Г. Г. Рапаков, С. Ю. Ржеуцкая

bhv®

Turbo Pascal для студентов и школьников

Алгоритмизация и программирование

Основные конструкции языка

Примеры программ

ОСНОВЫ ИНФОРМАТИКИ

Георгий Рапаков
Светлана Ржеуцкая

Turbo Pascal для студентов и школьников

Санкт-Петербург
«БХВ-Петербург»
2002

УДК 681.3.06(075.3+075.8)
ББК 32.973.26-018.1
Р23

Рапаков Г. Г., Ржеуцкая С. Ю.

Р23 Turbo Pascal для студентов и школьников. — СПб.: БХВ-Петербург, 2002. — 352 с.: ил.

ISBN 5-94157-240-9

Книга является обобщением многолетнего опыта авторов по обучению студентов и школьников старших классов основам программирования. Тематику ее можно определить как "Конспект начинающего программиста" или "Популярный учебник". Изложение материала построено таким образом, чтобы читатель сумел понять основные принципы разработки компьютерных программ, не увязнув в многочисленных тонкостях языка Turbo Pascal. В книге содержатся краткие сведения об алгоритмизации и программировании, операторах, процедурах, функциях и модулях, приводятся описания основных конструкций языка, иллюстрированные большим количеством примеров, рассмотрены основные приемы программирования и практической работы в интегрированной среде Turbo Pascal.

Для школьников старших классов и студентов

УДК 681.3.06(075.3+075.8)
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Анна Кузьмина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.08.02.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 28,38.
Тираж 5000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

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

ISBN 5-94157-240-9

Содержание

Введение

Несмотря на появление новых технологий *Turbo Pascal* (Турбо Паскаль), во многом задуманный как язык для обучения, и на сегодняшний день остается одним из самых удобных средств для изучения программирования. Это определяет его популярность среди широкой аудитории начинающих программистов: школьников и студентов.

Целью издания, которое держит в руках читатель, является оказание помощи в изучении основ программирования.

В книге содержатся краткие сведения об алгоритмизации и программировании; приводятся описания основных конструкций Turbo Pascal, иллюстрированные большим количеством примеров; рассмотрены основные приемы программирования и практической работы в интегрированной среде Turbo Pascal.

Книга, сочетающая свойства справочника и конспекта, рассчитана на школьников старших классов и студентов. Две части книги отвечают первому и второму семестрам (полугодиям) обучения студентов (школьников) программированию. *Тематику книги* можно определить как "Конспект начинающего программиста" или "Популярный учебник".

Объективную оценку книги может дать читательская аудитория и специалисты. С точки же зрения авторов их труд отличает от множества других изданий удачное сочетание нескольких факторов:

- *краткости*, свойственной конспекту лекций;
- строгой *систематизации* материала;
- *доступности для усвоения*, присущей отработанному курсу, вобравшему собственный опыт авторов и почерпнутый ими из многочисленных публикаций;
- отсутствия "наукообразности", с одной стороны, и "заигрывания с читателем", с другой: *деловой подход к получению максимума сведений в минимальные сроки*;

- *привлечения* внимания читателя к наиболее важным материалам в каждой теме;
- *большого количества подобранных примеров*, облегчающих обучение, не утомляющих читателя и стимулирующих у него интерес к материалу;
- наличия *комментариев* к текстам программ.

Представленный учебный материал прошел успешную проверку на занятиях со студентами и школьниками.

ЧАСТЬ I

Основы программирования на Turbo Pascal: путь от простейших задач до разработки собственных подпрограмм

ГЛАВА 1

Основы алгоритмизации

1.1. Алгоритмизация и требования к алгоритму

1.1.1. Алгоритм и алгоритмизация

Процессор электронно-вычислительной машины (ЭВМ) или персонального компьютера (ПК) — это ее "мозг", который умеет выполнять лишь простейшие команды. Для решения сложных задач обработки информации программист должен составить *алгоритм* — подробное описание последовательности арифметических и логических действий, расположенных в строгом логическом порядке и позволяющих решить конкретную задачу. Составление такого пошагового описания процесса решения задачи называется ее *алгоритмизацией*. Слово алгоритм, по существу, является синонимом таких слов, как способ, рецепт и т. п.

Требования, предъявляемые к алгоритму:

- *однозначность* — предлагаемые действия должны быть "понятны" компьютеру, а порядок исполнения этих действий должен быть единственно возможным, любая неопределенность или двусмысленность недопустимы;
- *массовость* — пригодность алгоритма для решения не только данной задачи, а множества родственных задач, относящихся к общему классу;
- *детерминированность* — повтор результата при повторе исходных данных;
- *корректность* — способность алгоритма давать правильные результаты решения задачи при различных исходных данных;
- *конечность* — решение задачи должно быть получено за конечное число шагов алгоритма, "зацикливание" недопустимо;
- *эффективность* — для успешного решения задачи должны использоваться ограниченные ресурсы конкретного компьютера (время работы процессора, объем оперативной памяти, быстродействие жесткого диска и др.).

1.2. Блок-схемы алгоритмов

1.2.1. Способы записи алгоритма

Разработка алгоритма решения задачи — сложный творческий процесс. Записать алгоритм в виде компьютерной программы без каких-либо предварительных рассуждений может только опытный программист при решении небольшой по объему, четко поставленной задачи. В реальной жизни такие задачи встречаются редко, поэтому обычно разработчик сначала продумывает алгоритм и записывает его в какой-либо удобной форме, а затем реализует алгоритм в виде программы.

При разработке сложных коммерческих программных продуктов часто алгоритмизацию выполняет один человек, а запись программы по имеющемуся алгоритму — другой. Следовательно, необходимо иметь такие способы записи алгоритмов, которые легко воспринимаются человеком, но являются достаточно строгими, чтобы их можно было впоследствии перевести на язык компьютера.

Существуют различные варианты записи алгоритмов. К основным относятся описательный и графический способы. *Описательным* называется алгоритм, составленный на естественном, в частности, математическом языке. *Графический* способ отличает компактная и наглядная форма записи в виде специальных графических знаков с указанием связи между ними.

1.2.2. Блок-схемы

При разработке программ рекомендуется использовать графический способ записи алгоритма в виде блок-схемы. *Блок-схема* — это графическое изображение алгоритма в виде плоских геометрических фигур (блоков), соединенных линиями. Внутри блока записывается действие, которое нужно выполнить, или условие, которое необходимо проверить. Блок-схема — стандартный способ записи алгоритма, существует государственный стандарт (ГОСТ), содержащий перечень правил построения блок-схем.

Основные блоки, которые используются при составлении графического алгоритма, изображены на рис. 1.1, где *а* — начало (конец) алгоритма; *б* — блок ввода/вывода; *в* — операционный блок; *г* — логический (условный) блок; *д* — цикл с параметром (для параметра цикла `i` указывается его начальное и конечное значение, шаг равен единице). Конструкция "цикл с параметром" отдельно ГОСТом не регламентируется, однако приведенное обозначение (шестиугольник) разрешено использовать для различных целей, поэтому такое изображение цикла на блок-схеме часто встречается в литературе по программированию. Цикл с параметром будет подробно рассматриваться далее *(см. разд. 3.9.3)*. В ГОСТ 19.701-90 (ИСО 5807-85), дата введения 01.01.1992 г., приводится еще один возможный вариант обозначений для циклов. Однако он представляется неудачным (рис. 1.1, *е—ж*).

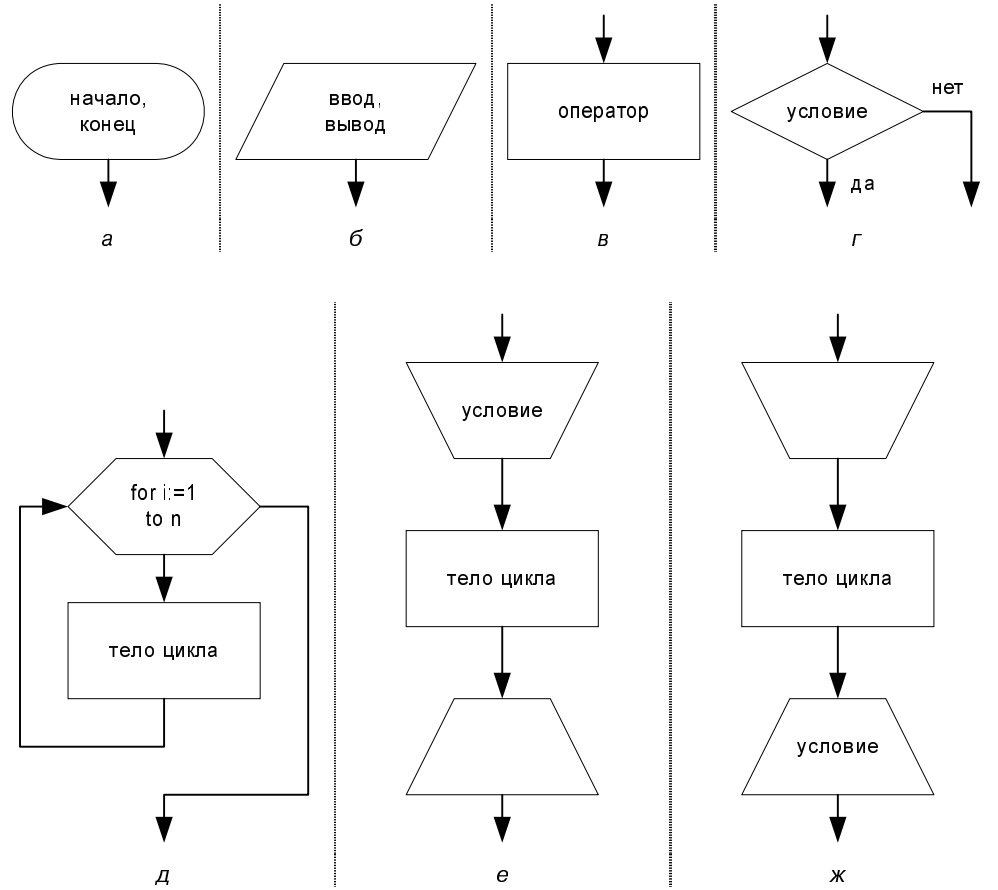

Рис. 1.1. Условные обозначения на схемах алгоритмов

1.2.3. Следование, ветвление, цикл

Алгоритмические структуры (рис. 1.1, *а*, *б*, *в*) образуют линейную последовательность операций, которые выполняются по очереди в порядке записи, — *следование*. Программную реализацию такой алгоритмической структуры называют *линейной программой*. Линейные программы обычно предназначены для решения простейших задач, в которых не предусмотрен выбор из нескольких возможных направлений хода программы или циклическое повторение операций.

Возможность альтернативного выбора при выполнении программы предоставляют *ветвления* (рис. 1.1, *г*), при выполнении которых алгоритм может пойти по одной из двух возможных ветвей в зависимости от справедливости проверяемого условия. Иногда выделяют также *обход*, который представляет

собой пропуск нескольких шагов алгоритма при выполнении или невыполнении какого-либо условия.

Цикл (рис. 1.1, *д*) представляет собой многократно повторяющуюся последовательность шагов алгоритма.

1.2.4. Пример блок-схемы алгоритма

Рассмотрим пример блок-схемы алгоритма игры "Угадай число".

Условие игры: игрок должен угадать число, "задуманное" компьютером — случайное число в диапазоне от 0 до 1000. Алгоритм игры представлен на рис. 1.2. Игра начинается с того, что компьютер задумывает случайное

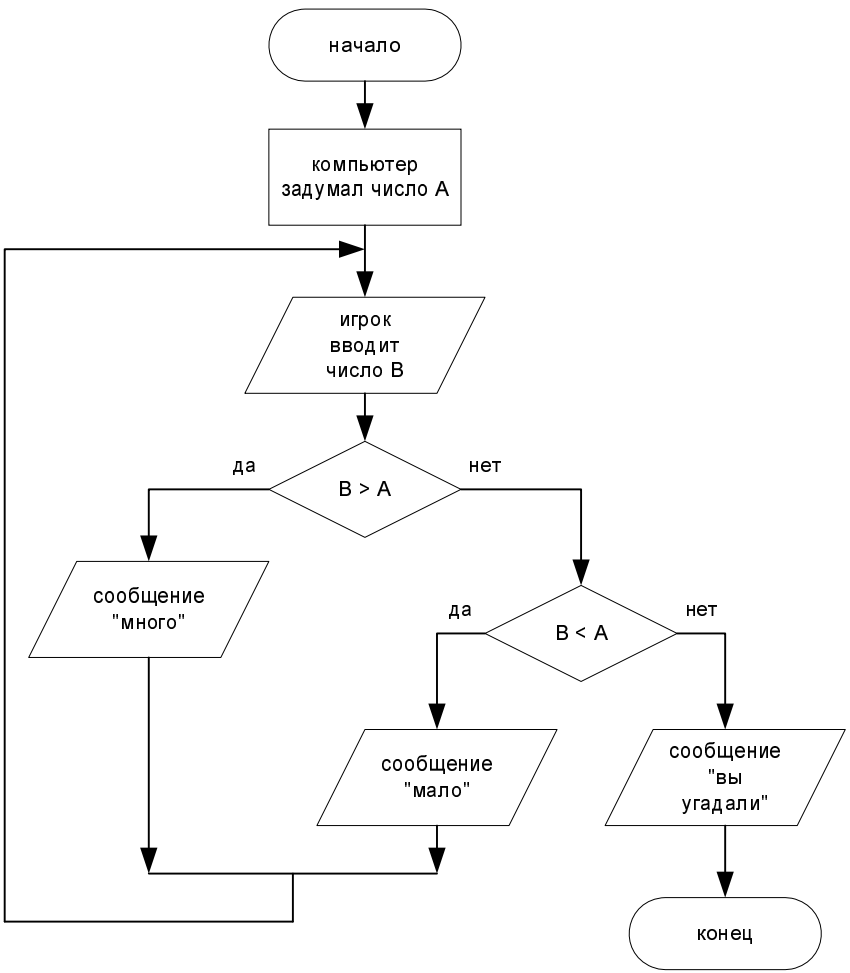

Рис. 1.2. Блок-схема алгоритма игры "Угадай число"

число *A*. В свою очередь, игрок вводит число *B*, пытаясь угадать значение *A*. Выполняется проверка: число *B* строго больше числа *A*? Если это так, т. е. "да", то компьютер выводит сообщение "много" и просит игрока ввести следующее число, еще раз испытав удачу. Если же результат — "нет", то компьютер выполняет следующую проверку: *B* строго меньше *A*? Если ее результат — "да", то выводится сообщение "мало", и компьютер ожидает ввода игроком следующего числа. Если же результат второй проверки — "нет", это означает, что игрок угадал число, задуманное компьютером. В таком случае выводится сообщение "вы угадали", и игра завершается. Программа, соответствующая блок-схеме, приведена в листинге 3.12.

1.3. Этапы разработки программы

1.3.1. Язык программирования. Программа

Команды, выполняемые процессором ПК, являются электрическими сигналами, которые можно представить в виде последовательностей нулей и единиц. Каждой команде соответствует свое число. Таким образом, процессор имеет дело с *машинным кодом*. Написать программу на нем может только очень опытный программист, хорошо знающий *архитектуру процессора* (его устройство) и *систему команд* (набор допустимых инструкций). Большинство программ создаются при помощи "посредников", в качестве которых выступают *языки программирования высокого уровня*.

Совокупность средств и правил представления алгоритма в виде, пригодном для выполнения вычислительной машиной, называется *языком программирования*.

Программа — это запись (реализация) алгоритма на языке программирования.

1.3.2. Этапы разработки

В процессе создания любой программы можно выделить несколько этапов.

- *Постановка задачи* — выполняется специалистом в предметной области на естественном языке (русском, английском и т. д.). Необходимо определить цель задачи, ее содержание и общий подход к решению. Возможно, что задача решается точно *(аналитически)*, и без компьютера можно обойтись. Уже на этапе постановки надо учитывать эффективность алгоритма решения задачи на ЭВМ, ограничения, накладываемые *аппаратным и программным обеспечением* (АО и ПО).
- *Анализ задачи и моделирование* — определяются исходные данные и результат, выявляются ограничения на их значения, выполняется формализованное описание задачи и построение (выбор) математической модели, пригодной для решения на компьютере.

- *Разработка или выбор алгоритма решения задачи* — выполняется на основе ее математического описания. Многие задачи можно решить различными способами. Программист должен выбрать оптимальное решение. Неточности в постановке, анализе задачи или разработке алгоритма могут привести к *скрытой ошибке* — программист получит неверный результат, считая его правильным.
- *Проектирование общей структуры программы* — формируется модель решения с последующей детализацией и разбивкой на подпрограммы, определяется "архитектура" программы, способ хранения информации (набор переменных, массивов и т. п.).
- *Кодирование* — запись алгоритма на языке программирования. Современные системы программирования, подобные *Delphi*, позволяют ускорить процесс разработки программы, автоматически создавая часть ее текста, однако творческая работа по-прежнему лежит на программисте. Для успешной реализации целей проекта программисту необходимо использовать *методы структурного программирования (см. разд. 6.1)*.
- *Отладка и тестирование программы.* Под *отладкой* понимается устранение ошибок в программе. *Тестирование* позволяет вести их поиск и, в конечном счете, убедиться в том, что полностью отлаженная программа дает правильный результат. Для этого разрабатывается *система тестов* — специально подобранных контрольных примеров с такими наборами параметров, для которых решение задачи известно. Тестирование должно охватывать все возможные ветвления в программе, т. е. *проверять все ее инструкции*, и включать такие исходные данные, для которых решение *невозможно*. Проверка особых, *исключительных ситуаций*, необходима для анализа корректности. Например, программа должна отказать клиенту банка в просьбе выдать сумму, отсутствующую на его счете. В ответственных проектах большое внимание уделяется так называемой "защите от дурака" *(fool-tolerance)*, подразумевающей устойчивость программы к неумелому обращению пользователя. Использование специальных программ — *отладчиков*, которые позволяют выполнять программу по отдельным шагам, просматривая при этом значения переменных, значительно упрощает этот этап *(см. приложение 1)*.
- *Анализ результатов* — если программа выполняет моделирование какого-либо известного процесса, следует сопоставить результаты вычислений с результатами наблюдений. В случае существенного расхождения необходимо изменить модель.
- *Публикация* результатов работы, *передача заказчику* для эксплуатации.
- *Сопровождение программы* — включает консультации представителей заказчика по работе с программой и обучение персонала. Недостатки и ошибки, замеченные в процессе эксплуатации, должны устраняться.

1.4. Ошибки

1.4.1. Компилятор. Синтаксис и семантика

Особое значение для программиста имеет предупреждение и исправление ошибок в алгоритме и программе решения задачи. Прежде чем выполнить программу, ее текст необходимо ввести в компьютер. Для ввода и изменения (редактирования) текстов используется специальная программа — *текстовый редактор.*

Текст набранной программы, для того чтобы быть "понятым" компьютером, должен быть переведен на язык машинных кодов. Такой перевод называется *компиляцией* и выполняется специальной программой — *компилятором.* Компилятор анализирует программу и определяет, содержит ли она ошибки. В случае их обнаружения вся работа останавливается. Если же правила языка программирования не нарушены, то формируется модуль на машинном языке, который затем и исполняется *(см. приложение 1).*

В отличие от естественных языков, таких как русский, английский и др., язык программирования имеет очень ограниченное количество "слов", понятных компилятору, и строгие правила записи команд. Совокупность этих требований образует *синтаксис* языка программирования, а смысл команд и других конструкций языка — его *семантику.*

1.4.2. Типы ошибок

Программирование является творческим процессом, поэтому ошибки неизбежно встречаются даже у опытных программистов. Различают следующие типы ошибок: синтаксические ошибки (ошибки компиляции), ошибки выполнения и ошибки в алгоритме программы (семантические).

- *Синтаксические ошибки* возникают при нарушении правил языка (в нашем случае — языка Turbo Pascal), их обнаруживает компилятор, который не может из-за ошибки "понять" назначение команды.
- *Ошибки выполнения* не нарушают синтаксис языка. Однако они приводят к ошибочным операциям в процессе выполнения программы, например попытке деления на ноль или извлечения квадратного корня из отрицательного числа. Перечень Turbo Pascal об ошибках содержит более 200 сообщений *(см. приложение 1).*
- *Ошибки в алгоритме* программы при верных исходных данных и внешне безошибочной работе программы приводят к неверным результатам. Этот тип ошибок наиболее коварен и труден для исправления, т. к. пользователь, получая ошибочный результат, считает его верным, поскольку никаких сообщений об ошибках не было. Семантические ошибки должен обнаруживать сам программист. В поиске и исправлении ошибок ему может оказать существенную помощь *интегрированная среда разработки Turbo Pascal* и ее встроенный отладчик *(см. приложение 1).*

ГЛАВА 2

Программа на Turbo Pascal

2.1. Начальные сведения

2.1.1. Пример учебной программы

Создадим первую учебную программу вычисления суммы двух целых чисел. Сценарий взаимодействия человека и компьютера при решении данной задачи можно предложить следующий. Компьютер запрашивает у человека значение первого целого числа, человек набирает значение на клавиатуре и нажимает клавишу <Enter>, компьютер считывает введенное число и записывает в память под именем `a`. Затем компьютер запрашивает значение второго целого числа, считывает его и записывает в память под именем `b`. После этого компьютер выполняет сложение чисел `a` и `b`, записывает результат в память под именем `sum`, выводит на экран сообщение "`Сумма чисел...`" и значение величины `sum` (листинг 2.1).

Листинг 2.1. Вычисление суммы двух целых чисел

```
{ Учебная программа вычисления суммы двух целых чисел }
program example;          { заголовок программы }
var
   a,b,sum: integer;      { переменные a,b,sum — целые }
begin                     { начало программы }
   { вывод запроса на экран }
   write('введите значение целого числа a: ');
   { ввод значения a с клавиатуры }
   readln(a);
   { вывод запроса и ввод значения b }
```

```
   write('введите значение целого числа b: ');
   readln(b);
   { вычисление переменной sum }
   sum:=a+b;
   { вывод ответа }
   write('сумма чисел ',a,' и ',b, ' = ',sum);
end.                    { конец программы }
```

Прочитав текст программы, обратите внимание на ее структуру *(см. разд. 2.2)*.

1. В данной программе использованы следующие зарезервированные слова языка Turbo Pascal — слова, за которыми закреплено строго определенное значение:

   - `program` — заголовок программы, определяющий ее название. Эту строку в программе можно было бы опустить, т. к. она имеет чисто декоративное значение;
   - `var` — начало объявления переменных (сокращение от английского слова *variable*, переменная);
   - `integer` — указание, что переменные `a`, `b`, `sum` — целые числа, т. е. могут принимать целочисленные значения, например 2, 3, 0, 287, 21, 32 и др. на интервале [−32 768, 32767];
   - `write('Текст')` — инструкция компьютеру о выводе на экран сообщения `'Текст'`. Обратите внимание, что текст справа и слева ограничен символом `'` — апострофом;
   - `readln(a)` — инструкция компьютеру о считывании значения переменной `a` с клавиатуры.

2. Для вычисления суммы чисел `a` и `b` в программе использована запись инструкции присваивания суммы чисел `a` и `b` переменной `sum`. Присваивание записывается с помощью пары символов `:=` в виде `sum := a + b`.

3. Программа представляет собой последовательность символов, для удобства восприятия разбитых на строки. Никакие знаки переноса не используются. Начиная строку с нескольких пробелов, можно добиться, чтобы некоторые зарезервированные слова располагались одно под другим. Это облегчает составление и понимание программы. Каждая инструкция программы завершается *разделителем* `;` (точкой с запятой), в конце программы ставится `.` (точка). Пояснения к программе, не влияющие на исполнение, записываются в фигурных скобках: `{ это комментарий }`. Все комментарии в данном примере можно убрать, при этом программа останется работоспособной.

2.1.2. Базовые понятия

Характеристики программы. Данные. Результаты

Программа реализует алгоритм решения задачи. Основные *характеристики программы* следующие:

- точность полученного результата;
- время выполнения;
- объем требуемой памяти.

Функционирование любой программы связано с обработкой *данных*. Данные, предназначенные для обработки, называются *исходными* и задаются обычно в начале выполнения программы. Программа по ходу выполнения может запрашивать недостающие исходные данные. Основной способ задания исходных данных — ввод с клавиатуры (см. листинг 2.1). Выбор какого-либо пункта меню, щелчок мышью на определенной кнопке на экране — также способы ввода исходных данных. Иногда программа может считывать исходные данные из файлов на диске *(см. гл. 12)*.

В процессе выполнения программы исходные данные преобразуются в *результаты*. Результаты выводятся на экран или печатающее устройство — принтер в текстовом или графическом виде, а также могут быть записаны в файлы на диске.

Константы

Каждый элемент данных, используемый в программе, является константой или переменной. *Константами* называются элементы данных, значения которых в процессе выполнения программы не изменяются. В языке Turbo Pascal используются константы следующих видов: числовые, логические (булевские), символьные и строковые. Числовые константы предназначены для представления числовых данных (целых и вещественных). Булевские константы используются для представления данных, имеющих смысл логических высказываний (да-нет, истина-ложь, 1—0). Символьные и строковые константы — это отдельные символы и их последовательности.

Переменные

Переменные, в отличие от констант, могут менять свои значения при выполнении программы. В программировании переменную можно трактовать как *одну или несколько ячеек оперативной памяти компьютера, которым присвоено определенное имя (идентификатор)*. Имя в данном случае выступает как посредник, позволяющий отвлечься от "неудобного" для использования адреса конкретной ячейки (или ячеек) с интересующим нас содержимым, но сохранить возможность "простого" обращения к нему. *Содержимое этих ячеек может меняться, но имя переменной остается неизменным*. Каждое *новое*

значение, записанное в ячейку памяти, *"затирает"* предыдущее значение, поэтому *в любой момент времени* переменная имеет только одно, *текущее*, значение. Обычно переменные используются для хранения исходных данных, результатов программы, а также промежуточных данных, которые образуются по ходу выполнения алгоритма.

В математике значение переменной в рамках определенной задачи неизменно. Именно поэтому высказывание `a := a + 1` математик сочтет неверным. Тем не менее, для программиста это абсолютно правильная конструкция, которая задает вычисление суммы содержимого ячейки `a` и числовой константы `1` и занесение полученного результата в ту же ячейку `a`. После выполнения этого действия старое значение переменной `a` будет безвозвратно потеряно, т. к. одна ячейка памяти не может вместить сразу несколько значений. *Это очень важный момент в программировании.*

Turbo Pascal позволяет давать имена не только переменным, но и константам, и это считается хорошим стилем программирования. В рассмотренном выше примере (см. листинг 2.1) простейшая программа содержала только имена переменных, реальные программы обычно содержат раздел определения констант *(см. разд. 2.2.4)*.

Описание данных. Типы

Именование констант и переменных в программировании очень похоже на использование символических выражений в алгебре, однако, для того чтобы компилятор смог их обрабатывать, нужно снабдить его некоторой дополнительной информацией — *выполнить описание*. В этой информации сообщается о *типе* каждой именованной величины. Идея типов берет свое начало в математике и логике и призвана предотвращать двусмысленные и ошибочные конструкции языка программирования.

Человек, решающий какую-нибудь задачу "вручную", обладает интуитивной способностью быстро разобраться в типах данных и тех операциях, которые для каждого типа справедливы; известно, например, что нельзя извлечь квадратный корень из слова или написать число с заглавной буквы. Одна из причин, позволяющих легко провести такое распознавание, состоит в том, что слова, числа и другие обозначения выглядят по-разному. Однако для компьютера все типы данных сводятся, в конечном счете, к последовательности *битов, образующих байты* — содержимому ячеек памяти, поэтому различие в типах следует делать явным.

Таким образом, Turbo Pascal, как и другие так называемые языки высокого уровня, позволяет отвлечься от представления данных в виде последовательности двоичных разрядов, наилучшего с точки зрения компьютера. При написании программы программист может использовать понятия, соответствующие терминам решаемой задачи: целое и вещественное число, матрица (массив), запись, файл и т. д. Это существенно упрощает решение. Есте-

ственно, что в конце концов все они отображаются на конкретное битовое представление.

Замечание

В словосочетании "язык высокого уровня" термин "высокий" не следует воспринимать буквально. Аналогично, для ассемблера — языка программирования низкого уровня — слово "низкий" не значит "плохой". Просто во втором случае имеется в виду, что инструкции языка близки к машинному коду и учитывают систему команд процессора.

Инструкции. Операторы

Алгоритм решения любой задачи состоит из отдельных, довольно мелких шагов. В программе для каждого шага алгоритма записывается отдельная инструкция (команда). Отдельные инструкции записываются также для организации ветвлений и циклов. Таким образом, программа состоит из отдельных инструкций, или команд. Эти инструкции в программировании принято называть *операторами*. Программа состоит из операторов подобно тому, как здание строится из отдельных кирпичиков. Часто в литературе по программированию программу определяют как последовательность операторов.

Операторы могут объединяться в более крупные конструкции — *составные операторы, процедуры и функции*. Такие конструкции состоят из нескольких элементарных операторов, однако в программе могут использоваться как один оператор. Продолжая аналогию со строительством, можно сказать, что используются как отдельные кирпичики (элементарные операторы), так и строительные блоки.

Процедуры и функции универсального назначения могут располагаться в особым образом оформленных файлах — *библиотечных модулях*. Все эти конструкции будут подробно разбираться ниже.

2.1.3. Типы данных

Тип. Классификация типов

Тип определяет множество значений, которые могут принимать объекты программы (константы и переменные), а также совокупность операций, допустимых над этими значениями.

Например, значения 1 и 3 относятся к целочисленному типу, и над ними можно выполнять любые арифметические операции. Значения `'отличная'` и `'учеба'` принадлежат к строковому типу и над ними можно выполнять только одну операцию — *склеивания, сцепления, или конкатенации* текста (обозначается через `+`).

Все типы данных, используемые в Turbo Pascal, можно разделить на две большие группы: скалярные (простые) и структурированные (составные). *Скалярные типы* в свою очередь подразделяются на стандартные и пользовательские (перечисляемый и интервальный). *Стандартные типы* предлагаются программисту разработчиками Turbo Pascal. К ним относятся: целочисленные, вещественные, символьный (литерный), логический (булевский) и указатели. *Структурированные типы* имеют в своей основе скалярные типы данных. К структурированным относятся: строки, массивы, множества, записи и файлы.

Целочисленные типы, символьный, логический и пользовательские типы данных (перечисляемый и интервальный) образуют группу так называемых *порядковых типов*, имеющих большое значение.

Тип данных очень важен при выделении памяти под переменные, поскольку каждому типу соответствует строго определенный размер ячейки памяти. В любом случае этот размер ограничен, следовательно, *все типы данных имеют ограниченный диапазон значений* (см. табл. 2.1—2.3). Этот факт не согласуется с нашими математическими представлениями о числовых множествах. Тем не менее, с ним приходится считаться.

Стандартные типы

Целые и *вещественные* типы предназначены для представления числовых данных. В математике рассматривается *бесконечное* множество целых чисел. Целый тип в языке Turbo Pascal — это *интервал* целых чисел (табл. 2.1). Операции над целыми числами (см. табл. 3.1) определены лишь тогда, когда исходные данные *(операнды)* и результат лежат в этом интервале. Иначе возникает ситуация, называемая *переполнением*. За исключением переполнения все операции над аргументами целого типа выполняются *точно (см. разд. 3.3.1)*.

Таблица 2.1. *Целочисленные типы данных*

Название целого типа	Диапазон возможных значений	Память, байт
`byte` (байтовый)	0—255	1
`shortint` (короткий целый)	–128—127	1
`integer` (целый)	–32 768—32 767	2
`word` (слово)	0—65 535	2
`longint` (длинный целый)	–2 147 483 648—2 147 483 647	4

В математике вещественные числа — это *бесконечное непрерывное* множество чисел. В вычислительных машинах вещественные числа представляются *конечным* множеством значений (табл. 2.2).

Например, внутреннее представление типа `real` может дать $2^{48} =$ $= 281\,474\,976\,710\,656$ (более чем 10^{14}) возможных комбинаций значащих разрядов в отведенных для него 6 байтах, или 48 битах. Это очень большое число, но все же оно не сопоставимо с множеством вещественных чисел.

Таблица 2.2. *Вещественные типы данных*

Название вещественного типа	Диапазон возможных значений (плюс-минус)	Количество значащих цифр	Память, байт
`single` (с одинарной точностью)	1,5e–45—3,4e38	7—8	4
`real` (вещественный)	2,9e–39—1,7e38	11—12	6
`double` (с двойной точностью)	5,0e–324—1,7e308	15—16	8
`extended` (с повышенной точностью)	3,4e–4932—1,1e4932	19—20	10
`comp` (сложный)	–2e63+1—2e63–1	19—20	8

Логический (булевский) тип имеет всего два значения: `true` (да — истина, 1) и `false` (нет — ложь, 0), причем данные значения упорядочены, т. е. в операциях сравнения `true > false` (табл. 2.3).

Символьный (литерный) и строковый типы представляют данные, являющиеся *символами* и их последовательностями — *строками* (см. табл. 2.3). В памяти компьютера символы хранятся в виде их *числовых кодов*. Числовые коды преобразуются в буквы и другие символы лишь в момент их вывода на экран или принтер. Соответствие между символом и его кодом задается при помощи *кодовой таблицы*, которая находится в памяти компьютера и используется при выводе символов *(см. приложение 2)*.

Таблица 2.3. *Символьный и логический (булевский) типы данных*

Тип	Диапазон возможных значений	Память, байт
`char` (символьный, литерный)	Символы кодовой таблицы	1
`boolean` (булевский)	`true`, `false`	1

Переменные, описываемые любым из типов `byte`, `shortint`, `integer`, `word`, `longint`, принимают только целые значения. Типы `byte`, `word` — беззнаковые.

Переменные, описываемые любым из типов `single`, `real`, `double`, `extended`, `comp` принимают только вещественные значения — положительные и отрицательные.

Наиболее часто в простейших программах используются типы `integer` и `real`.

Замечание

При разработке программ, критичных ко времени счета, следует заменять тип `real` на `single`, `double` или `extended`. Скорость вычислений при этом увеличивается не менее чем в 2—3 раза. Но пользоваться этими типами несколько сложнее, т. к. необходимо изменить способ компиляции, используемый по умолчанию. Сделать это можно двумя методами: при включенной директиве компилятора `{$N+}` или установив переключатель [×] для соответствующего пункта меню интегрированной среды Turbo Pascal: **Options | Compiler | 8087/80287**. Более подробно директивы компилятора рассмотрены ниже *(см. разд. 2.1.4)*.

Тип `comp`, являясь вещественным (приблизительно от $-9{,}2\times10^{18}$ до $9{,}2\times10^{18}$), фактически представляет "большое" целое число со знаком, сохраняющее 19—20 значащих десятичных цифр. В то же время в выражениях он полностью совместим с любым другим вещественным типом. Наиболее подходящая область для его применения — это бухгалтерские расчеты.

Данные целых типов могут быть представлены как в десятичной, так и в шестнадцатеричной *системах счисления*.

Замечание

Признаком записи *шестнадцатеричной* константы является знак доллара — $, после которого следуют шестнадцатеричные цифры (допустимый диапазон — от $00000000 до $FFFFFFFF, при адресации абсолютных переменных — от $0000 до $FFFF). В десятичной системе счисления для представления числа используются десять цифр от 0 до 9. Для шестнадцатеричной их шестнадцать — от 0 до 9 и буквы: A (это шестнадцатеричная цифра, соответствующая числу 10 обычной десятичной системы счисления), B (это — 11), C (это — 12), D (это — 13), E (это — 14), F (это — 15). Например: 123 ($123=1\times10^2+2\times10^1+3\times10^0$) — целое десятичное число, а $7B ($7B=7\times16^1+B\times16^0$) — число, равное ему, в шестнадцатеричной форме записи.

Бит. Байт

Вся информация (числа, логические значения, символы) хранится в памяти компьютера в *двоичной* форме в виде последовательности *битов* (от *binary digit*, двоичная цифра). Каждый бит может принимать значение одной двоичной цифры — *ноль* или *единица*. Восемь битов объединены в *байт*. Мак-

симальное число, которое можно записать при помощи восьми двоичных цифр — это 11111111, что соответствует десятичному числу 255, минимальное — 0. Поэтому значением байта может быть любое целое число от 0 до 255 (всего их 256).

Так как переменные разных типов могут принимать различные значения, то для их хранения нужен соответствующий им объем памяти (ячейки разных размеров). Память под переменные выделяется в байтах (целым числом).

Например, значением символьной переменной (типа `char`) может быть любой из 256 символов (столько разных символов в кодовой таблице). Поэтому для хранения переменной такого типа достаточно одного байта (8-разрядное слово). Значением переменной типа `integer` является обязательно *целое* число в диапазоне от −32768 до 32 767 (65 535 значений). Для хранения переменной этого типа требуется два байта (16-разрядное слово). Несимметричность диапазона значений относительно нуля вызвана тем, что при традиционной кодировке целых чисел в слове из n битов можно записать числа в диапазоне от -2^{n-1} до $2^{n-1}-1$. Для беззнакового числа типа `word` диапазон $[0..\ 2^{16}-1]$ соответствует значениям [0; 65535]. Очевидно, что чем больше диапазон значений типа, тем больше байтов нужно для хранения переменной. Так, для типа `longint` диапазон $[-2^{31};\ 2^{31}-1]$ соответствует значениям [−2 147 483 648; 2 147 483 647].

Формы записи вещественных чисел

Вещественные числа могут записываться двумя способами — в общепринятой и экспоненциальной форме. Общепринятая форма предполагает запись по обычным правилам арифметики. Целая часть от дробной отделяется *десятичной точкой*, а не запятой, как в математике. Если точка отсутствует, число считается *целым*.

Запись вещественного числа *в экспоненциальной форме* (в форме с *мантиссой* и *порядком*) использует степень десяти (например: 25×10^{-3}) и удобна для записи очень больших и очень маленьких чисел. При этом число изображается так: пишется мантисса, знак умножения опускается, вместо основания 10 пишется буква *e*, а следом указывается порядок (показатель степени). Буква *e*, предшествующая порядку, читается как "умножить на 10 в степени".

Например, 123,456 или −11,9 — общепринятая форма, а 5.18e+02 (518) или 10e−03 (0,01) — экспоненциальная.

Примеры *неправильной* записи вещественных чисел:

- 123 — отсутствует десятичная точка;
- 1,23 — запятая вместо точки;
- 0.123−03 — отсутствует обозначение порядка *e*;
- 12.34e1.2 — порядок числа должен быть целым.

Любое вещественное число хранится в памяти компьютера в экспоненциальной форме: отдельно — мантисса и отдельно — порядок. При этом под мантиссу и порядок отводится строго определенное количество двоичных разрядов. Выбор такого представления имеет несколько следствий:

- существуют очень маленькие значения (машинное ε — *"машинное эпсилон"*), которые не могут быть представлены (см. листинг 3.24). Попытки их использования обычно приводят к возникновению ошибок;
- каждое вещественное число будет иметь приблизительно одинаковое количество *значащих цифр* в его представлении (см. листинг 3.19). Как следствие этого, ошибка для очень больших чисел будет больше по абсолютной величине;
- представители вещественных чисел неравномерно распределены внутри диапазона значений. Их плотность уменьшается с увеличением абсолютного значения числа.

Вещественные числа представлены приближенно, следовательно, арифметические действия (см. табл. 3.1) над ними также выполняются *приближенно.*

Из изложенного следует несколько простых правил:

- вещественные числа нежелательно проверять на строгое равенство;
- необходимо проявлять осмотрительность при преобразовании вещественных чисел в целые и избегать вычитания почти равных чисел, т. к. могут возникнуть ошибки из-за потери многих значащих цифр;
- для уменьшения влияния ошибки округления при выполнении арифметических операций с вещественными числами необходимо иметь в виду следующее. Если складывается много чисел, то их нужно *разбить на группы чисел, близких по абсолютному значению*, произвести суммирование в группах, *начиная с меньшего числа*, после чего полученные суммы сложить, опять-таки начиная *с меньшей.* По аналогии с предыдущим получаются оценки для других арифметических операций и соответствующие практические рекомендации (см. листинг 3.29).

Вещественные числа в шестнадцатеричной системе счисления записывать нельзя.

Запись символов. Специальные и управляющие символы

В том случае, если в программе требуется использовать значение символьной переменной или константы, его необходимо заключить в апострофы или записать с использованием знака `#`, за которым следует код символа.

Например, `'A'` обозначает букву А, `';'` — точку с запятой, `' '` — пробел, `#32` или `#$20` являются также символом пробела (32 — это код, соответствующий пробелу, а шестнадцатеричное число 20 равно десятичному 32).