной поддержки со стороны крупных производителей программного обеспечения эта попытка, кроме издания книги и выпуска пробной версии системы, продолжения не получила.

Наконец, четвертое поколение Бейсика мы связываем с появлением в 1991 г. системы визуального программирования Visual Basic и ее последующим совершенствованием сотрудниками Microsoft. На момент выпуска нашей кни-

ги в эксплуатации находится версия VB-6.0. До версии 3.0 включительно Visual Basic был построен по схеме интерпретатора, однако последние его версии позволяют получать готовые к исполнению модули. Кроме полноценной системы программирования, фирма Microsoft с 1993 г. выпускает версии VBA (Visual Basic for Application — система для разработки программируемых процедур в пакете MS Office) и VBScript (ограниченная версия для разработки Web-приложений).

Алгоритмический язык Паскаль был придуман в 1968 г. профессором Института информатики при Швейцарской высшей технической школе Никлаусом Виртом. В 1970 г. под его руководством был разработан первый компилятор, а в следующем году появилась и первая публикация. Свое название язык получил в честь известного французского математика Блеза Паскаля, который в 19-летнем возрасте изобрел первую суммирующую машину.

Новый язык, являясь продолжателем традиций алгоритмического языка Алгол-60, был ориентирован, главным образом, на обучение курсу систематического программирования. В Паскале были представлены наиболее распространенные типы данных и средства для создания пользовательских структур. К числу элементов, способствующих созданию надежных программ, относились блочные конструкции и требование обязательного описания (объявления) всех объектов — типов данных, констант, переменных, меток, функций и процедур.

Заметную роль в разработке стандарта языка Паскаль и совершенствовании его средств ввода/вывода сыграла рабочая группа Британского института стандартов во главе с А. Эддиманом. Британский стандарт был принят в 1982 г., а несколько позднее его утвердила международная организация ISO. Однако к этому времени Н. Вирт, недовольный предложениями рабочей группы, отказался от сотрудничества по совершенствованию Паскаля и переключился на новый проект Модула.

Паскаль довольно долго оставался средством для изучения программирования в университетах, т. к. ни одна серьезная компьютерная фирма его не поддерживала. Перелом в отношении к этому языку наметился в 1984 г., когда молодой француз Филипп Кан привез в США необычайно скоростной компилятор Turbo Pascal для IBM-совместимых ПК и начал торговать им по смехотворно низкой цене 49 долларов 95 центов (для сравнения напомним, что первые версии Бейсик-интепретатора распространялись по цене порядка 500 долларов). Удачная реклама и бросовая цена позволили Кану продать за первый месяц более 3000 копий системы и заложить основы фирмы Borland International. Последующие восемь лет Turbo Pascal оставался наиболее опекаемым продуктом фирмы, которая сумела выпустить девять различных версий. Самая последняя из них (7.0, 1992 г.) включает две системы — Turbo Pascal, функционирующую под управлением MS-DOS, и расширенную версию Borland Pascal, работающую в среде Windows. Несмотря на то, что в январе 1995 г. Ф. Кан покинул пост президента и ушел из ком-

пании, фирма Borland на базе языка Object Pascal выпустила одну из наиболее популярных сред визуального программирования — Delphi и продолжает поддерживать ее версии в современных операционных системах. На сегодня наибольшей популярностью пользуется версия Delphi 5.0.

Язык Си был придуман в 1972 г. сотрудником Bell Laboratories (отделение известной телефонной компании AT&T) Деннисом Ритчи, одним из первых пользователей операционной системы Unix. Задумывался он не как универсальный алгоритмический язык, а, скорее, как инструмент для развития операционной системы и создания новых обслуживающих программ (утилит). Такой подход характерен для большинства системных программистов, разрабатывающих сложные проекты и придумывающих для облегчения своего труда различные сервисные процедуры, макрокоманды и т. п. По завершении разработки, как правило, эти инструментальные наборы предаются забвению или, в лучшем случае, остаются в личных архивах авторов. Язык Си эта участь миновала. Вполне возможно, что его становлению способствовало последующее всемирное признание операционной системы Unix.

Как алгоритмический язык сравнительно низкого уровня, т. е. достаточно близкий к Ассемблеру, Си имел предшественников в лице аналогичных инструментальных средств — языков CPL, BCPL (Basic Combined Programming Language — базовый комбинированный язык программирования) и B. Два первых разрабатывались в конце 70-х годов в Кембриджском университете в качестве машинно-независимых языков для создания трансляторов. Последний был придуман Кеном Томпсоном — сотрудником Bell Laboratories и автором операционной системы Unix. В отличие от своих предшественников Д. Ритчи наряду с машинно-ориентированными типами данных (байт, слово) ввел в состав Си объекты и операторы, присущие универсальным языкам (числовые и символьные переменные, структурные блоки), сохранив при этом элементы, характерные для макроассемблера MACRO-11 (логические операции над битами, сдвиги, работа с адресами и регистрами).

Первым программным продуктом, написанным почти полностью на Си, был компилятор с языка Си в код машинных команд компьютера PDP-11/20 (прототип мини-ЭВМ СМ-4). В 1973 г. Д. Ритчи и К. Томпсон переписали на Си большую часть операционной системы Unix. Из 13 000 машинных команд для PDP-7, на которой появилась первая версия Unix, только 800 пришлось вручную перевести в ассемблер PDP-11. В процессе перевода Unix из однопользовательской операционной системы, ориентированной на работу в конкретной ЭВМ, превратилась в мобильную операционную систему коллективного пользования. Успех этой операции в значительной мере предопределил популярность новой операционной системы и ее базового инструмента — языка Си. В 1976 г. Д. Ритчи и К. Томпсон перенесли Unix с ЭВМ фирмы DEC на компьютеры другой архитектуры (Interdata 8/32), практически ничего не изменив в ядре операционной системы,

написанном на Си. Точно таким же образом система Unix распространялась на десятки машин различных типов.

В 1978 г. появилась первая книга, посвященная описанию Си и технике программирования на этом языке, которая с большим запозданием была переведена на русский язык (Б. Керниган, Д. Ритчи, А. Фьюэр. "Язык программирования Си. Задачи на языке Си". — М.: Финансы и статистика, 1985). От фамилий двух первых авторов произошло сокращенное обозначение первого, никем не утверждавшегося, но принятого всеми программистами стандарта языка Си — K&R.

Дальнейшая работа по совершенствованию языка Си и принятию в 1987 г. первого настоящего стандарта ANSI C была выполнена на общественных началах рабочей группой при Американском национальном институте стандартов. Возглавлял эту работу Лэрри Рослер — сотрудник Bell Labs. Наиболее серьезный вклад в развитие языка Си за последние годы внес еще один представитель той же лаборатории Бьерн Страуструп, который ввел в обращение новые объекты — классы, объединяющие данные, и обрабатывающие их функции. С 1983 г. за расширенной версией языка Си с классами закрепилось название C++.

Первые версии Си подвергались серьезной критике за отсутствие достаточной строгости, приводившей к многочисленным ошибкам из-за работы с неинициализированными переменными, отсутствия контроля за выходом индексов у элементов массивов из установленных пределов, несоответствия типов формальных и фактических параметров функций и т. п. Перед системными программистами Bell Labs эти проблемы остро не стояли, т. к. они пользовались специальной программой Lint, которая проводила тщательный анализ программ, написанных на Си, перед их трансляцией и выполнением.

Для рядовых пользователей ситуация изменилась с появлением интегрированных сред, из которых наибольшую популярность приобрели Турбо-системы фирмы Borland. Первая версия Turbo C, работавшая в среде MS-DOS, была выпущена в 1987 г. Совсем недавно фирма Borland выпустила на рынок версию 5.0, предназначенную для работы под управлением Windows. На базе этого компилятора компания Borland (новое название — Inprise Corp.) разработала серию систем визуального программирования — Borland C++ Builder.

Известны и другие реализации языка Си на IBM-совместимых ПК — Microsoft C, Lattice C, Zortech C, Symantec C. В нашей стране продукция фирмы Borland получила наибольшее распространение, хотя за последнее время намечается тенденция к более широкому использованию среды Visual C++, разработанной фирмой Microsoft. В основном, это диктуется требованиями зарубежных фирм, размещающих свои заказы у отечественных производителей программных продуктов.

Современные версии трех наиболее устоявшихся алгоритмических языков обладают примерно равными функциональными возможностями. Подтверждению именно этого фундаментального свойства универсальных систем программирования посвящена наша книга. Завершая краткий исторический экскурс, остановимся еще раз на мотивах, объясняющих наш выбор языков и систем программирования.

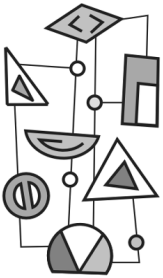
Бейсик — один из наиболее распространенных в мире алгоритмических языков, используемых, как правило, непрофессиональными программистами. По данным [3] из пяти миллионов программирующих пользователей ЭВМ Бейсиком пользуется не менее двух миллионов человек. Большая часть учащихся средних школ начинает освоение информатики с изучения Бейсика, т. к. он доступен на самой допотопной вычислительной технике, которой, к сожалению, еще до сих пор укомплектовано большинство школ России. Несмотря на то, что Бейсик является старожилом по сравнению с Си и Паскалем, в последние годы он переживает вторую молодость в связи с появлением модных систем визуального программирования и массовым использованием Microsoft Office.

Паскаль — алгоритмический язык, созданный специально для изучения программирования и широко используемый в высших учебных заведениях. Вслед за Бейсиком он тоже вступил на тропу визуального программирования, и система Delphi считается одним из лучших инструментальных средств для создания приложений, функционирующих под управлением Windows 95/98/NT.

Си — язык профессионалов, на котором сегодня написано большинство программных систем. Большая часть высших учебных заведений, особенно технического профиля, использует Си в качестве основного языка преподавания программирования. Такие среды визуального программирования, как Borland C++ Builder и Microsoft Visual C++ являются несомненными лидерами в разработках современного программного обеспечения.

В связи с тем, что книга ориентирована на начальное знакомство с перечисленными алгоритмическими языками, в качестве сред программирования нами выбраны самые непритязательные по требованиям к оборудованию системы — QBasic, Turbo C 2.0 или Borland C++ 3.1 и Turbo Pascal 7.0. Все они могут функционировать на самых примитивных IBM-совместимых ПК с оперативной памятью от 1 Мб, занимая на винчестере от 1 до 25 Мб. Кроме того, система QBasic является составной частью операционной системы MS-DOS и представлена там всего двумя небольшими файлами — интерпретатором qbasic.exe и файлом помощи qbasic.hlp.

Глава 2



Работа с числовыми данными

Современные представления о числовых данных базируются на так называемых позиционных системах счисления. Для этих систем характерно основание системы p, степень которого определяет вес цифры a (a = 0, 1, 2, ..., p-1) в зависимости от занимаемой позиции:

$a_k a_{k-1} \ldots a_1 a_0, b_{-1} b_{-2} \ldots b_{-m} = a_k * p^k + a_{k-1} * p^{k-1} + \ldots + a_1 * p^1 + a_0 * p^0 + b_{-1} * p^{-1} + b_{-2} * p^{-2} + \ldots + b_{-m} * p^{-m}$

Главенствующую роль в человеческом общении играет десятичная система счисления (p = 10; a = 0, 1, ..., 9). Однако ЭВМ используют более рациональную двоичную систему (p = 2; a = 0,1). Единственным ее недостатком является большая длина чисел. Количество разрядов (цифр) в двоичном представлении числа примерно в три раза превышает количество десятичных цифр. Для преодоления этого неудобства программисты прибегают к более компактной записи двоичных кодов в виде восьмеричных или шестнадцатеричных чисел. При этом три или четыре смежные двоичные цифры заменяют одной восьмеричной или шестнадцатеричной цифрой. Например:

$192_{10} = 11000000_2$,

$11000000_2 = 300_8$,

$11000000_2 = C0_{16}$.

Внешнее и внутреннее представление числовых данных

Под внешним представлением числовой информации подразумеваются способы записи данных, используемые в текстах программ, при наборе чисел, вводимых в ЭВМ по запросу программы, при отображении результатов на экране дисплея или на принтере. Кроме естественного представления числовых констант в виде целого или вещественного числа, языки программирования допускают различные добавки в начале ("префиксы") или конце ("суффиксы") числа, определяющие способы преобразования и хранения данных в памяти компьютера.

Во входном языке системы QBasic такого рода добавки представлены одним из символов `%`, `!`, `&` или `#`, приписываемым вслед за числом, и одной из двухсимвольных комбинаций `&B`, `&O` или `&H`, располагаемой перед числом:

- `5%` — целое число;
- `5&` — целое число с удвоенной точностью;
- `5` или `5!` — вещественное число;
- `5#` — вещественное число с удвоенной точностью;
- `&B0011100100110111` — двоичное число;
- `&O34467` — восьмеричное число;
- `&H3937` — шестнадцатеричное число.

В Си к аналогичным суффиксам относятся указания об удвоенной длине целых чисел (буквы `L` или `l`), указания о вещественном формате числа, не содержащего в своей записи десятичной точки или десятичного порядка (буква `F` или `f`), указания об использовании беззнакового представления целых чисел (буква `U` или `u`). Префиксы в Си используются для записи восьмеричных (число начинается с `0`) или шестнадцатеричных (числу предшествует одна из комбинаций `0x` или `0X`) констант:

- `5` — короткое целое число со знаком;
- `5U` — короткое целое число без знака;
- `5L` — длинное целое число со знаком;
- `5LU` или `5UL` — длинное целое число без знака;
- `05` — восьмеричное число;
- `0x5` или `0X5` — шестнадцатеричное число;
- `5f` или `5F` — вещественное число со знаком.

В Паскале используется единственный префикс — символ `$`, предшествующий шестнадцатеричному числу:

`$0A`, `$F0A5`, `$FF00140D`.

Наличие в естественной записи числа точки (`3.1415`) или указателя десятичного порядка (`314.159265e-02`) означает, что соответствующее значение представлено в ЭВМ в виде вещественного числа с плавающей запятой.

Машинные форматы представления чисел мы будем называть внутренними, и из приведенных ранее примеров следует, что машинные числа бывают целыми и вещественными. В свою очередь, каждый из этих типов данных допускает несколько представлений, отличающихся диапазоном допустимых чисел. Остановимся более подробно на машинных форматах числовых данных, соответствующих их аналогам в алгоритмических языках.

Самые короткие числа со знаком представлены в памяти ЭВМ одним байтом, в котором может разместиться любое число из диапазона от —128 до

+127. В Си для описания данных такого типа используется спецификатор `char`, а в Паскале — `shortint`.

В одном же байте может быть расположено и самое короткое целое число без знака. В Си для описания таких данных служит спецификатор `unsigned char`, в Паскале — `byte`. Диапазон допустимых данных при этом смещается вправо и равен [0, 255]. QBasic не располагает языковыми средствами для работы с однобайтовыми целыми числами.

Вторая категория целых чисел представлена двухбайтовыми данными. В варианте со знаком они предлагают диапазон от —32 768 до +32 767, в варианте без знака — от 0 до 65 535.

QBasic обеспечивает работу с двухбайтовыми числами со знаком для переменных, описанных как `AS INTEGER` или устаревшее `DEFINT`, и переменных, чьи имена заканчиваются символом `%`. Двухбайтовые целые без знака в QBasic записываются в виде двоичных, восьмеричных или шестнадцатеричных констант. Однако арифметические операции над такими данными иногда выполняются некорректно.

Си использует для описания двухбайтовых целочисленных данных спецификаторы `int` и `unsigned int`. В Паскале для этой же цели служат спецификаторы `integer` и `word`. Оба языка корректно выполняют арифметические операции и с беззнаковыми данными при условии, что результат не выходит за пределы разрешенного диапазона.

Третья категория целых чисел в IBM PC представлена четырехбайтовыми данными. В варианте со знаком они перекрывают диапазон от —2 147 483 648 до +2 147 483 647, в варианте без знака — от 0 до 4 294 967 295.

В QBasic допустимы только данные со знаком и имена переменных такого типа описываются как `AS LONG` или `DEFLNG`. К ним же относятся и переменные, чьи имена заканчиваются символом `&`.

Для описания четырехбайтовых данных целого типа в Си используются спецификаторы `long` (эквивалент `long int`) и `unsigned long`. В Паскале работа с длинными целыми без знака не предусмотрена. Там можно оперировать с четырехбайтовыми данными только типа `longint`.

Следует помнить, что для хранения любых целых чисел со знаком в IBM PC используется дополнительный код, что сказывается на представлении отрицательных чисел:

```
+5          0 0000101          0 000000000000101
-5          1 1111011          1 111111111111011
```

Наиболее часто применяемые типы вещественных чисел представлены короткими (4 байта) и длинными (8 байт) данными.

В QBasic им соответствуют описания `AS SINGLE` (устаревшее — `DEFSNG`) и `AS DOUBLE` (устаревшее — `DEFDBL`). Си использует для этой же цели спецификаторы `float` и `double`, Паскаль — `single` и `double`.

Короткий вещественный формат по модулю обеспечивает представление чисел в диапазоне от 10^{-38} до 10^{+38} примерно с 7—8 значащими цифрами. Для 8-байтового формата диапазон существенно расширяется — от 10^{-308} до 10^{+308}, а количество значащих цифр увеличивается до 15—16.

Сопроцессор IBM PC предлагает еще два формата данных, занимающих соответственно 8 и 10 байт. Первый из них допускает работу с целыми числами из диапазона от -2^{63} до 2^{63-1}. Второй формат перекрывает диапазон данных (по модулю) от 10^{-4932} до 10^{+4932}, сохраняя 19—20 значащих цифр. Расширенный формат вещественных данных можно использовать в программах на Си (`long double`) и на Паскале (`extended`). А формат сверхдлинных целых чисел используется только в Паскале, но там они почему-то отнесены к вещественным данным типа `comp`.

В Паскале по традиции сохранился еще один тип вещественных данных — `real`, занимающий в памяти 6 байт. Его диапазон совпадает с типом `single`, однако количество значащих цифр несколько больше — 10—12. Формат `real` не поддерживается аппаратными средствами IBM PC, поэтому операции над такими данными выполняются с помощью подпрограмм, что существенно увеличивает время решения задачи.

В машинном представлении вещественных данных разного типа на IBM PC не выдержана какая-то общая идеология. Объясняется это, по всей вероятности, разными наслоениями на прежние аппаратные решения, которые принимались при разработке процессоров в разных отделениях фирмы Intel. Поэтому здесь имеют место такие нюансы, как сохранение или не сохранение старшего бита мантиссы, представление мантиссы в виде чисто дробного ($0{,}5 \le m < 1$) или смешанного ($1 \le m < 2$) числа и т. п. Прикладных программистов эти детали мало интересуют, однако при создании специальных системных компонент с точным представлением данных приходится считаться.

Ввод числовой информации

Каких-либо особых проблем с вводом числовой информации в программах не возникает. Но на некоторые детали в том или ином алгоритмическом языке надо обратить внимание.

В QBasic можно натолкнуться на неприятности при вводе нескольких числовых значений в одном операторе. Например:

```
INPUT A,B,C
```

Числовые данные, набираемые пользователем, должны быть разделены пробелом. Если в наборе второго или третьего значения вы допустите ошибку,

то последует сообщение `Redo from start`, которое заставит вас повторить ввод всех трех чисел с самого начала. Некоторые программисты даже предпочитают вводить одним оператором только одно значение. Несоответствие между типом переменной и вводимым значением приводит к ошибке при попытке ввести вещественное значение в целочисленную переменную или при наборе недопустимого символа в числе.

В Си основные неприятности форматного ввода (функция `scanf`) связаны с попыткой указать в списке ввода не адрес переменной, а ее имя:

```
scanf("%d",x);  //правильно было бы scanf("%d",&x);
```

Компилятор TC/BC такую ошибку, к сожалению, не замечает и преобразует имя `x` в какой-то фантастический адрес. Последующую работу программы в этом случае предсказать трудно. Не обращает внимания компилятор Си и на несоответствие между спецификатором формата и типом переменной из списка ввода. Всего этого можно избежать, используя потоковый ввод:

```
cin >> x;
```

Самый тщательный контроль за соответствием между типами вводимых значений и типами соответствующих переменных в списке параметров процедур `read` и `readln` обеспечивает Паскаль. Однако здесь при попытке ввести непредусмотренный символ в числе вслед за сообщением об ошибке задача снимается, если программист не смог предвидеть заранее обход системной реакции на особые ситуации.

Вывод числовых результатов

Наиболее приятный вид имеет числовая информация, организованная по табличному типу — в виде колонок фиксированной ширины, в которых одноименные числовые разряды располагаются друг под другом (единицы — под единицами, десятки — под десятками, сотни — под сотнями и т. д.). При этом, в частности, более рационально используется площадь экрана, что достигается за счет управления форматами выводимых данных.

В QBasic для этой цели служит оператор `PRINT USING`, совмещающий в себе и строку с описанием формата и список выводимых значений:

```
PRINT USING "A=##.### B=### C=#.####^^^^"; A,B,C
```

Аналогичные средства имеются и в Си:

```
printf("A=%6.3f B=%3d C=%10.4e",A,B,C);
```

При выводе в поток, когда к Си-программе подключаются заголовочные файлы `iostream.h` и `iomanip.h`, тоже существует возможность управлять форматом выводимых данных:

```
cout << "A=" << setw(6) << setprecision(5) << A;
```

Несколько скромнее выглядит управление форматом в Паскале:

```
write('A=',A:6:3,"B=',B:3,'C=',C:10);
```

Среди форматных спецификаторов в Си есть дополнительные возможности, отсутствующие в QBasic и Паскале. Например:

```
printf("%4x %6o",x,y);
```

Приведенные здесь спецификаторы позволяют вывести значения целочисленных переменных `x` и `y` соответственно в виде шестнадцатеричного числа с четырьмя цифрами и восьмеричного числа с шестью цифрами.

В Си можно прижимать выводимое число к левой границе отведенного поля (по умолчанию действует правый прижим), печатать знак "+" у положительных чисел, подавлять или разрешать вывод незначащих нулей и др. Подробную информацию о структуре форматного описателя можно найти в файлах помощи.

Задачи, советы и ответы

Задание 2.01. Ввод и вывод целочисленных данных

Ввести 20 целых чисел. Вывести их компактно (в одну или несколько строк), размещая перед каждым числом его порядковый номер. После нажатия какой-либо клавиши вывести числа столбиком, располагая одноименные разряды друг под другом, подвести под столбиком черту и напечатать сумму введенных чисел.

Совет 1

Предположим, что вы будете работать с двухбайтовыми целыми числами. Тогда для размещения самого длинного числа потребуется не менее 7 позиций (число —32 768 занимает 6 позиций плюс пробел между числами). Если перед числом необходимо вывести его номер, то дополнительно потребуются еще, как минимум, 3 позиции (2 под номер из диапазона [1,20] и разделитель между номером и числом, например в виде двоеточия). Таким образом, для компактного вывода на каждое число вместе с его номером можно отвести 10 позиций, что хорошо согласуется с длиной строки на дисплее. Если бы оказалось, что длина строки не кратна ширине колонки, то часть последнего числа была бы автоматически перенесена на следующую строку.

Совет 2

Как организовать ожидание в программе до нажатия какой-либо клавиши? В QBasic для этой цели подойдет системная переменная `INKEY$`. Достаточно присвоить ее значение какой-либо символьной переменной, например `A$`, и организовать бесконечный цикл до тех пор, пока длина значения `A$` перестанет отличаться от нуля:

```
A$=""
M10 : A$=INKEY$ : IF A$="" THEN GOTO M10
```

Можно воспользоваться и другим приемом — включить в программу оператор ввода в какую-либо переменную символьного типа. Такая переменная предпочтительнее числовой, т. к. в нее можно ввести пустое значение, нажав только клавишу <Enter>. Кроме того, набор любого отображаемого символа не приведет к ошибке.

В Си временный приостанов до нажатия какой-либо клавиши организуют с помощью функции `getch`.

В Паскале можно организовать бесконечный цикл, аналогичный приведенному выше варианту для QBasic, с помощью логической функции `KeyPressed`:

```
while not KeyPressed;
```

Аналогом ввода пустого значения `INPUT A$` в QBasic в Паскале является использование процедуры `read/readln` без параметров. В этом случае для проталкивания программы потребуется нажать клавишу <Enter>.

Программа 2_01.bas

```
REM Ввод, вывод и суммирование 20 целых чисел
DIM A(20)
F$="##:###### "
PRINT "Введите 20 чисел, по одному в строке"
FOR I=0 TO 19: INPUT A(I): S=S+A(I): NEXT I
FOR I=0 TO 19: PRINT USING F$;I+1;A(I); : NEXT I
INPUT A$
FOR I=0 TO 19: PRINT USING F$;I+1;A(I): NEXT I
PRINT "----------"
PRINT USING "   ######"; S
END
```

Программа 2_01.c

```
/* Ввод, вывод и суммирование 20 целых чисел */
#include <stdio.h>
#include <conio.h>
  main()
{
  int j,s=0,a[20];
  clrscr();
  printf("Введите 20 чисел, по одному в строке\n");
  for(j=0; j<20; j++)
    { scanf("%d",&a[j]); s+=a[j];  }
```

```
  for(j=0;j<20; j++)
    printf("%2d:%-6d ",j+1,a[j]);
  for(j=0;j<20; j++)
    printf("\n%2d:%6d ",j+1,a[j]);
  printf("\n   ------\n%9d",s);
  getch();
}
```

Программа 2_01.pas

```
program io_numbers;
{ Ввод, вывод и суммирование 20 целых чисел }
uses Crt;
var
  j,s:integer;
  a:array [1..20] of integer;
begin
  s:=0;
  clrscr;
  writeln('Введите 20 чисел, по одному в строке');
  for j:=1 to 20 do
    begin readln(a[j]);  s:=s+a[j]; end;
  for j:=1 to 20 do write(j:2,':',a[j]:6,' ');
  writeln('Нажмите любую клавишу';
  readkey; clrscr;
  for j:=1 to 20 do writeln(j:2,':',a[j]:6,' ');
  writeln('   ------');
  writeln(s:9);
  readln;
end.
```

Задание 2.02. Ввод и вывод вещественных данных

Самостоятельно выполнить задание 2.01 в предположении, что вводимые числа вещественные и имеют две значащие цифры в дробной части.

Задание 2.03. Преобразование десятичного числа в системы с основанием 2, 8, 16

Ввести длинное неотрицательное число. Вывести его в двоичном, восьмеричном и шестнадцатеричном представлении.

Совет 1 (QBasic)

Предлагается воспользоваться стандартными функциями `OCT$` и `HEX$`, преобразующими числовой аргумент в строку с его представлением в соответствующей системе счисления. Для двоичного представления можно распечатать восьмеричные цифры их трехразрядными двоичными эквивалентами.

Совет 2 (Си)

Предлагается воспользоваться библиотечной функцией `ltoa`, преобразующей свой первый аргумент из машинного представления числа в строку, заданную вторым аргументом. Третий аргумент этой функции определяет основание системы счисления, в которую конвертируется исходное число. Вообще говоря, восьмеричный и шестнадцатеричный эквиваленты числа проще вывести, используя спецификатор формата `%o` и `%x`. Для вывода двоичного представления числа можно было бы организовать цикл со сдвигом числа на один разряд влево (`N = N << 1;`), пробой старшей цифры (`N & 0x8000`) и выводом соответствующего символа в зависимости от результата сравнения.

Совет 3 (Паскаль)

В этом языке отсутствуют какие-либо системные функции или процедуры, отмеченные выше. Поэтому единственным средством будет лобовое преобразование путем последовательного деления исходного числа на основание соответствующей системы и запоминание получающихся остатков в некотором массиве. Так как длинное число занимает в памяти ЭВМ 4 байта, максимальный размер массива для хранения цифр числа в соответствующем представлении не должен превышать 32 элемента. Небольшие проблемы могут возникнуть при выводе шестнадцатеричных цифр от `A` до `F`. Из них придется вычитать 10 и добавлять полученную разницу к коду буквы `A`.

Программа 2_03.bas

```
REM Перевод числа в системы с основаниями 2, 8 и 16
CLS
INPUT "Введите положительное число : ",N&
A$=OCT$(N&)
PRINT "В двоичном представлении ";N&;"= ";
FOR k=1 TO LEN(A$)
  B$=MID$(A$,k,1) : ' Выделение очередной восьмеричной цифры
  SELECT CASE B$
    CASE "0": IF k=1 THEN PRINT "";  ELSE PRINT "000";
    CASE "1": IF k=1 THEN PRINT "1";  ELSE PRINT "001";
    CASE "2": IF k=1 THEN PRINT "10";  ELSE PRINT "010";
    CASE "3": IF k=1 THEN PRINT "11";  ELSE PRINT "011";
    CASE "4": PRINT "100";
    CASE "5": PRINT "101";
```

```
    CASE "6": PRINT "111";
    CASE "7": PRINT "111";
  END SELECT
NEXT k
PRINT
PRINT "В восьмеричном представлении ";N&;"= ";OCT$(N&)
PRINT "В шестнадцатеричном представлении ";N&;"=";HEX$(N&)
END
```

Программа 2_03a.bas

```
REM Перевод числа в системы с основаниями 2, 8 и 16
CLS
INPUT "Введите положительное число : ",n&
a$=OCT$(n&) : ' Перевод в восьмеричную систему
IF n&=0 THEN
  PRINT "Это число в любой системе равно 0"
  STOP
END IF
PRINT "В двоичном представлении ";n&;"= ";
  B$=LEFT$(a$,1): : ' Выделение очередной восьмеричной цифры
  SELECT CASE B$
    CASE "0": PRINT "";
    CASE "1": PRINT "1";
    CASE "2": PRINT "10";
    CASE "3": PRINT "11";
    CASE "4": PRINT "100";
    CASE "5": PRINT "101";
    CASE "6": PRINT "111";
    CASE "7": PRINT "111";
END SELECT
FOR K=2 TO LEN(a$)
  B$=MID$(a$,K,1)
  SELECT CASE B$
    CASE "0": PRINT "000";
    CASE "1": PRINT "001";
    CASE "2": PRINT "010";
    CASE "3": PRINT "011";
    CASE "4": PRINT "100";
    CASE "5": PRINT "101";
```

```
    CASE "6": PRINT "111";
    CASE "7": PRINT "111";
  END SELECT
NEXT K
PRINT
PRINT "В восьмеричном представлении ";n&;"= ";OCT$(n&)
PRINT "В шестнадцатеричном представлении ";n&;"= ";HEX$(n&)
END
```

Программа 2_03.c

```
/* Перевод числа в системы счисления с основаниями 2, 8 и 16 */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
   main()
{
  long N;
  char a[33];
  clrscr();
  printf("Введите положительное число : ");
  scanf("%ld",&N);
  if(N==0)
    {
       printf("\nТакое число в любой системе = 0");
       exit(1);
    }
  ltoa(N,a,2);  /* перевод в двоичную систему */
  printf("\nВ двоичном представлении %ld = %s",N,a);
  ltoa(N,a,8);  /* перевод в восьмеричную систему */
  printf("\nВ восьмеричном представлении %ld = %s",N,a);
  ltoa(N,a,16); /* перевод в шестнадцатеричную систему */
  printf("\nВ шестнадцатеричном представлении %ld = %s",N,a);
  getch();
}
```

Программа 2_03.pas

```
program _2_8_16;
{ Перевод числа в системы с основаниями 2, 8 и 16 }
```

```
uses crt;
var
  N1,N:longint;
  a:array [0..31] of byte;
  j,k:byte;
  s:char;
begin
  clrscr;
  write('Введите положительное число : ');
  readln(N);
  if N=0 then
     begin
       writeln('Такое число в любой системе = 0');
       exit;
     end;
  N1:=N;
  for j:=0 to 31 do a[j]:=0;
  while N1<>0 do
    begin
      a[j]:=N1 mod 2; {цикл выделения двоичных цифр}
      dec(j);
      N1:=N1 div 2;
    end;
  write('В двоичном представлении ',N,'=');
  for k:=j+1 to 31 do write(a[k]:1);
  writeln;
  N1:=N;
  for j:=0 to 10 do a[j]:=0;
  while N1<>0 do
    begin
      a[j]:=N1 mod 8; {цикл выделения восьмеричных цифр}
      dec(j);
      N1:=N1 div 8;
    end;
  write('В восьмеричном представлении ',N,'=');
  for k:=j+1 to 10 do write(a[k]:1);
  writeln;
  N1:=N;
  for j:=0 to 7 do a[j]:=0;
```

```
  while N1<>0 do
    begin
      a[j]:=N1 mod 16;
      dec(j);
      N1:=N1 div 16; {цикл выделения шестнадцатеричных цифр}
    end;
  write('В шестнадцатеричном представлении ',N,'=');
  for k:=j+1 to 7 do
    begin
      if a[k]<10 then s:=chr(ord('0')+a[k])
      else s:=chr(ord('A')+a[k]-10);
      write(s);
    end;
  readln;
end.
```

Задание 2.04. Преобразование десятичного числа в систему с основанием `r`

Составить функцию `num_to_str(num, r)`, возвращающую в качестве своего значения строку с представлением натурального числа `num` в системе счисления с основанием `r`. Предполагается, что число `num` в памяти компьютера представлено 4-байтовым целым, а основание `r` принадлежит диапазону [2, 16]. Для обозначения цифр, превосходящих 9, рекомендуется воспользоваться латинскими буквами `A`, `B`, ... , `F`.

Совет 1 (общий)

Так как основание системы может быть любым, необязательно совпадающим с наиболее распространенными вариантами (`r = 2,8,16`), то наиболее универсальным способом перевода остается последовательное деление исходного числа на основание с запоминанием целочисленных частных и остатков. Небольшое осложнение будет вызвано преобразованием двухзначных остатков (при `r > 10`) в односимвольную "цифру". Однако подход к решению этого вопроса уже был продемонстрирован в предыдущей программе на Паскале. Очередной остаток `q` надо сравнить с 10 и, в зависимости от исхода сравнения, либо преобразовывать в символ полученную цифру, либо добавлять разницу `q-10` к коду буквы `A`. Вторая тонкость заключается в том, что предлагаемый способ перевода позволяет получать цифры разложения справа налево, начиная с младшей. Поэтому перед выдачей результата полученную строку придется перевернуть.

Совет 2 (QBasic)

Вы можете выбрать один из двух способов описания типов данных — более современный, хотя и несколько более длинный, с использованием переменных,

описанных как AS LONG, AS STRING, AS INTEGER, или более старомодный с добавками %, &, $ в конце имени переменной. Возможность "складывать" символьные значения в любом порядке позволяет избежать инвертирования результирующей строки. Для этого вместо оператора A$ = A$ + CIF$ удобнее воспользоваться левосторонней конкатенацией A$ = CIF$+A$.

Совет 3 (Си)

Выбор длины строки, в которой будет накапливаться промежуточный или окончательный результат, определяется самым длинным разложением при r = 2 (т. е. 32 разряда плюс еще один байт под признак конца строки). Заводить две строки под правое и левое представление, наверное, не стоит. Проще произвести перестановку симметричных байтов (первый с последним, второй с предпоследним и т. д.) в одной строке. Для выполнения этой операции придется ввести счетчик количества цифр или прибегнуть к библиотечной функции strlen из раздела string.h.

Программа 2_04.bas

```
REM Перевод числа в систему с основанием r
DECLARE FUNCTION NToStr$(NUM&,R%)
CLS
INPUT "Введите натуральное число : ",N&
PRINT "Его представление в разных системах счисления таково : "
FOR J%=2 TO 16
  PRINT "по основанию ";J%,N&;"= ";NToStr$(N&,J%)
NEXT J%
END

FUNCTION NToStr$(NUM&,R%)
  A$="":   M&=NUM&
  DO
    CIF=M& MOD R% : ' Выделение очередной цифры
    IF CIF<10 THEN
       A$=CHR$(ASC("0")+CIF)+A$ : ' Замена цифры кодом ASCII
    ELSE
       A$=CHR$(ASC("A")+CIF-10)+A$
    END IF
    M&=(M&-CIF)/R% : ' Исключение обработанной цифры
  LOOP UNTIL M&=0
  NToStr$=A$
END FUNCTION
```

Программа 2_04.c

```
/* Перевод числа в систему с основанием r */
#include <stdio.h>
#include <conio.h>
char *num_to_str(long num,char r);

void main(void)
{
  long N;
  char j;
  printf("\nВведите натуральное число ");
  scanf("%ld",&N);
  printf("Его представление в разных системах счисления таково :\n");
  for(j=2; j<17; j++)
    printf("\nпо основанию %2d = %s",j,num_to_str(N,j));
  getch();
}

char *num_to_str(long num, char r)
{
  static char a[33];
  char cif, k, count=0;
  do
    {
      cif=num % r; /* Выделение очередной цифры */
      if(cif<10) a[count++]='0'+cif;
/* Замена цифры кодом ASCII, если цифра меньше 10 */
      else a[count++]='A'+cif-10;
/* Замена цифры кодом ASCII, если цифра больше 9 */
      num=num/r; /* Исключение обработанной цифры */
    }
/* Цикл изменения порядка цифр в массиве a */
  while (num != 0);
  a[count--]=0x0;
  for(k=0; k<=count/2; k++)
     { cif=a[count-k];  a[count-k]=a[k]; a[k]=cif; }
  return a;
}
```

Программа 2_04.pas

```
program num_to_pos;
{ Перевод числа в систему с основанием r }
var
  N:longint;
  j:byte;
function num_to_str(num:longint;r:byte):string;
var
  a:string[33];
  cif,k:byte;
begin
  a:='';
  repeat
    cif:=num mod r; { Выделение очередной цифры }
    if (cif<10) then a:=chr(48+cif)+a
{ Замена цифры кодом ASCII, если цифра меньше 10 }
    else   a:=chr(65+cif-10)+a;
{ Замена цифры кодом ASCII, если цифра больше 9 }
    num:=num div r; { Исключение обработанной цифры }
  until (num=0);
  num_to_str:=a;
end;
begin
  write('Введите натуральное число - ');
  readln(N);
  writeln('Его представление в разных системах счисления таково :');
  for j:=2 to 16 do
    writeln('по основанию ',j:2,' = ',num_to_str(N,j));
  readln;
end.
```

Задание 2.05. Симметричное разложение с наименьшим основанием

Дано натуральное число N (N < 30000). Найти систему счисления с наименьшим основанием Pmin, в которой N имеет симметричное представление. Например,

для N=9 Pmin=2 (910 = 10012),

для N=1000 Pmin=9 (100010 = 13319)

Программа должна запрашивать число `N`, выдавать `Pmin` и значения всех цифр в представлении числа `N` в этой системе (цифры можно выводить в виде обычных десятичных чисел).

Совет 1 (общий)

Очевидно, что самой расточительной по количеству цифр является двоичная система счисления. Однако при заданном ограничении на диапазон исходных чисел для записи самого большого числа `N` потребуется не более 15 двоичных разрядов. Поэтому для хранения цифр, получаемых при переводе в ту или иную систему счисления, вполне можно обойтись массивом из 15 элементов.

Второй очевидный факт заключается в том, что всегда найдется такая система счисления, в которой разложение любого числа `N` будет симметричным. Такой системой, в частности, является система с основанием `N-1`, т. к. в ней число `N` выглядит как `11`. Поэтому остается только организовать цикл по основаниям систем от `2` до `N-1` и, как только будет найдено симметричное разложение, прекратить перебор.

Совет 2 (общий)

Разумно организовать в программе две внешние функции — `perevod` (перевод исходного числа в систему с заданным основанием) и `proba` (проверка симметрии полученных цифр). Первая функция может быть построена по аналогии с функцией `num_to_str`, с той лишь разницей, что цифры, получаемые в процессе перевода, надо запоминать в массиве.

Совет 3 (QBasic)

Обратите внимание на схему вывода разложения, которое может не убраться в одной строке. Поэтому принято решение выводить в каждой строке по одному элементу разложения. Количества строк на экране вполне достаточно для размещения самого длинного представления в двоичной системе.

Программа 2_05.bas

```
REM Поиск симметричного разложения числа
DECLARE FUNCTION perevod!(n%,p%,a%())
DECLARE FUNCTION proba!(a%(),k%)
DIM a%(16)
CLS
INPUT "Введите число из диапазона [0,30000] : ",n%
FOR p%=2 TO n%-1
  k%=perevod(n%,p%,a%()) : ' Перевод в p-ричную систему
  IF proba(a%(),k%) <> 0 THEN
    PRINT "Симметричное разложение с минимальным основанием : "
    FOR i = 0 TO k% - 1
      PRINT TAB(0);a%(i);"*";p%; "^";k%-i;"+";
```

```
    NEXT i
    PRINT : PRINT a%(0);"*";p%;"^";0
    END
  END IF
NEXT p%
END

FUNCTION perevod (n%,p%,a%())
REM Перевод числа n в систему с основанием p
REM Цифры p-ричного числа запоминаются в массиве a
m%=n%
FOR i=0 TO 15
  a%(i)=m% MOD p%
  m%=(m%-a%(i))/p%
  IF m%=0 THEN perevod=i: EXIT FUNCTION
NEXT i
END FUNCTION

FUNCTION proba (a%(),k%)
REM Анализ числа, представленного k цифрами в массиве a
REM Если число - палиндром, то proba=1
proba=1
FOR i=0 TO k%/2
  IF a%(i)<>a%(k%-i) THEN proba=0: EXIT FUNCTION
NEXT i
END FUNCTION
```

Программа 2_05.c

```
/* Поиск симметричного разложения числа */
#include <conio.h>
#include <stdio.h>
int perevod(int n,int p,int *a);
int proba(int *a,int k);
  int i;
  main()
{
  int k,p,N,a[16];
  clrscr();
  scanf("%d",&N);
```

```
  for (p=2; p<N; p++)
    {
      k=perevod(N,p,a); /* Перевод в p-ричную систему */
      if (proba(a,k)==0) continue;
      printf("\nминимальное основание = %d\n",p);
      for(i=0; i<=k; i++)  printf("%d ",a[i]);
      break;
    }
  getch();
}
int perevod(int n,int p,int *a)
/* Перевод числа n в систему с основанием p
   Цифры p-ричного числа запоминаются в массиве a */
{
  for(i=0; i<16; i++)
    {
      a[i]=n % p;     n=n/p;
      if(n==0) return i;
    }
}
int proba(int *a,int k)
/* Анализ числа, представленного k цифрами в массиве a
   Если число - палиндром, то proba=1 */
{
  for(i = 0; i <= k/2; i++)
    if(a[i] != a[k-i]) return 0;
  return 1;
}
```

Программа 2_05.pas

```
program min_base;
{ Поиск симметричного разложения числа }
uses Crt;
var
  i,k,p,N:integer;
  a:array [0..15] of integer;
function perevod(n,p:integer):integer;
{ Перевод числа n в систему с основанием p
  Цифры p-ричного числа запоминаются в массиве a }
```

```
begin
  for i:=0 to 15 do
  begin
    a[i]:=n mod p;
    n:=n div p;
    if n=0 then
      begin perevod:=i; exit; end;
  end;
end;
function proba(k:integer):integer;
{ Анализ числа, представленного k цифрами в массиве a
  Если число - палиндром, то proba=1 }
begin
  for i:=0 to k div 2 do
    if a[i]<>a[k-i] then
    begin proba:=0; exit; end;
  proba:=1;
end;
begin
  clrscr;
  writeln('Поиск симметричного разложения с минимальным основанием');
  writeln('Введите число');
  readln(N);
  for p:=2 to N do
    begin
      k:=perevod(N,p);
      if proba(k)=0 then continue;
      writeln('минимальное основание = ',p);
      for i:=0 to k do  writeln(a[i]);
      break;
    end;
  readln;
end.
```

Задание 2.06. Суммирование десятичных цифр

Составить функцию `sum_dig(n)`, аргументом которой является длинное целое число. Возвращаемое значение должно быть равно сумме десятичных цифр числа `n`.

Совет 1 (общий)

Обратите внимание на ситуацию, когда аргумент отрицателен. Остаток от деления отрицательного числа на 10 будет также отрицательным.

Совет 2 (QBasic)

Будьте внимательны при определении очередного частного от деления числа на 10. QBasic производит округления, и результат деления, например, целого числа 15 на целое число 10 даст в частном не 1, а 2.

Совет 3 (Си, Паскаль)

Обратите внимание на то, что циклы `do` (Си) и `repeat` (Паскаль) требуют противоположных условий выхода из цикла.

Программа 2_06.bas

```
REM Суммирование десятичных цифр числа
DECLARE FUNCTION SUMDIG (N&)
CLS
INPUT "Введите целое число";M&
K=SUMDIG(M&)
PRINT "Сумма его цифр = ";K
END

FUNCTION SUMDIG(N&)
IF N&<0 THEN N&=-N& : ' Смена знака у отрицательного числа
RESULT=0
DO
  RESULT=RESULT+(N& MOD 10&) : ' Накопление суммы цифр
  N&=(N&-(N& MOD 10&))/10& : ' Удаление обработанной цифры
LOOP WHILE N&<>0
SUMDIG=RESULT
END FUNCTION
```

Программа 2_06.c

```
/* Суммирование десятичных цифр числа */
#include <stdlib.h>
int sum_digits(long N);
  main()
{
```

```
  long M;
  printf("\nВведите целое число: ");
  scanf("%ld",&M);
  printf("\nСумма его цифр = %d",sum_digits(M));
  getch();
}
int sum_digits(long N)
{
  int Result=0;
  N=labs(N); /* Смена знака у отрицательного числа */
  do
    {
      Result=Result+N%10; /* Накопление суммы цифр */
      N=N/10; /* Удаление обработанной цифры */
    }
    while(N != 0);
    return Result;
}
```

Программа 2_06.pas

```
program sumdigits;
{ Суммирование десятичных цифр числа }
var
  M:longint;
  function sum_digits(N:longint):integer;
  const
    Result:integer=0;
  begin
    if N<0 then N:=-N; { Смена знака у отрицательного числа }
    repeat
      Result:=Result+N mod 10; { Накопление суммы цифр }
      N:=N div 10; { Удаление обработанной цифры }
    until N=0;
    sum_digits:=Result;
  end;
begin
  write('Введите целое число:');
  readln(M);
```

```
  writeln('Сумма его цифр =',sum_digits(M));
  readln;
end.
```

Задание 2.07. "Счастливый" билет

Составить логическую функцию `luck(n)`, аргументом которой является число `n` из диапазона [0, 999999]. Функция должна возвращать значение `true` (Паскаль) или `1` (Си, QBasic), если ее аргумент представлен "счастливым" числом, у которого суммы трех старших и трех младших цифр совпадают.

Программа 2_07.bas

```
REM Анализ "счастливого" билета
DECLARE FUNCTION LUCK(M AS LONG)
INPUT "Введите номер билета ";N&
IF LUCK(N&)=1 THEN
  PRINT "Радуйтесь - счастливый"
ELSE
  PRINT "Нет счастья в жизни"
END IF
END

FUNCTION LUCK(M AS LONG)
REM Подсчет и сравнение сумм старших и младших цифр M
REM Если суммы совпадают LUCK=1
DIM A(6)
LUCK=0
IF M<0 OR M>999999 THEN
  PRINT "luck : недопустимый аргумент": EXIT FUNCTION
END IF
FOR I=0 TO 5
  A(I)=M MOD 10 : ' Выделение очередной цифры
  M=(M-A(I))/10 : ' Удаление обработанной цифры
NEXT I
IF (A(0)+A(1)+A(2)=A(3)+A(4)+A(5)) THEN LUCK=1
END FUNCTION
```

Программа 2_07.c

```
/* Анализ "счастливого" билета */
#include <stdio.h>
int luck(long M);
  main()
{
  long N;
  printf("\nВведите номер билета ");
  scanf("%ld",&N);
  if (luck(N)==1)
    printf("\nРадуйтесь - счастливый");
  else
    printf("\nНет счастья в жизни");
  getch();
}
int luck(long M)
/* Подсчет и сравнение сумм старших и младших цифр M
   Если суммы совпадают luck=1*/
{
  int i;
  char a[6];
  if((M<0) || (M>999999))
    {
      printf("\nluck : недопустимый аргумент");
      exit(0);
    }
  for(i=0; i<6; i++)
     {
       a[i]=M % 10; /* Выделение очередной цифры */
       M=M/10; /* Удаление обработанной цифры */
     }
  if(a[0]+a[1]+a[2]==a[3]+a[4]+a[5]) return 1;
  return 0;
}
```

Программа 2_07.pas

```
program lucky_ticket;
{ Анализ "счастливого" билета }
```

```
var
  N:longint;
function luck(M:longint):boolean;
{ Подсчет и сравнение сумм старших и младших цифр M
   Если суммы совпадают luck=true }
var
  i:integer;
  a:array [1..6] of byte;
begin
  if (M<0) or (M>999999) then
  begin
    writeln('luck : недопустимый аргумент');
    exit;
  end;
  for i:=1 to 6 do
  begin
    a[i]:=M mod 10; { Выделение очередной цифры }
    M:=M div 10;    { Удаление обработанной цифры }
  end;
  luck:=(a[1]+a[2]+a[3])=(a[4]+a[5]+a[6]);
end;
begin
  writeln('Введите номер билета');
  readln(N);
  if luck(N) then
    writeln('Радуйтесь - счастливый')
  else
    writeln('Нет счастья в жизни');
  readln;
end.
```

Задание 2.08. Количество различных цифр в десятичном числе

Составить функцию `num_digits(n)`, аргументом которой является длинное целое число. Возвращаемое значение должно быть равно количеству различных цифр в десятичной записи `n`. Например: `num_digits(1998)=3`.

Совет 1 (общий)

Предлагается завести массив из 10 счетчиков, каждый из которых будет подсчитывать количество своих цифр — первый счетчик считает нули, второй —

единицы и т. д. Вначале массив счетчиков обнуляется, а после выделения всех цифр числа проверяется, сколько счетчиков хранят ненулевые значения. Вообще говоря, можно не заниматься подсчетом количества появлений каждой цифры — достаточно фиксировать сам факт появления той или иной цифры.

Совет 2 (Си, Паскаль)

Обратите внимание на различное оформление функций `num_digits` на Си и Паскале. Локальные переменные в функциях на Си, которым присваиваются начальные значения, при повторных обращениях к функциям инициализируются заново. В процедурах и функциях Паскаля такая инициализация производится только один раз, поэтому повторный вызов может привести к неверным результатам.

Программа 2_08.bas

```
REM Определение количества различных цифр в числе
DECLARE FUNCTION NumDig!(N&)
CLS
INPUT "Введите число : ", N&
PRINT "Количество разных цифр в его записи = ";NumDig(N&)
END

FUNCTION NumDig(N&)
REM Выделение и подсчет количества разных цифр в числе N
DIM d(10) : ' Массив для фиксации обнаруженных цифр
REM При вызове функции d(i)=0
IF N&<10 THEN NumDig=1: EXIT FUNCTION
DO
  k%=N& MOD 10 :  ' Выделение очередной цифры
  d(k%)=d(k%)+1 : ' Подсчет количества цифр, равных k
  N&=(N&-k%)/10 : ' Удаление обработанной цифры
LOOP UNTIL N&=0
FOR k%=0 TO 9 : ' Цикл подсчета количества обнаруженных цифр
  IF d(k%)<>0 THEN s=s+1 : ' Если d(i)=0, цифры i не было
NEXT k%
NumDig=s
END FUNCTION
```

Программа 2_08.c

```
/* Определение количества различных цифр в числе */
#include <stdio.h>
#include <conio.h>
```

```
int num_digits(long N);
  main()
{
  long N;
  printf("\nВведите число - ");
  scanf("%ld",&N);
  printf("\Количество разных цифр в его записи = %d ",
          num_digits(N));
  getch();
}
int num_digits(long N)
/* Выделение и подсчет количества разных цифр в числе N */
{
  char d[10]={0,0,0,0,0,0,0,0,0,0}; /* Массив счетчиков цифр */
  int k,s=0;
  if(N<10) return 1;
  while (N)
    {
      d[N % 10]++; /* Выделение и учет очередной цифры */
      N=N/10; /* Удаление обработанной цифры */
    }
/* Цикл подсчета количества обнаруженных цифр */
  for(k=0; k<10; k++) if(d[k]) s++;
  return s;
}
```

Программа 2_08.pas

```
program NumDigits;
{ Определение количества различных цифр в числе }
var
  N:longint;
function num_digits(N:longint):byte;
{ Выделение и подсчет количества разных цифр в числе N }
var
   d:array [0..9] of byte; { Массив счетчиков }
   s,k:byte;
begin
  if N<10 then num_digits:=1
```

```
  else
    begin
      for k:=0 to 9 do d[k]=0; { Сброс счетчиков }
      s:=0;
      while N <> 0 do
        begin
          inc(d[N mod 10]); { Выделение и учет очередной цифры }
          N := N div 10; { Удаление обработанной цифры }
        end;
{ Цикл подсчета количества обнаруженных цифр }
      for k:=0 to 9 do if(d[k]<>0) then inc(s);
      num_digits:=s;
    end;
end;

begin
  write('Введите число : ');
  readln(N);
  writeln('Количество разных цифр в его записи = ',num_digits(N));
  readln;
end.
```

Задание 2.09. Определение цифры в заданной позиции

Составить функцию `digit_in_pos(n,i)`, аргументами которой являются длинное целое число `n` и номер `i` позиции цифры в десятичном представлении `n`. Отсчет номеров позиций ведется справа налево от 0 и соответствует "весу" цифры (степени основания), с которым цифра входит в число. Например:

```
n=1985
  в  0-й позиции находится цифра 5;
  в  1-й позиции находится цифра 8;
  во 2-й позиции находится цифра 9;
  в  3-й позиции находится цифра 1;
```

Функция `digit_in_pos` должна возвращать цифру, расположенную в `i`-й позиции числа `n`.

Программа 2_09.bas

```
REM Анализ цифр в каждой позиции заданного числа
DECLARE FUNCTION DIGINPOS(N AS LONG,J AS INTEGER)
```