Министерство общего и профессионального образования
Российской Федерации
Томский государственный университет

УТВЕРЖДАЮ
Декан ФТФ
_____Э. Р. Шрагер
«____» _____1999 г.

ПОДПРОГРАММЫ В ПАСКАЛЕ

Методические указания

Томск -1999

РАССМОТРЕНЫ И ОДОБРЕНЫ

кафедрой МДТТ
физико-технического факультета

Зав. кафедрой,
доцент /В.И.Масловский/

Рассмотрены и утверждены методической комиссией физико-технического факультета ТГУ

Протокол № от 1999г.
Председатель комиссии,
профессор, доктор физ. - мат. наук / В. А. Скрипняк/

Методические указания содержат основные сведения по описанию и использованию подпрограмм в основном в программах блочной структуры алгоритмического языка ТУРВО ПАСКАЛЬ 7.0 в интегрированной среде Turbo Pascal для операционной системы MS-DOS.

Методические указания предназначены для студентов, обучающихся по программе высшего базового образования по направлениям : 553100 - «Техническая физика», 553300 - «Прикладная механика» в Томском государственном университете.

СОСТАВИТЕЛЬ: Н.И. Лужанская, ст. преподаватель.

Рецензент: В. Г. Прокофьев, доцент, канд. физ. - мат. наук.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. **Что такое подпрограмма? Виды подпрограмм в Паскале. В чём эффективность использования подпрограмм?**
2. **Изобразите синтаксические диаграммы (описания) объявления подпрограммы - процедуры и подпрограммы - функции.**
3. **В чём состоит отличие описания процедуры и функции?**
4. **Виды параметров в подпрограммах.**
5. **Каковы отличия параметров - значений от параметров - переменных, особенности их описания и применения.**
6. **Какие параметры называются формальными и какие — фактическими?**
7. **Соотношения между формальными и фактическими параметрами.**
8. **Способы передачи значений фактических параметров в подпрограмму.**
9. **При каком условии можно выполнить описание процедуры без параметров и в чем состоит особенность этого описания?**
10. **Локальные и глобальные переменные. Области действия идентификаторов при сложной структуре программы.**
11. **Каковы основные правила определения области действия для идентификаторов подпрограмм?**
12. **Сущность рекурсивных подпрограмм и их отличия от обычных подпрограмм.**
13. **Для чего предназначены директивы FAR и FORWARD?**
14. **В каких случаях в программе указывается директива компилятору {$I}?**
15. **Процедурные типы.**
16. **Обращение к подпрограммам - процедурам и подпрограммам - функциям.**
17. **Опишите последовательность событий при вызове процедуры и функции.**

1. ПОДПРОГРАММЫ

Подпрограмма — это обособленная часть программы , оформленная в виде отдельной синтаксической конструкции (программной единицы) и снабжённая собственным именем.

Подпрограммы предназначены для того, чтобы избежать повторных записей часто встречающихся однотипных участков алгоритмов.

Все подпрограммы языка Турбо Паскаль делятся на две группы: стандартные (библиотечные, встроенные) и определяемые пользователем.

Стандартные подпрограммы входят в состав языка, хранятся в специализированных библиотечных модулях (System, Crt, Dos, Graph, Overlay, Printer, Turbo Vision) и вызываются для выполнения по строго фиксированному имени. Подпрограммы из модуля System автоматически доступны любой программе или модулю, так как этот модуль подключается по умолчанию, а все остальные в случае необходимости должны подключаться программистом с помощью указания имени соответствующего модуля в предложении uses (разделе модулей). Например: uses Crt, Graph.

Собственные подпрограммы создаются и именуются самим пользователем. Они описываются только один раз, но допускают многократное обращение к ним из различных точек программы.

Использование подпрограмм позволяет:

- **разделить некоторую сложную задачу на несколько простых меньших по объёму и сложности задач (структурировать программу);**
- **экономить время программирования;**
- **снизить объём и трудоёмкость программы;**
- **улучшить читаемость и наглядность программы;**
- **уменьшить вероятность и количество ошибок;**
- **облегчить процесс отладки программы;**
- **повысить надёжность и эффективность программ.**

Подпрограммы являются одним из основных средств структурирования программ языка программирования Турбо Паскаль.

2. ПРОЦЕДУРЫ И ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ

В Турбо Паскале реализованы подпрограммы - функции и подпрограммы - процедуры.

Подпрограмма - процедура (процедура) используется тогда, когда надо в подпрограмме получить несколько результатов (хотя процедуру можно применять и в тех случаях, когда выходной результат лишь один)
или же выполнить какую-либо обработку данных, не возвращая в вызывающую программу ни одного результата (например, ввести или вывести многомерный массив). Если же в результате выполнения подпрограммы необходимо получить одно значение в качестве результата, то оформляется подпрограмма - функция (функция).

Подпрограммы состоят из данных, внутренних процедур и функций и операторов. В языке Паскаль допускается любой уровень вложенности подпрограмм.

Структура процедур и функций аналогична структуре основной программы на языке Паскаль. Подпрограмма состоит из заголовка и блока, включающего раздел описаний (объявлений) и раздел действий (операторную часть, называемую телом подпрограммы), и заканчивается в отличие от основной программы точкой с запятой.

Для того, чтобы воспользоваться подпрограммами программисту надо знать, как они описываются (объявляются) и как используются (т.е. как обратиться к ним).

Описание процедур и функций в программе располагается между разделами переменных и операторов и называют это место разделом подпрограмм.

Рассмотрим особенности описания и использования процедур и функций.

2.1. ОПИСАНИЕ ПРОЦЕДУРЫ. ОПЕРАТОР ПРОЦЕДУРЫ

Описание процедуры начинается с заголовка, состоящего из служебного слова PROCEDURE, за которым указывают имя процедуры , а в круглых скобках — список формальных параметров со своими описаниями, и заканчивающегося точкой с запятой. Процедуры могут быть и без параметров, тогда в заголовке указывается только её имя.

Формальными параметрами называются параметры, указываемые в заголовке подпрограммы (процедуры или функции) при её описании. Они используются только для описания подпрограммы, а при вызове подпрограммы заменяются на фактические параметры, которые могут иметь совершенно другие имена.

Общая форма записи заголовка процедуры имеет вид:

PROCEDURE <имя процедуры>(<список формальных параметров>);

Имя процедуры — идентификатор, уникальный в пределах программы. Через параметры осуществляется передача исходных данных в процедуру и результатов работы обратно в ту программу, которая вызывала её.

Список формальных параметров может включать в себя параметры-константы (перед ними должно стоять служебное слово CONST, используются только в версии Турбо Паскаля 7.0), параметры - значения,

параметры - переменные (перед ними должно стоять служебное слово VAR), параметры - процедуры и параметры - функции.

Параметры различаются :

1. По механизму передачи:

а)передача по значению(VALUE); б)передача по адресу(по ссылке) (ADDR).

2. По взаимодействию вызывающей и вызываемой процедур/функций :

а) только как входной параметр (IN);

б) только как выходной параметр (OUT);

в) как входной, так и как выходной параметр (INOUT).

Из шести теоретически возможных способов передачи параметров в Турбо Паскале 7.0 реализованы три: VALUE IN, ADDR IN, ADDR IN-OUT. Механизм передачи параметров рассмотрим далее.

После заголовка процедуры следуют разделы в том же порядке, что и в программе.

Структура описания процедуры имеет следующий вид:

```
PROCEDURE <имя >(<список формальных параметров>);
label
const          Описания локальных меток, констант,
type           типов и переменных
var
function       Описания внутренних
procedure      функций и процедур
begin
 .............
 .............  Операторы ( раздел действий )
end;
```

Пример описания процедуры MI нахождения минимального элемента и его номера в одномерном массиве:

```
procedure mi(x:bb;k:integer;var mini,ks:integer);
var i:integer;
begin
  mini:=x[1];ks:=1;
  for i:=2 to k do
     if x[i]<mini then
        begin mini:=x[i];ks:=i  end
end;
```

Видно, что процедура имеет формальные параметры - значения :x — массив типа bb, который должен быть описан в программе в разделе type: type bb=array[1..n] of integer; (в заголовке программы нельзя

указывать составной (новый) тип) и k - переменная типа integer для задания длины массива; формальные параметры - переменные: mini - для минимального элемента и ks - для его номера, оба целого типа. В этой процедуре входными параметрами являются x и k, а выходными mini и ks. Параметры, через которые передаются результаты , всегда в списке формальных параметров должны быть описаны как параметры - переменные.

Помимо формальных параметров, эта процедура содержит переменную i, которая необходима для организации цикла. Эта переменная является локальной для процедуры MI.

Порядок перечисления формальных параметров в заголовке процедуры несуществен, но лучше если сначала перечисляются входные параметры (аргументы), а затем выходные (результаты).

Имени процедуры никакого значения не присваивается. Оно нужно для того, чтобы различать разные процедуры.

Следует отметить, что в теле процедуры доступны любые глобальные константы и переменные за исключением тех, имена которых совпадают с именами её локальных переменных.

Идентификаторы называются локальными, если допускается их использование в пределах одной подпрограммы, а идентификаторы, действие которых распространяется на несколько вложенных (не менее одной) подпрограмм, называются глобальными. Область действия идентификаторов определяется местом их описания, т.е. понятия «локальные» и «глобальные» следует понимать относительно — по отношению к конкретной подпрограмме. Так как в Паскале допускается любой уровень вложенности подпрограмм, то для сложных программ, у которых, например, процедура, описанная в основной программе, в свою очередь, имеет описания внутренних процедур или функций и т. д., существуют правила локализации имён, определяющие область действия любого имени.

1. **Любое имя (константы, типа, переменной, процедуры или функции) определено только в пределах той подпрограммы, в которой оно объявлено (описано), а область действия распространяется на все внутренние процедуры и функции.**
2. **Одно и то же имя может быть определено в каждой процедуре, функции или программе. Тогда областью действия этого имени является подпрограмма или вся программа, в которой описан объект с данным именем, за исключением внутренних подпрограмм, содержащих описание объекта с таким же именем. Другими словами: при совпадении имен локального и глобального идентификаторов в подпрограмме**

будет действовать только внутренний локальный идентификатор, внешний как бы не виден.

Описание процедуры не является приказом выполнить действия, а представляет собой лишь описание этих действий. Чтобы выполнить действия, описанные в процедуре, надо обратиться к ней. Обращение к процедуре (вызов процедуры) и выполнение процедуры осуществляется при помощи **оператора процедуры**:

<имя процедуры>(<список фактических параметров>);

Как видим, чтобы вызвать процедуру, в вызывающей программе достаточно написать имя этой процедуры вместе с заключённым в скобки списком фактических параметров, каждому из которых предстоит заменить соответствующий формальный параметр, например: mi(c,m,b[i],ns[i]); или mi(b,n,min,l); — обращения (вызовы) описанной выше процедуры.

Фактическими параметрами называются параметры, указываемые при вызове подпрограммы (процедуры или функции) . Это могут быть константы, переменные, а в отдельных случаях и выражения, к которым применяют процедуру.

Между формальными и фактическими параметрами должно быть полное согласование (соответствие) по количеству, порядку следования и типу.

При вызове процедуры сначала передаются параметры, при этом параметры - значения передаются по значению, а параметры - константы и параметры - переменные по ссылке.

Встретив в вызывающей программе оператор процедуры, компьютер осуществляет следующие действия:

1. Разыскивается описание процедуры с тем же именем, что указано в операторе.
2. Все формальные параметры - значения и параметры - константы приобретают значения соответствующих фактических параметров непосредственно перед выполнением тела процедуры.
3. Формальные параметры - переменные везде в теле процедуры заменяются соответствующими фактическими параметрами.
4. После этого выполняется модифицированное тело процедуры так, как если бы оно было написано на том месте, где находится оператор процедуры.
5. По окончании выполнения тела процедуры управление передаётся оператору, следующему за выполненным оператором процедуры.

Ниже приведена программа, в которой используется процедура поиска минимального элемента и его номера в одномерном массиве для решения задачи:

Пример 1. Составить программу с использованием подпрограммы определения минимального элемента и его номера в одномерном массиве для определения сначала минимумов строк матрицы a(5*3), а затем и всей матрицы.

```
program min53;
const n=5;m=3;
type bb=array[1..n] of integer;
    aa=array[1..n,1..m] of integer;
const a:aa=((3,1,2),(5,2,4),(6,7,3),(-9,4,8),(7,0,8));
var i,j,min,l:integer;
    ns,b,c:bb;
procedure mi(x:bb;k:integer;var mini,ks:integer);
var i:integer;
begin
  mini:=x[1];ks:=1;
  for i:=2 to k do if x[i]<mini then
    begin
      mini:=x[i];ks:=i
    end
end;
begin
  for i:=1 to n do
    begin
      for j:=1 to m do c[j]:=a[i,j];
      mi(c,m,b[i],ns[i])
    end;
  mi(b,n,min,l);
  writeln('str=',l:2,' stl=',ns[l]:2,' min=',min:3)
end.
```

2.2. ОПИСАНИЕ ФУНКЦИИ. УКАЗАТЕЛЬ ФУНКЦИИ

Подпрограмма - функция позволяет получить только один результат. У описания функции три основных отличия от описания процедуры:

1. **В заголовке другое служебное слово, а именно, function.**

2. **После указания в круглых скобках списка формальных параметров через двоеточие записывается тип функции. Это связано с тем, что у функции единственный выходной результат возвращается через имя функции, а потому среди формальных параметров нет параметра для результата.**
3. **Поэтому среди входящих в функцию операторов должен обязательно присутствовать хотя бы один оператор присваивания, в левой части которого стоит имя данной функции.**

Общая вид заголовка функции имеет вид:

function <имя >(<список формальных параметров>):<тип результата>;

Пример описания функции MI для нахождения минимального элемента в одномерном массиве:

```
type bb=array[1..n] of integer;
function mi(x:bb;k:integer):integer;
var i,mm:integer;
begin
  mm:=x[1];
  for i:=2 to k do if x[i]<mm then mm:=x[i];
  mi:=mm
end;
```

Функция имеет формальные параметры - значения :x — массив типа bb, который должен быть описан в программе в разделе type, и k - переменная типа integer для задания длины массива; локальные параметры i, mm, описанные внутри функции. Вспомогательная переменная mm нужна потому, что в условии x[i]<mm компилятор обнаружит имя функции, будет трактовать его как вызов функции и выдаст сообщение об ошибке, связанной с отсутствием параметров вызова функции.

Обращение к функции оформляется в виде операнда. Для этого в выражении (или операторе вывода) записывается (указатель функции) имя функции со списком фактических параметров, которые должны совпадать по количеству, порядку следования и типам с формальными параметрами функции. Примеры вызова описанной выше функции:

```
b[i]:=mi(c,m);  min:=mi(b,n);
```

После выполнения функции выработанный ею результат используется в качестве значения указателя функции в том выражении, в кото-

рое входит этот указатель. Передача фактических параметров при вызове функции производится так же, как и при вызове процедуры.

Ниже приведена программа, в которой используется функция поиска минимального элемента в одномерном массиве для решения задачи:

Пример 2. Составить программу с использованием подпрограммы определения минимального элемента одномерного массива для определения сначала минимумов строк матрицы a(5*3), а затем и всей матрицы.

Вариант 1 программы с подпрограммой - функцией:

```
program min1f53;
const n=5;m=3;
type bb=array[1..n] of integer;
    aa=array[1..n,1..m] of integer;
const a:aa=((3,1,2),(5,2,4),(6,7,3),(9,4,8),(7,0,8));
var i,j,min:integer;
     b,c:bb;
function mi(x:bb;k:integer):integer;
var i,mm:integer;
begin
 mm:=x[1];
 for i:=2 to k do if x[i]<mm then mm:=x[i];
 mi:=mm
end;
begin
 for i:=1 to n do
   begin
     for j:=1 to m do c[j]:=a[i,j];
     b[i]:=mi(c,m);
   end;
 min:=mi(b,n);
 writeln('min=',min:3)
end.
```

Чтобы нагляднее показать отличие функции от процедуры, представлена программа решения той же задачи с применением процедуры для поиска минимального элемента в одномерном массиве:

Вариант 2 программы с подпрограммой - процедурой:

```
program min53;
const n=5;m=3;
type bb=array[1..n] of integer;
    aa=array[1..n,1..m] of integer;
const a:aa=((3,1,2),(5,2,4),(6,7,3),(9,4,8),(7,0,8));
var i,j,min:integer;
      b,c:bb;
procedure mi(x:bb;k:integer;var mini:integer);
var i:integer;
begin
  mini:=x[1];
  for i:=2 to k do if x[i]<mini then mini:=x[i]
end;
begin
  for i:=1 to n do
    begin
      for j:=1 to m do c[j]:=a[i,j];
      mi(c,m,b[i])
    end;
  mi(b,n,min);
  writeln('min=',min:3)
end.
```

3. ПАРАМЕТРЫ ПОДПРОГРАММ И МЕХАНИЗМ ИХ ПЕРЕДАЧИ

Все формальные параметры, задаваемые в заголовках подпрограмм, разбиваются на группы (подсписки) однотипных элементов, которые отделяются друг от друга точками с запятыми. В группах имена отделяются запятыми. Тип указывается через двоеточие после перечисления идентификаторов.

Все формальные параметры можно разбить не следующие группы:

- **параметры-константы (перед ними должно стоять служебное слово CONST, используются только в версии Турбо Паскаля 7.0);**
- **параметры - значения (в основной программе эти параметры подпрограммой не могут быть изменены);**

- **параметры - переменные (перед ними должно стоять служебное слово VAR, в основной программе эти параметры подпрограммой могут быть изменены);**
- **параметры - процедуры и параметры - функции (процедурного типа).**

3.1. ПАРАМЕТРЫ - ЗНАЧЕНИЯ

Параметры - значения реализуют способ передачи параметров вида VALUE IN. Эти параметры в заголовке программы указываются своими именами и через двоеточие - типы. Перед их идентификаторами никаких дополнительных ключевых слов не ставится, например: function mi(x:bb; k:integer):integer; — здесь x и k - параметры - значения.

При обращении к подпрограмме в качестве фактического параметра - значения могут использоваться переменные, константы или выражение совместимого для присваивания типа, кроме файлового типа и типа, опирающегося на файловый, например: min:=mi(b,n);

Значения фактических параметров, соответствующих параметрам - значениям, в ячейки памяти формального параметра передаются через стек в виде их копий, поэтому в вызывающей программе они остаются неизменными.

Вообще входные параметры, если они не должны изменяться в результате выполнения подпрограммы, лучше задавать как параметры - значения. Этот способ более безопасен, так как значения фактических параметров не будут случайным образом изменены.

3.2. ПАРАМЕТРЫ - ПЕРЕМЕННЫЕ

Параметры - переменные реализуют способ передачи параметров вида ADDR INOUT. Эти параметры в заголовке программы указываются своими именами и через двоеточие - типы. Перед их идентификаторами ставится ключевое слово VAR, действие которого распространяется до ближайшей точки с запятой, например:

```
procedure mi(x:bb; k:integer;  var mini,ks:integer);
```

Здесь mini, ks - параметры - переменные.

При обращении к подпрограмме в качестве фактического параметра - переменой могут использоваться переменные любого типа, включая файловые и опирающиеся на файловый. Но не допускается использование констант, например: mi(c,m,b[i],ns[i]).

Значения фактических параметров, соответствующих параметрам - переменным, передаются через стек своими адресами в порядке, объявленном в заголовке подпрограммы. Поэтому изменение параметра -

переменной в подпрограмме является, по существу, изменением фактического параметра в вызывающей программе.

3.3. ПАРАМЕТРЫ - КОНСТАНТЫ

Параметры - константы реализуют способ передачи параметров вида ADDR IN. Эти параметры в заголовке программы указываются своими именами и через двоеточие - типы. Перед их идентификаторами ставится ключевое слово CONST, действие которого распространяется до ближайшей точки с запятой, например:

```
procedure mi( const x:bb; k:integer;  var mini,ks:integer);
```

Здесь x - параметр -константа.

При обращении к подпрограмме в качестве фактического параметра - константы могут использоваться как переменные, так и константы любых типов, кроме файлового типа и типа, опирающегося на файловый, например: mi(c,m,b[i],ns[i]).

Формальным параметрам - константам запрещается выполнять присваивания и передавать их в качестве фактических параметров другим подпрограммам.

Параметры - константы используются тогда, когда надо передавать структуры данных большого размера (запись, массив и т. д.), но изменять их значения нельзя. При этом экономно используется оперативная память, меньше вероятность переполнения стека и сохраняются исходные данные.

3.4. ПАРАМЕТРЫ - ПРОЦЕДУРЫ И ПАРАМЕТРЫ - ФУНКЦИИ

Во многих подпрограммах надо в качестве параметров использовать имена других подпрограмм (функций, процедур), например, при вычислении интеграла, когда вид подынтегральной функции заранее неизвестен.

В этих случаях используют параметры процедурного типа. Это не должны быть стандартные и вложенные подпрограммы. Фактически параметры процедурного типа являются параметрами - значениями, так как они записываются без служебного слова VAR.

Процедурный тип представляет собой заголовок подпрограммы со всеми параметрами, но без имени.

В качестве фактических параметров в таких случаях используются соответствующие подпрограммы, имеющие необходимое число параметров требуемых типов.

Для использования процедурных типов надо:

1. **Описать процедурный тип.**
2. **Указать дальнюю модель вызова: а) глобальной директивой компилятора {$F+} или б) директивой FAR, которая эквивалентна ключу компилятора {$F+}, но её действие в отличие от ключа распространяется только на одну подпрограмму.**
3. **Описать подпрограмму.**

Пример 3 программы, иллюстрирующей применение процедурных типов.

Вычислить $S_1 = \sum_{1}^{100} \frac{x^2}{x+4}$; $S_2 = \sum_{1}^{200} (6x^2 + 8)$.

```
program proctyp;
type func=function(x:integer):real;
var s1,s2:real;
{$F+}
function f1(x:integer):real;
begin  f1:=x*x/(x+4) end;
function f2(x:integer):real;
begin  f2:=6*x*x+8 end;
{$F-}
function sum(n:integer;f:func):real;
var i:integer; s:real;
begin
  s:=0;
  for i:=1 to n do s:=s+f(i);
  sum:=s
end;
begin
  s1:=sum(100,f1); writeln('s1=',s1);
  s2:=sum(200,f2); writeln('s2=',s2)
end.
```

4. ПРОЦЕДУРЫ БЕЗ ПАРАМЕТРОВ

Обмен информацией между вызывающей программой и вызываемой подпрограммой может осуществляться только с помощью глобальных параметров.

Подпрограмма может оперировать глобальными параметрами двумя способами: или использовать механизм формальных параметров, или непосредственно обращаться к глобальным параметрам. Подпрограмма может использовать любые глобальные параметры кроме тех, имена которых совпадают с её локальными параметрами.

В тех случаях, когда в вызывающей программе и вызываемой подпрограмме используются одни и те же имена параметров (подпрограмма связана с главной программой посредством глобальных переменных), подпрограмму можно оформить без параметров.

Итак, подпрограмма без параметров — это подпрограмма, у которой отсутствует список формальных параметров в заголовке. У такой подпрограммы описание имеет вид:

```
PROCEDURE  <имя процедуры>;
 {или}
FUNCTION  <имя  функции> : <тип результата>;
  label
  const          Описания локальных меток, констант,
  type           типов  и  переменных
  var
  function       Описания  внутренних
  procedure      функций  и  процедур
  begin
   .............  Операторы ( раздел действий )
   .............
  end;
```

Из описания видно, что подпрограмма без параметров может быть составлена либо в виде функции, либо в виде процедуры.

В теле подпрограммы используются глобальные переменные и при необходимости локальные переменные.

Глобальные переменные обеспечивают передачу в тело подпрограммы без параметров исходных данных и передачу в вызывающую программу одного или нескольких результатов, которые после обращения к подпрограмме могут быть использованы в последующих действиях программы. Локальные переменные служат в теле подпрограммы вспомогательными переменными для получения промежуточных результатов.

Обращение к подпрограмме без параметров происходит всегда с одними и теми же параметрами и изменяются лишь их значения.

Обращение к процедуре без параметров осуществляется с помощью оператора процедуры, а к функции без параметров — указателя функции.

Пример 4. Составить программу для вычисления $t = \frac{s_1 + s_2 + s_3}{k_1 k_2 k_3}$, **где** s_i, k_i **— соответственно суммы и количества положительных элементов массивов X, Y, Z, состоящих из 30 элементов каждый.**

Предлагаемая программа демонстрирует применение подпрограммы - процедуры без параметров для вычисления суммы и количества положительных элементов массива.

```
program glpar;
const m=30;
type mas=array[1..m] of real;
var i,n:integer;
     t:real;x:mas;
     s:array[1..3] of real;
     k:array[1..3] of integer;
procedure ks;
var j:integer;
begin
  s[i]:=0;k[i]:=0;
  for j:=1 to m do if x[j]>0 then
    begin
      k[i]:=k[i]+1;
      s[i]:=s[i]+x[j]
    end
end;
begin
  for i:=1 to 3 do
    begin
      for n:=1 to m do read(x[n]);
      ks
    end;
    t:=(s[1]+s[2]+s[3])/(k[1]*k[2]*k[3]);
  writeln('t=',t:3)
```

```
end.
```

Если бы в этой программе задавались три разных имени для ввода элементов массивов X,Y, Z, то прежде чем обратиться к процедуре без параметров , надо было выполнять переименования параметров, чтобы обеспечить совпадение имен передаваемых параметров с именами глобальных переменных, используемых в теле процедуры.

5. РЕКУРСИВНЫЕ ПОДПРОГРАММЫ

Рекурсивная подпрограмма — это подпрограмма, в теле которой есть обращение к самой себе. Такое обращение возможно, так как при каждом обращении память под параметры и локальные переменные отводится в специально организованной области памяти (по принципу первый пришел — последним ушел: FILO), называемой программным стеком.

В результате рекурсивного обращения шаг за шагом будет образована группа копий тела подпрограммы. Программа должна оформляться таким образом, чтобы на конечном шаге появилось нерекурсивное решение.

Пример 5. Программы, использующие рекурсивное описание функции вычисления факториала.

```
program cochr;
var n:integer;
   fact:longint;
function f(k:integer):longint;
begin
 if k=1 then f:=1 else f:=k*f(k-1)
end;
begin
 write('Введи n:');
 readln(n);
 fact:=f(n);
 writeln('fact=',fact);
 readln
end.
```

```
program cochr;
```

```
var n:integer;
   fact:real;
function f(k:integer):real;
begin
  if k=1 then f:=1 else f:=k*f(k-1)
end;
begin
  write('Введи n:');
  readln(n);
  fact:=f(n);
  writeln('fact=',fact);
  readln
end.
```

Пример 6. Описание рекурсивной подпрограммы - процедуры для вычисления факториала.

```
procedure fact(k:integer; var f:real);
var p:real;
begin
  if k=1 then f:=1.0 else begin fact((k-1),p); f:=k*p end
end;
```

Применение рекурсивных подпрограмм позволяет получить более компактный текст программ, но выполняются такие программы, как правило, медленнее и возможно переполнение стека, особенно если у них много параметров. Кроме этого, следует избегать рекурсии там, где существует итерационное решение, требующее меньше времени для его получения.

6. ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ ПОДПРОГРАММ

Директивы дают дополнительную информацию транслятору о расположении подпрограмм.

6.1. ДИРЕКТИВА КОМПИЛЯТОРУ {$I <имя файла>}

Подпрограмма, которая написана и хранится в отдельном файле, во

время трансляции может быть включена компилятором в текст программы в качестве подпрограммы, если в разделе описаний подпрограмм записана директива компилятору {$I <имя файла>}.

Пример 7. Построить график функции $y = x^2 - 4$ в заданной области экрана.

```
program grafik;                                          { 1}
uses crt,graph;                                          { 2}
var i,dr,md:integer;                                     { 3}
   u,v,l,h:integer;                                      { 4}
   n,m,nx,ny,uo,vo:integer;                              { 5}
   xp,yp,xp1,yp1:integer;                                { 6}
   s,a,b,x,y,fm,am:real;                                 { 7}
{$I C:\workpas\grafika\fynk.tp7}                         { 8}
begin                                                    { 9}
 write('Введи u,v,l,h:');                                {10}
 readln(u,v,l,h);                                        {11}
 write(' Введи a,b,n:');                                 {12}
 readln(a,b,n);                                          {13}
 s:=(b-a)/n; x:=a; fm:=abs(f(x));                        {14}
 for i:=1 to n do                                        {15}
   begin                                                 {16}
     x:=x+s;                                             {17}
     if abs(f(x))>fm then fm:=abs(f(x))                  {18}
   end;                                                  {19}
 am:=abs(a);                                             {20}
 if abs(b)>am then am:=abs(b);                           {21}
 m:=trunc(h/(2*fm));                                     {22}
 if m>trunc(l/(2*am)) then m:=trunc(l/(2*am));           {23}
 dr:=detect;                                             {24}
 InitGraph(dr,md,'c:\turbo7\bgi');                       {25}
 if GraphResult <> grOk then Halt(1);                    {26}
 uo:=u+l div 2;vo:=v-h div 2;                            {27}
 line(u,vo,u+l,vo);                                      {28}
 line(uo,v,uo,v-h);                                      {29}
 nx:=round(l/m)-1; ny:=round(h/m)-1;                     {30}
 for i:=1 to nx do line(u+m*i,vo,u+m*i,vo-3);            {31}
 for i:=1 to ny do line(uo,v-m*i,uo+3,v-m*i);            {32}
 x:=a; y:=f(x);                                          {33}
```

```
  xp:=round(uo+m*x); yp:=round(vo-m*y);        {34}
  for i:=1 to n do                             {35}
    begin                                      {36}
      x:=x+s; y:=f(x);                         {37}
      xp1:=round(uo+m*x);                      {38}
      yp1:=round(vo-m*y);                      {39}
      line(xp,yp,xp1,yp1);                     {40}
      xp:=xp1; yp:=yp1                         {41}
    end;                                       {42}
  readln                                       {43}
end.                                           {44}

{ Файл с именем fynk.tp7 }
function f(x:real):real;
begin
  f:=x*x - 4
end;
```

6.2. ДИРЕКТИВА FORWARD

Для реализации взаимного обращения подпрограмм друг к другу (косвенный рекурсивный вызов), à òàêæå äëÿ óëó÷øåíèÿ ñòðóêòóðèðîâàííîñòè òåêñòà è ïîâûøåíèÿ åãî ÷èòàáåëüíîñòè используется директива FORWARD, которую называют опережающим описанием. При использовании этой директивы сначала записывается полный заголовок первой подпрограммы тело которой заменяется директивой FORWARD. Далее полностью описывается вторая подпрограмма, после которой полностью описывается первая подпрограмма, возможно с сокращенным заголовком, включающим служебное слово procedure или function и имя подпрограммы (список формальных параметров и тип результата в подпрограмме - функции не указывается).

Пример 8. Процедура P1 вызывает процедуру P2, а та, в свою очередь, процедуру P1.

```
Procedure P1(n: byte);
forward;
procedure P2(m: byte);
begin
  .....
```

```
  P1;
  ..........
end;
procedure P1;
begin
  ............
  P2;
  ..........
end;
```

Директива FORWARD применяется и для более удобного размещения подпрограмм: сначала описываются заголовки всех подпрограмм, а далее — сами подпрограммы.

7. ПРОГРАММЫ С ПОДПРОГРАММАМИ

Пример 9. Вычислить функцию

$$z = 5(3x^2 + 6x + 1)^2 + 7(3x^2 + 6x + 1) + 2.$$

```
program pprog1;
var a,b,c,x,z:real;
function f(a,b,c,x:real):real;
begin
  f:=a*sqr(x)+b*x+c
end;
begin
  write('Введи a,в,c,x:');
  readln(a,b,c,x);
  z:=f(a,b,c,x);
  write('Введи a,в,c:');
  readln(a,b,c);
  z:=f(a,b,c,z);
  writeln('z=',z)
end.
```

Пример10. Дана последовательность из не менее чем двух натуральных чисел, за которой следует 0. Вычислить сумму тех из них, порядковые номера которых — простые числа.

```
program N5t52Pil;
var n,m:integer;
   s:real;
function simple(n:integer):boolean;
{ В интервале [n div 2+1] не могут }
{ находиться делители числа n }
var j:integer;
begin
 if n=2 then simple:=true
 else if not odd(n) then simple:=false
 else
   begin
     j:=n div 2;
     if not odd(j) then j:=j+1;
     while n mod j<>0 do j:=j-2;
     simple:=j=1
   end
end;
begin
 write('Введи 1 число:');
 readln(m); n:=1; s:=m;
 repeat
   write('Введи число:');
   read(m); n:=n+1;
   if simple(n) then s:=s+m
 until m=0;
 writeln('s=',s)
end.
{F.e.1 2 2 2 4: s=9 }
```

Пример11. Составить программу для вычисления определенного интеграла $\int_a^b \frac{x}{(x^2+1)^2}\,dx$ **методом прямоугольников с точностью ε (изменяя число разбиений n).**

```
program integ;
var a,b,h,x,s:real;
        i,n:integer;
```

```
function f(x:real):real;
begin
  f:=x/sqr(sqr(x)+1)
end;
begin
  write('Введи a,в,n:');
  readln(a,b,n);h:=(b-a)/n;
  s:=0;x:=a-h/2;
  for i:=1 to n do
    begin
      x:=x+h;s:=s+f(x)
    end;
  s:=s*h; writeln('s=',s:7:4)
end.
```

Пример 12. Оформление программы модульной структуры.

Программа находит минимум и максимум в массиве, для чего используются функции — функция min(x,y), возвращающая минимум двух чисел, и функция max(x,y), возвращающая максимум двух чисел, описанные в модуле study.

```
program modmm;
uses crt,study; const n=7;
type mm=array[1..n] of integer;
const  m:mm=(6,8,2,0,5,6,9);
var i,xmin,xmax:integer;
begin
  xmin:=m[1]; xmax:=m[1];
  for i:=2 to n do
    begin
      xmin:=min(xmin,m[i]);
      xmax:=max(xmax,m[i])
    end;
  sound(300);delay(100);nosound;
  writeln('min=',xmin,' max=',xmax)
end.

unit study;
```

```
interface
  function min(x,y:integer):integer;
  function max(x,y:integer):integer;
implementation
  function min(x,y:integer):integer;
  begin
    if x<y then min:=x else min:=y
  end;
  function max(x,y:integer):integer;
  begin
    if x>y then max:=x else max:=y
  end;
end.
```

ЛИТЕРАТУРА

1. **Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию.—М.: Наука, 1988.**
2. **Бабушкина И.А., Бушмелева Н.А., Окулов С.М., Черных С.Ю. Практикум по Турбо Паскаля. — М.: АБФ, 1998.**
3. **Васюкова Н.Д., Тюляева В.В. Практикум по основам программирования. Язык ПАСКАЛЬ. — М.: Высш. шк., 1991.**
4. **Дайитбегов Д. М., Черноусов Е.А. Основы алгоритмизации и алгоритмические языки. — М.: Финансы и статистика, 1992.**
5. **Емелина Е.И. Основы программирования на языке Паскаль.- М.: Финансы и статистика, 1997.**
6. **Епанешников А.М., Епанешников В.А. Программирование в среде Turbo Pascal 7.0 . — М.: «ДИАЛОГ-МИФИ», 1996.**
7. **Зубов В.С. Программирование на языке TURBO PASCAL .- М.:ИИД ‘Филинъ’, 1997.**
8. **Зуев Е.А. Программирование на языке TURBO PASCAL 6.0, 7.0.- М.: Веста, Радио и связь, 1993.**
9. **Инструментальные средства персональных ЭВМ. В 10 кн. Кн. 4. Программирование в среде Турбо ПАСКАЛЬ: Практ. пособие /Агабеков Л.Е., Борисов С.В., Ваулин А.С. и др. — М.: Высш. шк.,1993.**
10. **Йенсен К., Вирт Н. Паскаль: руководство для пользователя. — М.: Финансы и статистика, 1989.**
11. **Марченко А.И., Марченко Л.А. Программирование в среде TURBO PASCAL 7.0. — М.: Бином Универсал, К.: ЮНИОР, 1997.**

12. **Офицеров Д.В., Долгий А.Б., Старых В.А. Программирование на персональных ЭВМ: Практикум. — Мн.: Высш. шк.,1993.**
13. **Офицеров Д.В., Старых В.А. Программирование в интегрированной среде Турбо - ПАСКАЛЬ. — Мн.: Высш. шк.,1992.**
14. **Першиков В.И., Савинков В.М. Толковый словарь по информатике. — М.: Финансы и статистика, 1995.**
15. **Попов В.Б. Turbo Pascal для школьников. Версия 7.0. — М.: Финансы и статистика, 1996.**
16. **Семашко Г.Л., Салтыков А.И. Программирование на языке паскаль. — М.: Наука, 1988.**
17. **Турбо Паскаль 7.0. — К.: Торгово - издательское бюро BHV, 1996.**
18. **Пильщиков В.Н. Сборник упражнений по языку Паскаль. — М.: Наука, 1989.**
19. **Файсман А.В. Профессиональное программирование на Турбо Паскале. - Info&F — Infomex — Koinko, 1992.**
20. **Фаронов В.В. Основы Турбо Паскаля.- М.: МВТУ - ФЕСТО ДИДАТИК,1992.**
21. **Хершель Р. Турбо Паскаль. — Вологда: МП «МИК», 1991.**
22. **Электронные вычислительные машины: В 8-ми кн. Под редакцией Савельева А.Я. Кн. 5. Языки программирования (ПАСКАЛЬ, ПЛ/ М) /Алексеев В.Е., Ваулин А.С.— М.: Высш. шк.,1987.**

ПОДПРОГРАММЫ В ПАСКАЛЕ.

Методические указания

Томский государственный университет - Томск, 1999. - 27с.

Подписано в печать Тираж экз. Бесплатно

Заказ № УОП ТГУ, Томск, ул. Никитина, 4