№ 1475

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
ИНСТИТУТ СТАЛИ и СПЛАВОВ
Технологический университет
МИСиС

Г.И. Светозарова, Е.В. Сигитов

ПРОГРАММИРОВАНИЕ И АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

Программирование
на языках Турбо-Паскаль и Си

Лабораторный практикум

МОСКВА 2002

№ 1475

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
ИНСТИТУТ СТАЛИ и СПЛАВОВ
Технологический университет
МИСиС

Кафедра инженерной кибернетики

Г.И. Светозарова, Е.В. Сигитов

ПРОГРАММИРОВАНИЕ И АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

Программирование на языках Турбо-Паскаль и Си

Лабораторный практикум
для студентов всех специальностей

Переиздание

Рекомендован редакционно-издательским советом института

МОСКВА 2002

УДК 681.3.06.(075.8)
С24

С24 *Светозарова Г.И., Сигитов Е.В.* Программирование и алгоритмические языки: Программирование на языках Турбо-Паскаль и С Лаб. практикум / Переиздание – М.: МИСиС, 2002. – 149 с.

Лабораторный практикум предназначен для выполнения в курсе "Программирование и алгоритмические языки". Ориентирован на использование языков Турбо-Паскаль 7.0 и Си. Содержит краткие сведения по языкам и лабораторные работы, ориентированные на приобретение навыков использования основных средств языка, приемов программирования, типовых алгоритмов, структур данных и методов сортировки. Может быть также использован в курсах программирования для других специальностей и для самостоятельного изучения основ программирования и алгоритмических языков Турбо-Паскаль и Си.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ. ОСНОВНЫЕ ПОНЯТИЯ ПРОГРАММИРОВАНИЯ

Алгоритм — четкое описание последовательности действий, которые необходимо выполнить для решения задачи. Разработать алгоритм решения задачи означает разбить задачу на последовательно выполняемые шаги (этапы), причем результаты выполнения предыдущих этапов могут использоваться при выполнении последующих. При этом должны быть четко указаны как содержание каждого этапа, так и порядок выполнения этапов. Отдельный этап (шаг) алгоритма представляет собой либо другую (всегда более простую) задачу, либо должен быть достаточно простым и понятным без дополнительных пояснений. Если этап – другая задача, то к ней применяется тот же прием разбиения на более простые этапы. Это – так называемый *метод пошаговой детализации алгоритма*.

Программой называется обычно окончательный вариант решения задачи, ориентированный на конкретного исполнителя (в частности, компьютер).

При решении различных задач используется несколько типичных последовательностей действий, а именно

– *следование* – размещение отдельных этапов в порядке друг за другом;

– *цикл* – многократное выполнение одной и той же последовательности действий. Если количество повторений заранее задано, то говорят о *цикле по счетчику.* Если количество повторений неизвестно, но имеется условие окончания цикла, то говорят о *цикле по условию;*

– *разветвление* – выбор одного из двух вариантов вычислительного процесса после проверки условия;

– *обход* – частный случай разветвления, когда одна из ветвей не содержит никаких действий;

– *множественный выбор* – выбор одного из многих вариантов вычислительного процесса (обычно в зависимости от какой-либо целой величины).

Перечисленные последовательности называются *типовыми структурами алгоритма*. Каждая из них имеет один вход и один выход, и их можно объединять в любой последовательности, создавая программы ясной и прозрачной структуры.

Использование только перечисленных структур лежит в основе так называемого *структурного подхода к программированию,* или структурного программирования.

Подробнее о типовых структурах алгоритма и структурном подходе к разработке программ см.[2], глава 1.

Разработка языка Паскаль осуществлена с использованием идей структурного программирования. В нем имеются специальные конструкции, позволяющие представить каждую из перечисленных последовательностей действий (типовых структур алгоритма) в виде одного оператора. Собственно автор идей структурного программирования, Н. Вирт, является и автором первого варианта языка Паскаль. Этот язык в дальнейшем изменялся в соответствии с новыми возможностями вычислительной техники, но основные идеи остались в неприкосновенности.

Данное пособие ориентировано на версию Турбо-Паскаль 7.0.

Часть I. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ТУРБО-ПАСКАЛЬ. ОСНОВЫ ЯЗЫКА ТУРБО-ПАСКАЛЬ

1. Вводные понятия. Структура программы

Первый оператор – заголовок программы:

`program` *имя*; (не является обязательным)

Далее следует раздел описаний, где должны быть объявлены используемые в программе переменные, метки и т.д.

После раздела описаний следует тело программы, т.е. последовательность операторов программы, начинающаяся словом `begin` и заканчивающаяся словом `end`, после которого следует точка.

Таким образом, программа должна иметь следующую структуру:

```
program имя;
раздел описаний;
begin
раздел операторов;
end.
```

Для именования объекта программы, а также самой программы, подпрограмм и т.п. используются *идентификаторы*.

Идентификатор может включать буквы латинского алфавита (строчные и прописные буквы в идентификаторах неразличимы), цифры и символ подчеркивания. Первым символом обязательно должна быть буква. Количество символов может быть произвольным, значащими являются первые 63 символа. В качестве идентификаторов нельзя использовать так называемые зарезервированные слова (например, `begin`, `end`). Полный список зарезервированных слов см. например, в [1], Приложение А.

2. Переменные. Типы данных

Одним из основных объектов программы является *переменная*. Под переменную выделяется ячейка памяти, в которую можно поместить какое-либо значение (*присвоить* переменной значение). В Турбо-Паскале присваивание осуществляется при помощи *оператора присваивания*, например,

a := 5,

Каждая переменная имеет тип. *Тип* определяет объем памяти, выделяемый под переменную (*длину ячейки памяти*), множество допустимых значений, способ преобразования информации из внешнего представления во внутреннее и из внутреннего во внешнее и набор операций, допускаемых для переменной. Основные принципы представления информации в памяти см. [2], с. 13–16.

В языке Турбо-Паскаль есть возможность использовать стандартные (базовые) типы или определить в программе новый тип.

К *стандартным типам* относятся целые типы (`shortint` – занимает в памяти 1 байт, `byte` – 1 байт, `word` – 2 байта, `integer` – 2 байта, `longint` – 4 байта), вещественные типы (`single` – 4 байта, `real` – 6 байт, `double` – 8 байт, `extended` – 10 байт, `comp` – 8 байт), символьный тип (`char` – 1 байт), логический тип (`boolean` – 1 байт), тип-строка (`string`), тип- указатель (`pointer`), текстовый файл (`text`), (в Турбо-Паскале 7.0 еще ASCIIZ-строка – `pchar`).

Перечисленные стандартные типы можно использовать для описания переменных непосредственно. Например,

```
var
  x,y:real;
  n:integer;
```

Определяемые в программе типы должны быть объявлены в разделе объявления типов (см. пп. 4, 10), после чего их можно использовать для описания переменных.

Константы могут задаваться в программе своими непосредственными значениями, либо идентификаторами, которым в разделе описаний поставлены в соответствие конкретные значения.

Целые константы – последовательность цифр со знаком +, – или без знака (5, –18,...).

Вещественные константы могут содержать дробную часть. В программе записываются одним из двух способов:

– используя точку для отделения целой части от дробной (0.5, 5.1, –6.0);

– используя показательную форму записи (например, 2.3E–5 соответствует обычной записи $2{,}3 \cdot 10^{-5}$, 0.8E2 соответствует записи $0{,}8 \cdot 10^{2}$, ...).

Имя константы можно задать в разделе описания констант. Например,

```
const
  n=12;
  a=0.5;
```

Если при описании константы указать ее тип (описать *типизированную константу*), то ее имя можно использовать в программе как обычную переменную и даже изменять ее значение. Например,

```
const
  s:integer=0;
```

Примеры описания и использования констант см., например, в программах program p1_1, program p1_1_1 (Лабораторная работа 1).

Выражения определяют правила для получения значения. *Арифметические выражения* включают переменные, константы вещественных или целых типов и стандартные (или определяемые пользователем, см. п 8 3) функции. В арифметических выражениях используются операции +, –, *, /.

Результат любой из этих операций будет целый, если оба операнда целые, в остальных случаях результат вещественный. Исключение составляет операция / Результат её выполнения всегда вещественный. Кроме того, для целых операндов могут использоваться операции `div` (деление нацело) и `mod` (вычисление остатка от деления).

Заметим, что в Турбо-Паскале отсутствует операция возведения в степень Возведение в любую целую степень можно легко осуществить, используя стандартную функцию `sqr`. Так для вычисления $y = x^7$ можно использовать следующую последовательность операторов

```
p:=sqr(x);
p1:=sqr(p);
y:=p*p1*x;
```

Для возведения в вещественную степень можно использовать сочетание exp и ln Так например, $y = x^p$ (при x>0) можно вычислить, используя оператор

```
y:=exp(p*ln(x));
```

3. Операторы

3 1. *Оператор присваивания*

Общий вид оператора

переменная = выражение,

Примеры:

```
y:=5.6;
y:=x/p*v;
y:=sqrt(x*sin(x));
```

Второй пример соответствует вычислению выражения $y = \frac{x}{p} \cdot V$.

З а м е ч а н и е. Если переменная левой части целого типа, то выражение правой части должно иметь целое значение.

3.2 *Операторы ввода-вывода*

Операторы ввода-вывода являются операторами вызова стандартных процедур модуля System (см п 11). Здесь рассматривается только ввод с клавиатуры и вывод на экран дисплея.

Ввод с клавиатуры осуществляется операторами

```
read (список ввода);
readln (список ввода);
```

где *список ввода* может содержать переменные целого или вещественного типа, типа `char`, `string` или `pchar`, в которые помещается вводимая информация. Отдельные значения при наборе отделяются друг от друга пробелами. После выполнения `readln` осуществляется переход к следующей строке.

Вывод на экран дисплея осуществляется операторами

```
write (список вывода);
writeln (список вывода);
```

где *список вывода* может включать переменные, перечисленные для *списка ввода,* и переменные логического типа, а также выражения (в частности, константы) перечисленных типов.

Логические значения выводятся в виде `true, false`.

После выполнения оператора `writeln` осуществляется перевод строки.

Элемент списка вывода может включать формат, т.е размер поля (количество позиций), в которые должен быть осуществлен вывод,

ля вещественных чисел указываются еще и количество позиций под юбную часть. Например,

```
var
  n:integer;
  x:real;
    .  .  .
  writeln('n=',n:4;'x=',x:8:2);
```

На экран будет выведено n= и далее в четыре позиции значение , затем x= и в восемь позиций значение x, из них дробная часть азмещается в последних двух позициях.

Если указанного поля не хватает, то формат игнорируется

3.3 *Оператор безусловного перехода*

Оператор goto позволяет изменить порядок выполнения операторов и перейти в любую точку программы Оператор, на который осуществляется переход, должен быть помечен меткой. *Меткой* может быть либо набор от 1 до 5 цифр, либо идентификатор. Метки должны быть описаны Метка от оператора отделяется двоеточием. Например,

```
label 10,20,30, pom;
    .  .  .
goto 10;
    .  .  .
10:p:=1;
    .  .  .
goto pom;
    .  .  .
pom:p:=2;
    .  .  .
```

На практике оператор `goto` используется крайне редко Современное программирование считается "программированием без `goto`". Использование оператора `goto` оправдано лишь в редких случаях, когда использование только типовых структур алгоритмов приводит к неоправданному усложнению программы.

3 4. *Пустой оператор*

Пустой оператор никак не изображается Он не выполняет никаких действий, но он может быть помечен и отделяется точкой с запя-

той. Использование помеченного пустого оператора позволяет, в частности, перейти на конец программы. Например,

```
. . .
label nx;
begin
. . .
  goto nx;
 . . .
  nx:
end.
```

Ниже приводится пример программы, использующей операторы присваивания и ввода-вывода:

```
program p;
var
  x,y,z,f:real;
const
  m:=10;
begin
  readln(x,y);
  z:=x+y;
  f:=y/x*m;
  writeln('z=',x:8:2,'f=',f:8:3)
end.
```

З а м е ч а н и е. В программе каждое предложение (в частности, оператор) заканчивается ;. Перед `end` ; ставить не обязательно.

Далее рассматриваются операторы, реализующие типовые (базовые) структуры алгоритма.

3.5 *Составной оператор*

Составной оператор реализует структуру "следование". Составной оператор – это последовательность операторов, заключенная в операторные скобки `begin, end`. Например,

```
begin x:=2; y:=3 end;
```

В этом смысле часть программы, представляющая раздел операторов, является одним составным оператором.

3.6 *Условный оператор*

Условный оператор реализует структуру "разветвление". Общий вид оператора

```
if условие then оператор 1 else оператор 2;
```

Здесь *условие* в простейшем случае может быть задано как отношение (в общем случае – логическое выражение, см п. 6). *Отношение* – это два арифметических выражения, соединенных знаком операции отношения <, <=, >, >=, =, <>. *Оператор 1, оператор 2* – любые операторы Например,

```
if i<=10 then x:=5 else x:=0;
```

Возможна усеченная форма условного оператора

if *условие* then *оператор,*

реализующая структуру "обход". Например,

```
if i>10 then writeln ('конец');
```

Если *условие* удовлетворяется, то выполняется *оператор*. В противном случае осуществляется переход к следующему оператору программы.

Примеры программ с использованием условного оператора см. в Теоретическом введении к Лабораторной работе 2.

3 7 *Операторы цикла*

Оператор цикла `for, while` и `repeat` реализуют циклы по счетчику и по условию *Оператор цикла* `for` имеет общий вид

`for` *переменная* := ih `to` ik `do` *оператор;*

или

`for` *переменная* := ih `downto` ik `do` *оператор,*

где *переменная* – управляющая переменная цикла – целого типа (в общем случае порядкового типа, см. п 10), ih, ik – начальное и конечное значения управляющей переменной цикла – константы, переменные или выражения, совместимые для присваивания с управляющей переменной цикла, *оператор* – оператор, выполняемый многократно (в цикле) Шаг изменения управляющей переменной не указывается и равен 1 (в случае использования `to`) и –1 (в случае использования `downto`) Если ih<ik, то цикл не выполняется ни разу (для случая `to`). Оператор цикла `for` реализует структуру "цикл по счетчику".

Оператор цикла `while` имеет общий вид

`while` *условие* `do` *оператор,*

где *условие* имеет тот же смысл, что и в условном операторе.

Оператор выполняется столько раз, сколько раз при проверке *условие* удовлетворяется.

Оператор цикла `repeat` имеет общий вид:

```
repeat
 оператор 1; оператор 2; .
until условие;
```

Здесь *оператор 1, оператор 2, .* – тело цикла выполняются многократно, *условие* – условие выхода из цикла В отличие от оператора цикла `while` первая проверка *условия* осуществляется после выполнения последовательности операторов (тела цикла), т е. эта последовательность всегда выполняется хотя бы один раз.

Операторы цикла `while` и `repeat` могут использоваться для реализации цикла по условию.

С помощью этих операторов могут быть организованы и циклы по счетчику, однако при этом необходимо самому обеспечивать изменение управляющей переменной цикла

Примеры программ с использованием операторов цикла см. в Теоретическом введении к Лабораторной работе 1.

Операторы `break, continue` являются операторами вызова стандартных процедур. Их использование позволяет досрочно выйти из цикла, не дожидаясь выполнения условия выхода (`break`) или начать новую итерацию, даже если предыдущая итерация не завершена (`continue`)

3.8. *Оператор выбора*

Оператор выбора case обеспечивает реализацию структуры "множественный выбор". Он имеет вид

```
case s of
  c1:оператор1;
  c2:оператор2;
     .  .  .
  cn:оператор n;
  else оператор
end;
```

Здесь s – выражение целого (в общем случае порядкового) типа, значение которого вычисляется; c1, c2,. , cn – константы, с которыми сравнивается значение s; *оператор1, оператор2, . , оператор n* – операторы, из которых выполняется тот, для которого значение s совпадает с константой; *оператор* выполняется, если значение s не сов-