ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Язык программирования Pascal
Процедуры и функции. Рекурсия

Практикум

Специальность 010101 (010100) Математика

ВОРОНЕЖ
2005

Утверждено научно-методическим советом Математического факультета –
(28 февраля 2005 года, протокол №6)

Составители: Васильев В.В., Хливненко Л.В.

Практикум подготовлен на кафедре математического моделирования математического факультета Воронежского государственного университета.

Рекомендуется для студентов вечернего отделения математического факультета Воронежского государственного университета.

1. Метод последовательной детализации. Функции

При решении объемных задач, приводящих к большим программам, приходится структурировать программы, то есть разбивать на части - блоки.

Под **алгоритмическим блоком** обычно понимают часть алгоритма, имеющую определенное назначение с одним входом и одним выходом (*конструкция, напоминающая "черный" ящик*). Для алгоритмического блока четко определяются исходные (*входные*) данные и результаты (*выходные данные*), реакция на неправильные данные (*аномалии данных*) и работа в особых случаях.

Алгоритмический блок, вызываемый из другого блока, называется подблоком или **подпрограммой**.

Структурирование **методом последовательной детализации** заключается в пошаговой разработке алгоритма. Вначале пишется общая крупноблочная схема решения задачи. Потом каждый крупный блок разбивается на более мелкие и т.д. В итоге мы получаем иерархически упорядоченный набор элементарных блоков, представимых через имеющиеся процедуры и функции.

При **нисходящем способе** написания и отладки программы (*методе программирования сверху вниз*) пишется крупноблочная схема программы. На начальном этапе каждый блок заменяется **заглушкой** (*одноименный блок, имитирующий правильные результаты при конкретных исходных данных, либо пустой блок*). В процессе детализации программы заглушки заменяются работающими блоками.

При **восходящем способе** написания и отладки программы (*методе программирования снизу вверх*) детально прорабатываются элементарные блоки, которые затем состыкуются в более сложные блоки. Восходящая разработка программ используется при создании однотипных программ. Если Вы располагаете коллекцией отлаженных работающих блоков, то часть из них может подойти для новых программ.

Новую задачу лучше вначале разбить на более простые задачи (*провести декомпозицию задачи*), а затем при желании посмотреть свою коллекцию алгоритмических блоков решения простых задач. В этом случае вероятность упустить из виду важные детали несколько ниже, чем если пытаться собрать решение новой задачи из того, что Вы делали раньше.

Языки программирования, располагающие средствами структурирования программ, называются **процедурно-ориентированными**.

Turbo Pascal является процедурно-ориентированным языком. Основной файл программы содержит главный блок - основную программу. К основной программе можно подключать блоки, расположенные в других файлах - **модули**. Из основной программы можно вызывать вложенные в нее и в подключенные модули блоки, реализованные в Паскале в виде **процедур** и **функций**.

Процедуры и функции являются подпрограммами.

Нам знакомы стандартные процедуры (*например,* read(), write()) и функции

(*например*, abs(), exp()). Научимся создавать собственные (*пользовательские*) процедуры и функции.

Функции могут вычисляться на основе нескольких входных данных, но всегда имеют единственный результат, определенного в заголовке функции типа. Тип значения, возвращаемого функцией, может быть любым порядковым, вещественным, строковым типом String (*без указания длины строки!)* и типом Point (*ссылочным типом*).

Результат возвращается через имя функции. Поэтому допустимо включать функции в выражения, условия и процедуры вывода. Для возвращения значения из функции в вызывающую программу часто необходимо наличие оператора присваивания.

В общем виде описание функции выглядит так:

```
Function Имя (Список формальных параметров): Тип результата;
Label    Описание локальных меток;
Const    Описание локальных констант;
Type     Описание локальных типов;
Var      Описание локальных переменных;
Procedure  Описание внутренних процедур;
Function   Описание внутренних функций;
Begin Операторы, среди которых должен быть хотя бы один,
    который присваивает имени функции значение результата
End;
```

Параметром функции называется входное данное, передаваемое в подпрограмму из основной программы (*или из подпрограммы, обращающейся к данной функции*).

Формальный параметр функции - переменная, выполняющая роль аргумента функции.

Фактический параметр функции - выражение, переменная или константа типа формального параметра функции, значение фактического параметра передается в подпрограмму в качестве аргумента функции.

Областью видимости (*действия*) данных называют ту часть программы, где они могут быть использованы. Метки, константы, типы, переменные называются **локальными**, если они используются только в рамках одной функции или процедуры. Если метки, константы, типы, переменные действуют в рамках нескольких процедур или функций, то они называются **глобальными**.

Понятия “локальные” и “глобальные” весьма условны, и их нужно трактовать по отношению к конкретной процедуре или функции. Например, для некоторой функции описанные в ней переменные будут локальными, а для ее внутренней функции те же переменные будут глобальными.

При определении области видимости переменных следует помнить, что:

- внутри процедуры и функции действуют все идентификаторы, которые определены в этой процедуре или функции;

- внутри процедуры и функции действуют все идентификаторы внешних

блоков, в которые вложена данная процедура или функция (*если имена идентификаторов из внешнего окружения отличны от имен, объявленных внутри процедуры или функции*);

- локальные идентификаторы во внешнем окружении не действуют;
- в случае совпадения имен локальных и глобальных идентификаторов внутри функции или процедуры действие глобальных идентификаторов отменяется и используются одноименные локальные идентификаторы.

Напишем программу, в которой создается и используется новая функция.

Задача 1. Найти все числа Мерсенна типа integer.

♣ Французский математик **М.Мерсенн** (**1588-1648**) заметил, что многие простые числа имеют вид 2^p-1, где p является тоже простым числом. Все числа указанного вида называются числами Мерсенна. Не все числа Мерсенна являются простыми. Например, до **1903** г. считалось, что число $2^{67}-1$- простое. На самом деле, это число представляет собой произведение двух простых чисел **193 707 721** и **761 838 257 287**. Неясно, бесконечно ли множество чисел Мерсенна.

p+1-е число указанного вида можно выразить через p-е число следующим образом $2^{P+1}-1=2(2^P-1)+1$. Переменная p принимает значения идущих подряд натуральных чисел (2,3,4,...). Переход к вычислению очередного числа будем делать после проверки на неперепполнение типа integer. На экран будем выводить только те числа вида 2^P-1, для которых p - простое число.

Чтобы найти числа Мерсенна, нам потребуется функция, проверяющая, является ли число p ($p \geq 2$) делители в интервале $[2, p \ \ div \ \ 2]$. Делители будем искать только для нечетных чисел. Напомним, что функция odd() возвращает значение **истина**, если аргумент является нечетным числом.

В основной программе переменная n используется для перебора показателей степеней двойки. Mers хранит очередное число вида 2^n-1. Переменная логического типа p принимает значение true при выходе очередного числа вида 2^n-1 из диапазона integer.

В подпрограмме аргументом служит переменная p целого типа. Переменная n предназначена для перебора делителей числа p. ♠

```
Program Mersenn;
Uses crt;
Var n,mers:integer; p:boolean;
Function prost(p:integer):boolean;
Var n:integer;
Begin
 if p=2
   then prost:=true
   else if not odd(p)
           then prost:=false
           else
```

```
            begin
             n:=p div 2;  if not odd(n) then n:=n+1;
             while p mod n <> 0 do n:=n-2;
             prost:=n=1
            end
End; {prost}
Begin
    Textbackground(7); Textcolor(blue); Clrscr;
    writeln('Числа Мерсенна:');
    n:=2; mers:=3;  p:=false;
    repeat
     if prost(n) then writeln(mers);
     n:=n+1;+
     if mers<=maxint div 2
        then mers:=mers*2
        else p:=true;
     if mers<maxint
        then mers:=mers+1 else p:=true
    until p;
    readkey
End.{Mersenn}
```

В результате работы программы должно быть напечатано шесть чисел Мерсенна: 3, 7, 31, 127, 2047, 8191. Следующее число 131071 выходит за пределы типа integer.

Локальные данные создаются при вызове процедуры или функции и существуют только в период ее исполнения. В начале выполнения процедуры или функции выделяется память под локальные данные. Когда выполнение процедуры или функции заканчивается, то память, отведенная под хранение локальных данных, освобождается. Таким образом, **время жизни локальных данных** совпадает со временем работы процедуры или функции, в которой они определены. Поэтому значения всех локальных данных теряются вместе с освобождением памяти по завершении работы процедуры или функции.

Задача 2. В книге n страниц (n - *входное данное*). Составьте программу, которая будет находить, сколько цифр понадобится для того, чтобы занумеровать все страницы данной книги.

♣ Чтобы занумеровать все однозначные натуральные числа, нужно 9 цифр. Чтобы занумеровать все двузначные числа, нужно 90*2 цифр. Для трехзначных чисел понадобится 900*3 цифр и т.д. Чтобы занумеровать все k-значные числа, нужно $1 \cdot 2 \cdot 3 \ldots 900 \ldots 00 * k$ цифр.
$k-1$

Количество двузначных чисел находится как разность наибольшего и наименьшего двузначного числа плюс один: 90 = 99 - 10 + 1. Количество трехзначных чисел находится как разность наибольшего и наименьшего трехзначного числа плюс один: 900 = 999 - 100 + 1. Количество k-значных чисел находит-

ся как разность наибольшего и наименьшего k-значного числа плюс один: s = max - min + 1.

Напишем функцию str(n), которая будет вычислять количество цифр в книге из n страниц. Для этого определим количество цифр k в числе n, написав функцию kol_cifr(n) . Искомое значение str(n) найдем как сумму цифр, необходимых для нумерации всех i-значных чисел (i=1,2,...,k-1), и цифр, необходимых для нумерации оставшихся k-значных чисел. ♠

```
Program Nomer;
Uses crt;
Var k:integer;
function kol_cifr(n:integer):integer;
                {количество цифр в числе  n}
 var s:integer;
 begin
   s:=0;
   repeat
     s:=s+1; n:=n div 10  until n=0;
   kol_cifr:=s;
 end; {kol_cifr}
function str(n:integer):integer;
         {количество цифр в номерах книги из  n  страниц}
  var i, min, max, s:integer;
 begin
   s:=0; min:=1; max:=9;
   for i:=1 to kol_cifr(n)-1 do
     begin
       s:=s+(max-min+1)*i; min:=min*10; max:=min*10-1
     end;
    str:=s+(n-min+1)*kol_cifr(n)
  end; {str}
Begin
  Textbackground(7); Textcolor(blue); Clrscr;
  repeat
   write('Введите количество страниц в книге:'); readln(k)
  until k>0;
  writeln('Для нумерации всех страниц книги нужно ',str(k),' цифр');
  readkey
End.{Nomer}
```

В программе к задаче 2 при вычислении функции str() идет обращение к функции kol_cifr(). Функция kol_cifr() является подчиненной функции str(), хотя функция kol_cifr() не является внутренней функцией для str().

⓫ *Блоки похожи на "черные ящики" с односторонне зеркальной крышкой: из них видны (доступны) блоки и данные, описанные выше!*

2. Процедуры. Способы передачи параметров

Если подпрограмма должна возвращать в основную программу несколько значений, либо возвращать результаты структурированного типа, либо вообще не иметь параметров, то такую подпрограмму оформляют в виде **процедуры**.

В общем виде описание процедуры имеет следующий вид:

```
Procedure Имя (Список формальных параметров);
Label     Описание локальных меток;
Const     Описание локальных констант;
Type      Описание локальных типов;
Var       Описание локальных переменных;
Procedure   Описание внутренних процедур;
Function    Описание внутренних функций;
Begin       Операторы
End;
```

Параметром процедуры называется переменная (*иногда константа или выражение*), которая содержит входные и/или выходные данные процедуры.

Параметры, указываемые в заголовке процедуры при ее описании, называются **формальными параметрами**. Параметры, указываемые при вызове процедуры, называются **фактическими параметрами**.

Список формальных параметров может включать и результаты, передаваемые из подпрограммы в основную программу. Перед параметрами, возвращаемыми в основную программу, пишется служебное слово var.

При обращении к процедуре значения фактических параметров присваиваются соответствующим формальным параметрам. По завершению работы процедуры значения формальных параметров, объявленных результатами, доступны в вызывающем блоке. Соответствие определяется порядком следования формальных и фактических параметров.

Фактические параметры необходимо располагать в том же порядке следования, в котором расположены соответствующие им формальные параметры. За корректность передачи данных в подпрограмму и обратно отвечает программист. Компилятор отслеживает лишь очевидные случаи несоответствия формальных и фактических параметров - разное количество параметров и несоответствие типов.

Формальные параметры могут быть только переменными. Фактические параметры, значения которых передаются в подпрограмму, могут быть переменными, константами и выражениями (*типы формальных и фактических параметров должны совпадать*!).

Фактические параметры могут быть именами других функций!

Фактические параметры, служащие буфером для выходных данных подпрограммы, должны быть переменными.

Имена фактических и формальных параметров могут не совпадать!

Рассмотрим задачу, приводящую к программе с процедурой.

Задача 1. Аня нарвала яблок и поровну раздала своим сестрам Оле, Маше и Свете, а что осталось, съела. Оля свои яблоки поделила между тремя сестрами, а что осталось, съела. То же самое сделали Маша и Света. Сколько яб-

лок съела каждая сестра?

♣ Количество яблок, собранных Аней, обозначим через n. Будем считать, что девочки при дележе не разрезали яблоки на части. Каждую из четырех раздач можно выразить процедурой, имеющей четыре параметра - количество яблок у Ани, Оли, Маши и Светы. Все четыре параметра служат одновременно и входными и выходными данными для процедуры. При обращении к процедуре параметры хранят количества яблок до очередной раздачи. При выходе из процедуры фактическим параметрам присваиваются количества яблок после очередной раздачи. Первым в списке формальных параметров стоит количество яблок у той сестры, которая ими делится. ♠

```
Program Sestri;
Uses crt;
Var a,o,m,s:integer;
 procedure delezh(var w,x,y,z:integer);
 begin
   x:=x+w div 3;y:=y+w div 3; z:=z+w div 3;
   w:=w mod 3
 end;{delezh}
Begin
    Textbackground(7); Textcolor(blue); Clrscr;
    write('Сколько яблок нарвала Аня?');readln(a);
    o:=0; m:=0; s:=0;
    delezh(a,o,m,s);
    delezh(o,a,m,s);
    delezh(m,a,o,s);
    delezh(s,a,o,m);
    writeln('После дележа у Ани - ',a,', у Оли - ',o,', у Маши - ',m,',
у Светы -',s,' яблока');
    readkey
End.{Sestri}
```

⓵ *Область видимости формальных параметров такая же, как у локальных данных!*

Рассмотрим **способы передачи параметров**, принятые в Паскале. **По механизму передачи параметров** процедуры/функции различают передачу по значению и передачу по ссылке (*по адресу*).

При **передаче параметров по значению** при вызове процедуры/ функции выполняется копирование значений фактических параметров в память, выделяемую для формальных параметров.

При **передаче параметров по ссылке** при вызове процедуры/ функции выполняется копирование адресов (*но не значений*!) фактических параметров в выделяемую для них память.

По взаимодействию вызывающего и вызываемого блока параметры могут передаваться только как входные, только как выходные, одновременно как входные и выходные.

Теоретически возможны шесть способов передачи параметров.

1. Передача входного параметра по значению.
2. Передача входного параметра по ссылке.
3. Передача выходного параметра по значению.
4. Передача выходного параметра по ссылке.
5. Передача входного и выходного параметра по значению.
6. Передача входного и выходного параметра по ссылке.

В Паскале реализованы три из шести возможных способов передачи параметров процедуры/функции.

1. **Передача входного параметра по значению.** При вызове процедуры/функции выполняется копирование значений фактических параметров в память, выделяемую для формальных параметров. Во время работы процедуры/функции изменение значений формальных параметров не влияет на содержимое ячеек фактических параметров. При завершении работы процедуры/функции память, отведенная для значений формальных параметров, очищается, и значения формальных параметров теряются.
   Входные параметры, передаваемые по значению, называются **параметрами-значениями**. В заголовке процедуры/функции при описании таких параметров перед их идентификаторами дополнительные служебные слова не ставятся.
   Фактические параметры-значения могут быть переменными, константами, выражениями. Не допустимыми являются файловые типы и их производные.
2. **Передача входного параметра по ссылке.** При вызове процедуры/функции выполняется копирование адресов фактических параметров в выделяемую для них память, а также выделяется память для локальных данных. Во время работы процедуры/функции запрещено изменять значения формальных параметров. Запрещено также передавать параметры-константы в качестве фактических параметров другим процедурам/функциям. При завершении работы процедуры/функции память, отведенная для работы процедуры, очищается.
   Входные параметры, передаваемые по ссылке, называются **параметрами-константами**. В заголовке процедуры/функции при описании параметров-констант перед их идентификаторами ставится служебное слово const.
   Фактические параметры-константы могут быть переменными, константами, выражениями. Не допустимыми являются файловые типы и их производные.
   Параметры-константы обычно используют при передачи больших структур данных. При этом экономится оперативная память и обеспечивается целостность данных.
3. **Передача входного и выходного параметра по ссылке.** При вызове процедуры/функции выполняется копирование адресов фактических параметров в выделяемую для них память, а также выделяется память для локальных данных. Запрещается использовать в качестве входных параметров константы и выражения. Во время работы процедуры/функции разрешено изме-

нять значения формальных параметров. При этом изменение выполняется непосредственно в ячейках памяти фактических параметров. При завершении работы процедуры/ функции память, отведенная для работы процедуры, очищается. При этом новые значения фактических параметров не теряются. Входные/выходные параметры, передаваемые по ссылке, называются **параметрами-переменными**. В заголовке процедуры/функции при описании параметров-переменных перед их идентификаторами ставится служебное слово var.

Фактические параметры-переменные могут быть переменными любого типа, в том числе и файлового.

Одной из распространенных задач, приводящих к процедурам и функциям, является задача, связанная с операциями над большими числами.

Есть люди, которым нравится получать точные значения математических величин. Например, в XVI в. **Рудольф Ван Койне** (*Голландия*) вычислил значение p с точностью до 35-го знака после запятой и завещал высечь это число на своем надгробии.

Задача 2. Вычислить число e с точностью до 50-го знака после запятой.

♣ Многие функции вычисляются в ПК по формулам Тейлора. Математические константы могут быть найдены как суммы бесконечных последовательностей, выражающих данные константы. Например, $\frac{p}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+...+\frac{(-1)^{n-1}}{2n-1}+...$

Разложим функцию e^x в ряд Маклорена при

$x=1: e=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+...+\frac{1}{n!}+...$

Если переменная, хранящая очередной член ряда, будет иметь тип real, то мы не сможем достигнуть желаемой точности, поскольку очередной член ряда будет слишком мал (*фактически, в памяти будет храниться ноль*). Действительные числа в памяти ПК представлены приближенно. Количество точных цифр числа ограничивается количеством знаков, выделенных для представления числа.

Ниже приведен текст программы, которая печатает число десятичных знаков, выделяемых для представления действительного числа.

```
Program Tochnost;
Uses crt;
Const a=1.0;
Var n:integer; r:real;
Begin
    Textbackground(7); Textcolor(blue); Clrscr;
    r:=1.0; n:=0;
    repeat
      n:=n+1; r:=r/10
    until a=a+r;
    write('Точность типа real равна 1E-',n);
```

```
    readkey
End.{Tochnost}
```

Полученный ответ зависит от типа ПК и приблизительно равен 10^{-13}. Если переменная, хранящая очередной член ряда, будет иметь тип real, то мы можем получить число e с точностью до 13-го знака после запятой.

Для того чтобы найти число e с точностью до 50-го знака после запятой, цифры числа e сохраним в массиве следующего типа:

```
const t=50;
type vector = array [0..t] of [0..9];
```

Заметим, что для получения e с большей точностью достаточно будет в программе изменить значение константы t (*точность*).

Первый элемент массива цифр числа e, имеющий индекс 0, содержит целую часть числа e. Дробная часть начинается с элемента, индекс которого равен 1.

Проведем декомпозицию поставленной задачи.

1. Начальное заполнение массива e (*с цифрами числа* e) и массива i (*с цифрами n-го члена ряда*).
2. Сложение чисел, цифры которых хранятся в массивах e и i, и запись результата в массив e.
3. Получение следующего члена ряда в массиве i.
4. Повторение пунктов 2 , 3 до достижения желаемой точности.
5. Печать числа e (*цифр, хранящихся в массиве* e).

Разберем более детально каждую из перечисленных выше простых задач.

1. Первые два слагаемых в последовательности, выражающей число e, равны единице. Первую единицу будем считать начальным значением, записываемым в массив e. Вторую единицу запишем как начальное значение массива i (n=1). Для начального заполнения массивов составим процедуру `one(var a:vector)`, которая элементу массива a (*формальный параметр*) с индексом 0 (*целая часть числа*) присваивает 1, а все остальные элементы (*дробная часть*) заполняет нулями.
2. Складывать числа, цифры которых хранятся в массивах e и i, будем, используя способ сложения “в столбик”. Напишем процедуру `add(var e,i: vector)`. Введем локальную переменную v_ume (*в уме*). Найдем сумму s элементов e[k] и i[k], k = t, t-1,..., 0. Переменной v_ume присвоим частное от целочисленного деления s на 10, а e[k] присвоим остаток от такого деления. При k=0 (*целая часть числа*) мы не перейдем границу 10, потому что складываются члены последовательности, выражающей число e.
3. Для получения n-го члена ряда достаточно n-1-й член (*цифры которого хранятся в массиве* i) разделить на n. Составим процедуру `division(var i:vector; n:integer)`, в которой число из массива i делится столбиком на число n. Введем локальные переменные r для остатка и a для делимого. Первоначальное значение r равно нулю. a получается как остаток, увеличенный в 10 раз и сложенный с очередной цифрой делимого i[k] (k = 0, 1, ..., t).

Цифры искомого частного (*новое значение* i[k]) получаются как частные от целочисленного деления a на n, новые значения r получаются как остатки от такого деления.

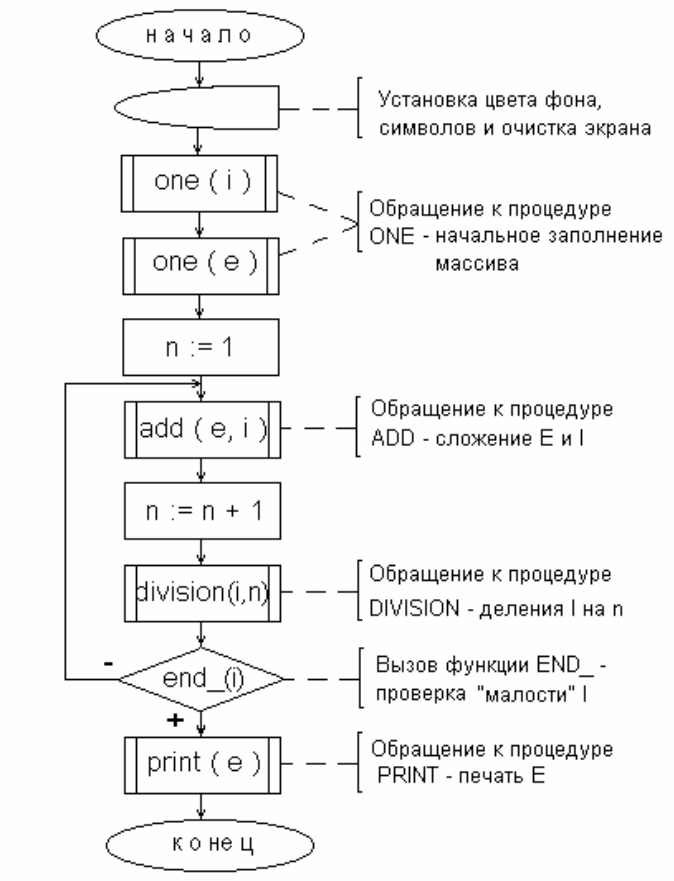


Схема алгоритма процедуры
ONE (a: vector)

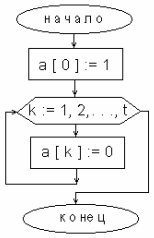


Схема алгоритма процедуры
ADD (e, i: vector)

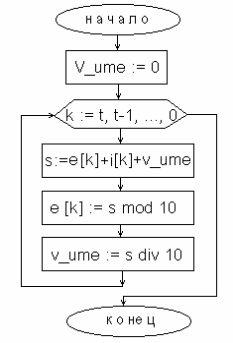


Схема алгоритма функции
END_ (i: vector)

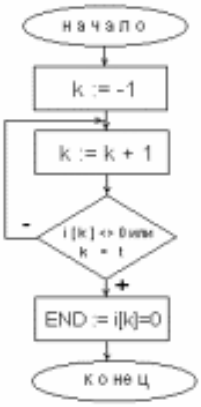


Схема алгоритма процедуры
DIVISION (a:vector; n:integer)

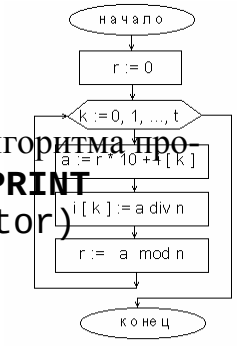
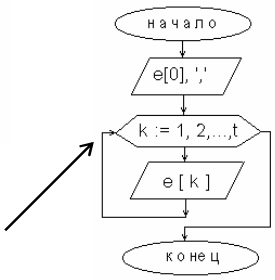


Схема алгоритма процедуры **PRINT** (e:vector)



4. Будем считать, что желаемая точность достигается, когда очередной член ряда оказывается настолько маленьким, что его цифры, отличные от нуля, начинаются с позиции, имеющей номер превосходящей t+1. То есть для нового члена ряда все i[k] (k = 0, 1, ..., t) равны нулю. Напишем функцию **end_(**const i:**vector**):**boolean**, которая будет проверять "малость" очередного члена ряда. Функция возвращает значения "истина" только когда все i[k] (k = 0, 1, ..., t) равны нулю. В этом случае цикл, в ходе которого накапливается сумма членов ряда и находится новый член ряда, прекращается.
5. Процедура **print(const** e:**vector)** выводит на экран элемент e[0] (*целая часть числа* e). Затем выводится запятая и в цикле печатаются остальные элементы массива e.

На стр. 13 представлена блок-схема основной программы и блок-схемы, вызываемых в основной программе процедур и функции. *Обратите внимание на новые элементы в оформлении блок-схем!* ♠

```
Program Epsilon;
 Uses crt;
 Const t=50;{количество знаков после запятой в числе e}
 Type vector=array [0..t] of 0..9;
      {тип массива для хранения t+1 цифр }
 Var n:integer; {номер члена ряда}
     e,i:vector;
   {массивы для хранения цифр числа e и очередного члена ряда}
Procedure one (var a:vector);
          {первоначальное заполнение массива}
 var k:1..t;{параметр цикла}
begin
   a[0]:=1;    for k:=1 to t do a[k]:=0
end; {one}
Procedure add(var e,i:vector);{сложение двух больших чисел}
 var v_ume:0..1{в_уме}; s:0..19{сумма двух цифр + в_уме};
     k:0..t;{параметр цикла}
begin
 v_ume:=0;
 for k:=t  downto 0 do
  begin
   s:=e[k]+i[k]+v_ume; e[k]:= s mod 10; v_ume:=s div 10
  end;
end;{add}
Procedure division(var i:vector; n:integer);
          {нахождение n-го члена ряда}
 var r{остаток от деления}, k{параметр цикла},
     a{делимое}:integer;
begin
 r:=0;
```

```
 for k:=0 to t do
  begin
   a:=r*10+i[k]; i[k]:=a div n; r:=a mod n
  end
end;{division}
function end_(const i:vector):boolean;
        {проверка достижения заданной точности}
 var k:integer;{параметр цикла}
begin
 k:=-1;
 repeat
  k:=k+1   until (i[k]<>0) or (k=t);
 end_:=i[k]=0
end; {end_}
procedure print(const e:vector);
         {печать массива с цифрами числа e}
 var k:1..t;{параметр цикла}
begin
 write('e=',e[0]:1,',');
 for k:=1 to t do write(e[k]:1);
end; {print}
Begin
    Textbackground(7); Textcolor(blue); Clrscr;
    one(i); one(e); n:=1;
    repeat
     add(e,i); n:=n+1; division(i,n)
    until end_(i);
    print(e);
    readkey
End.{Epsilon}
```

В результате работы программы на экране появится запись

e = 2,7182818284590452353602874713526624977572470936997 8

В Паскале допустимо использовать процедуры без параметров! В этом случае в заголовке процедуры после ее имени круглые скобки не ставятся.

Познакомимся с **открытыми параметрами-массивами**. В процедуру можно передавать через один параметр массивы различного размера. В этом случае в заголовке процедуры надо описать открытый параметр-массив:

```
procedure имя_процедуры ( имя_массива:
   array of тип_элементов_массива );
```

Например, в процедуру symbol нужно передавать массивы разного размера, состоящие из символов. Напишем заголовок процедуры:

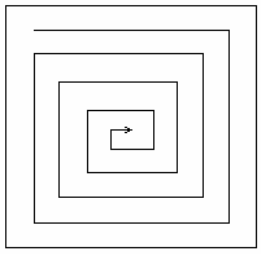
```
procedure symbol ( m: array of char );
```

Для определения характеристик переданного фактического параметра в процедуре могут быть использованы три стандартных функции: Low() - всегда возвращает 0 (*индекс первого элемента массива*), High() - возвращает индекс

последнего элемента массива, SizeOf() - возвращает размер фактического параметра - массива.

Фактические параметры-массивы должны состоять из элементов одного типа (совпадающего с типом элементов формального параметра-массива) и могут быть описаны как массивы разного размера.

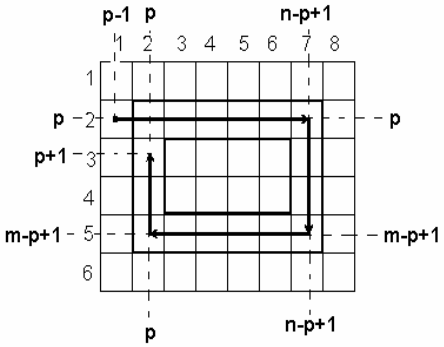
Разберем задачу, в которой в функцию требуется передана длины.



Задача 3. Для заданной вещественной матрицы определите, образуют ли ее элементы упорядоченную последовательность при их переборе по спирали, представленной справа. Для определения факта упорядоченности части строки (столбца) используйте функции.

♣ Будем считать, что задана вещественная матрица A размера mxn. Элементы матрицы, перебираемые по указанной схеме, могут быть упорядочены по неубыванию и невозрастанию. Создадим логические функции rost и spad, которые будут проверять, упорядочен ли передаваемый им в качестве аргумента массив по невозрастанию и по неубыванию соответственно.

Аргументом функций rost и spad будет вспомогательный массив v, в который помещается часть строки или столбца очередного витка спирали. В качестве аргумента функциям rost и spad передается и количество элементов в части строки или столбца.



На схеме справа показана закономерность изменения индексов при обходе матрицы по спирали. p - номер текущего витка спирали. Количество витков спирали p_max находится как результат целочисленного деления на два суммы единицы и меньшего из m и n.

Логические переменные f и g используются для хранения ответов на вопросы: "Упорядочены ли элементы матрицы, перебираемые по спирали, по неубыванию и по невозрастанию?" Первоначально переменным f и g присваивается значение true.

Организуем цикл по p от 1 до p_max.

Проверим, упорядочена ли **верхняя строка** p-го витка. Для этого организуем цикл по j от p-1 до n-p+1 (*см. схему выше*). В цикле перебираются столбцы. Запишем верхнюю строку p-го витка во вспомогательный массив v размера r, где r = n-2p+3. Во вспомогательный массив записываются элементы матрицы A [p, j]. Начинать цикл от p-1 нужно для того, чтобы обеспечить проверку упорядоченности "стыков" двух витков спирали. Отсюда следует **особый случай**: на первом витке спирали "стыков" нет, поэтому при p=1 первую итерацию цикла нужно пропустить без действий.

Переменная f принимает значение результата конъюнкции своего прежнего значения и значения функции rost(v,r). Переменная g принимает значение

результата конъюнкции своего прежнего значения и значения функции spad(v,r).

Если оказывается, что обе переменные f и g имеют значение false, то цикл по p прекращается и печатается неудовлетворительный результат.

Проверка упорядоченности **правого столбца** p-го витка проводится аналогично. Для формирования вспомогательного массива организуется цикл по j от p до m-p+1, в котором перебираются строки (*см. схему выше*). Во вспомогательный массив записываются элементы матрицы A [j,n-p+1]. Если на последнем витке спирали правого столбца нет (*например, у матриц размера 3x5*), то вспомогательный массив будет содержать один элемент a[p, n-p+1]. Функции rost и spad в этом случае возвратят значение true, и значение переменных f и g не будет искажено.

При проверке упорядоченности **нижней строки** p-го витка для формирования вспомогательного массива организуется цикл по j от n-p+1 до p, в котором перебираются столбцы (*см. схему выше*). Во вспомогательный массив записываются элементы матрицы A [m-p+1,j]. **Особый случай**: на последнем витке спирали нижней строки нет. В этом случае формировать вспомогательный массив и использовать функции rost и spad для получения новых значений f и g нельзя (*подумайте, почему?).* Особый случай имеет место на последнем витке спирали у матриц с нечетным количеством строк m.

При проверке упорядоченности **левого столбца** p-го витка для формирования вспомогательного массива организуется цикл по j от m-p+1 до p+1, в котором перебираются строки (*см. схему выше*). Во вспомогательный массив записываются элементы матрицы A [j,p]. **Особый случай**: на последнем витке спирали левого столбца нет. В этом случае формировать вспомогательный массив и использовать функции rost и spad для получения новых значений f и g нельзя (*подумайте, почему?).* Особый случай имеет место на последнем витке спирали у матриц с нечетным количеством столбцов n.

По завершению внешнего цикла печатается положительный результат в случае истинности дизъюнкции f и g.

В представленной ниже программе для наглядности ее работы предусмотрена возможность выводить на экран элементы вспомогательных массивов. Операторы, печатающие элементы вспомогательных массивов, оформлены как комментарии, содержащие слово “контроль”. ♠

```
Program Snake;
 Uses crt;
 Const m_max=10; n_max=10;
 Type matr=array[1..m_max,1..n_max] of real;
      vect=array[1..n_max] of real;
 Var a:matr;k,r,p,i,j,m,n,p_max:integer; f,g:boolean;
v:vect;
function rost(v:vect; n:integer):boolean;
 var s:boolean; i:integer;
```

```
begin
 s:=true;
 for i:=2 to n do s:=s and (v[i-1]<=v[i]);   rost:=s
end;{rost}
function spad (v:vect; n:integer):boolean;
 var s:boolean; i:integer;
begin
 s:=true;
 for i:=2 to n do s:=s and (v[i-1]>=v[i]);
 spad:=s
end;{spad}
Begin
 Textbackground(7); Textcolor(blue); Clrscr;
 writeln('Введите количество строк матрицы (не больше ',m_max,')');
readln(m);
 writeln('Введите количество столбцов матрицы (не больше ',n_max,')');
readln(n);
 writeln('Введите вещественную матрицу размером ',m,'x',n);
 for i:=1 to m do
  for j:=1 to n do read(a[i,j]);
 writeln('Введена вещественная матрица:');
 for i:=1 to m do
  begin for j:=1 to n do write (a[i,j]:7:1); writeln end;
 {-----------------------------------------------------}
 f:=true; g:=true;
 if m<=n then p_max:=(m+1) div 2 else p_max:=(n+1) div 2;
 for p:=1 to p_max do
  begin
{верхняя строка p-го витка}
   k:=1;
   for j:=p-1 to n-p+1 do
    begin
     if j=0 then continue;  v[k]:=a[p,j]; k:=k+1
    end; r:=k-1;
  { (* контроль *)  for j:=1 to r do write(v[j]:3:0,' '); writeln;}
   f:=f and rost(v,r); g:=g and spad(v,r);
  if not(f or g) then break;
{правый столбец p-го витка}
   k:=1;
   for j:=p to m-p+1 do
    begin v[k]:=a[j,n-p+1]; k:=k+1 end; r:=k-1;
   { (* контроль *)   for j:=1 to r do write(v[j]:3:0,' '); writeln;}
   f:=f and rost(v,r); g:=g and spad(v,r);
   if not(f or g) then break;
{нижняя строка p-го витка}
```

```
   k:=1;
   if not(odd(m)) or (p<>p_max)  then
    begin
     for j:=n-p+1 downto p do
      begin v[k]:=a[m-p+1,j]; k:=k+1 end; r:=k-1;
     { (* контроль *)    for j:=1 to r do write(v[j]:3:0,' '); writeln;  }
      f:=f and rost(v,r); g:=g and spad(v,r);
      if not(f or g) then break;
    end;
{левый столбец p-го витка}
   k:=1;
   if not(odd(n)) or (p<>p_max) then
    begin
      for j:=m-p+1 downto p+1 do
       begin v[k]:=a[j,p]; k:=k+1 end; r:=k-1;
      { (* контроль *)   for j:=1 to r do write(v[j]:3:0,' '); writeln;}
        f:=f and rost(v,r); g:=g and spad(v,r);
        if not(f or g) then break
    end;
  end;
  if f or g
   then writeln('Элементы образуют упорядоченную последовательность')
   else writeln('Элементы не образуют упорядоченную последовательность');
 readkey
End.{Snake}
```

Задача 3 решена без использования открытых параметров-массивов. В созданные нами процедуры rost и spad передавался массив одного размера, хотя фактически количество элементов в передаваемых массивах было различным. *Как можно исправить программу, включив в функции открытые параметры-массивы?*

Познакомимся с процедурами, вызывающими досрочное прекращение работы блоков. Нам знакома процедура Break, которая позволяет досрочно завершить цикл и перейти к оператору, следующему за циклом. Существует оператор Exit, который досрочно завершает работу блока с возвратом в вызывающий блок. Оператор Halt прекращает работу программы, даже если он находится в подблоке. В основной программе действие операторов Exit и Halt одинаково.

Вычислим неберущийся интеграл с заданной точностью.

Задача 4. Пусть даны вещественные числа a, b, e (a<b, e>0). С точностью e вычислите интеграл $\int_a^b f(x)dx$, $f(x)=\frac{\sin x}{x}$, используя формулу трапеций $\int_a^b f(x)dx=\frac{h}{2}\left(f(a_0)+2f(a_1)+2f(a_2)+\ldots+2f(a_{n-1})+f(a_n)\right)$. Для обеспечения нужной точности воспользуйтесь следующим правилом Рунге: если прибли-

женное значение интеграла I_n вычислять при $n = n_0, 2n_0, 4n_0$ и т.д. (*где* n_0 *– некоторое начальное число отрезков разбиения, например,* $n_0=10$), тогда при $\left|I_{2n} - I_n\right| / 3 < e$ за искомую величину можно взять I_{2n}.

♣ Вычисление подынтегральной функции опишем в подпрограмме-функции с именем f(x). **Особый случай**: при **x**=0 значением функции будем считать единицу (*первый замечательный предел*). В процедуре Integral вычислим I_n по формуле трапеций. Правило Рунге реализуем в основной программе в виде цикла, в теле которого вызывается процедура Integral. Цикл будет выполняться, пока $\left|I_{2n} - I_n\right| / 3 \geq e$. ♠

```
Program Runge;
Uses crt;
Const n0=10;
Var n:integer; a,b,e,Ip,I:real;
Function f (x:real):real;
 begin
    if x=0
     then f:=1
     else f:=sin(x)/x;
 end;{f}
Procedure Integral(n:integer; var S:real);
 var i:integer; x,h:real;
 begin
   x:=a; S:=f(a); h:=(b-a)/n;
   for i:=1 to n-1 do
    begin
     x:=x+h; S:=S+2*f(x);
    end;
   S:=S+f(b); S:=S*h/2
 end;{Integral}
Begin
    Textbackground(7); Textcolor(blue); Clrscr;
    write('Введите пределы интегрирования:');
    read(a);
    repeat
     readln(b)   until b>a;
    write('Введите точность вычислений:');
    repeat
     readln(e)   until e>0;
   {-----------------------------------------------}
    Integral(n0,Ip); n:=2*n0; Integral(n,I);
    while abs(I-Ip)/3>=e do
     begin
      Ip:=I; n:=2*n; Integral(n,I);
     end;
```

```
    writeln('Искомый интеграл равен ',I:8:4);
    readkey
End. {Runge}
```

Проверьте работу программы на ПК!

3. Рекурсивные подпрограммы

Процедуры и функции, которые вызывают сами себя, называются **рекурсивными**.

Рекурсивные объекты частично определяются через самих себя. То есть рекурсивные алгоритмы появляются, когда решение задачи для случая n выражается через решение задачи для случая n-1.

Например, факториал можно определить с помощью рекуррентной формулы:

0! = 1, n! = n * (n-1)!

Рекурсивная функция для вычисления n! может быть такой:

```
function fact (n:byte) :integer;
 begin
    if n=0
      then fact:=1
      else fact:=n*fact(n-1)
 end;
```

Решение задачи, реализуемое рекурсивным алгоритмом, можно выразить нерекурсивным алгоритмом. Например, n! можно вычислить так:

```
function fact_ (n:byte) :integer;
 var i, k:integer;
 begin
    k:=1;
    for i:=2 to n do k:=i*k;
    fact_:=k
 end;
```

Рекурсивные процедуры и функции просты, наглядны и компактны. Однако рекурсивные процедуры и функции требуют большего размера оперативной памяти во время выполнения, чем нерекурсивные.

Значения всех локальных переменных и параметров, передаваемых по значению, при очередном вызове рекурсивной процедуры или функции помещаются в стек.

Стеком называется структура, напоминающая трубку с запаянным концом. При заполнении стека действует принцип «первым пришел, последним ушел». Если стек заполняется 1-м, 2-м, 3-м элементами, то извлекаться эти элементы будут в обратном порядке – 3-й, 2-й, 1-й.

То есть одной локальной переменной на разных уровнях рекурсии будут соответствовать разные ячейки памяти.

Глубина рекурсии – максимальное число рекурсивных вызовов процедуры. **Текущий уровень рекурсии** – количество рекурсивных вызовов в текущий момент времени.

В рекурсивном алгоритме должно присутствовать условие выхода из рекурсии. В противном случае рекурсивный алгоритм может вызывать себя «бесконечное» число раз. Ждать результатов работы программы придется долго.

В действительности, бесконечного самовызова не получится, так как ресурсы оперативной памяти ограничены.

Вызов рекурсивной процедуры или функции должен выполняться по условию, которое на каком-то уровне рекурсии станет ложным!

Если условие истинно, то осуществляется **рекурсивный спуск**. Как только условие стало ложным, начинается **рекурсивный возврат.**

Рассмотрим три разных формы рекурсивной процедуры/ функции.

1. Процедура с выполнением действий на рекурсивном спуске:

```
Procedure Рекурсия;
begin
    Действия;
    if Условие then Рекурсия
end;
```

2. Процедура с выполнением действий на рекурсивном возврате:

```
Procedure Рекурсия;
begin
    if Условие then Рекурсия;
    Действия;
end;
```

3. Процедура с выполнением действий на рекурсивном спуске и возврате:

```
Procedure Рекурсия;
begin
    Действия;
    if Условие then Рекурсия;
    Действия;
end;
```

Рассмотрим задачу, в которой действия выполняются на рекурсивном спуске и возврате.

Задача 1. Пусть дана строка текста. Напечатайте ее в обратном порядке.

♣ Рекурсивная процедура Rec читает символы строки на рекурсивном спуске и печатает считанные символы на рекурсивном возврате. Локальная переменная c:char хранит считанный на очередном уровне рекурсии символ. Условием выхода из рекурсии служит значение функции eoln (*конец строки*). При нажатии клавиши Enter начинается рекурсивный возврат и считанные символы печатаются в обратном порядке. ♠

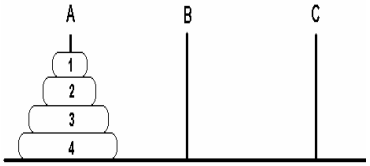
```
Program Text;
 Uses crt;
 Procedure Rec;
  var c:char;
  begin
    if not eoln
```

```
      then  begin read(c); Rec; write(c)  end;
  end; {Rec}
Begin
  Rec; readkey
End. {Text}
```

Познакомимся с красивой задачей, связанной с легендой, решение которой приводит к рекурсивному алгоритму. Говорят, что когда-то в Ханое стоял храм, а рядом с ним – три башни (*стержня*). На первую башню надеты 64 диска различного диаметра. Сложенные диски на стержне похожи на детскую игрушку – пирамиду. Монахи этого храма должны переместить все диски с первого стержня на третий, соблюдая следующие правила:

1. можно перемещать лишь по одному диску;
2. больший диск нельзя класть на меньший;
3. снятый диск нельзя отложить – его необходимо сразу же надеть на другой стержень.

Легенда утверждает, что когда монахи закончат свою работу – а они перемещают диск за одну секунду! – наступит конец света. Когда дисков много, решение этой задачи оказывается слишком долгим.



Задача 2. «Ханойские башни». Пусть имеются три столбика **A**, **B**, **C** и n дисков разного размера, перенумерованных от 1 до n в порядке возрастания их размеров. Сначала все диски надеты на столбик **A** в виде пирамиды. Требуется перенести все диски со столбика **A** на столбик **C**, соблюдая следующие правила: диски можно переносить только по одному, диск большего размера нельзя ставить на меньший. Диск B можно использовать в качестве промежуточного.

♣ Задачу можно решить, составив рекурсивный алгоритм. Решение задачи для n дисков можно сформулировать через решение задачи для n-1 дисков и т.д. Для случая n=1 решение сводится к перемещению единственного диска с исходного столбика на целевой столбик. Составим рекурсивный алгоритм:

```
Если n>0, то
переместить верхние n-1 диск со столбика A на столбик B,
    используя столбик C как вспомогательный,
  переместить оставшийся нижний диск со столбика A на
    столбик C,
    переместить n-1 диск со столбика B на столбик C,
    используя столбик A как вспомогательный.
```

Параметрами рекурсивной процедуры move являются переменные n (*число перемещаемых дисков на текущем уровне рекурсии*), Source (*буква исходного столбика*), Help (*буква вспомогательного столбика*), Res (*буква целевого столбика*). В глобальной переменной k накапливается число сделанных ходов. ♠

```
Program Hanoi;
Uses crt;
Var n,k:integer;
```

```
 Procedure move(n:byte; Source, Help, Res: Char);
  begin
   if n>0
    then
     begin
      move(n-1,Source,Res,Help); k:=k+1;
      writeln('Снять диск №',n,' со столбика ',Source,
              ' и надеть на столбик ',Res);
      move(n-1,Help,Source,Res)
     end    end; {move}
Begin
  Textbackground(7); Textcolor(blue); Clrscr;
  Write('Введите количество дисков:');Readln(n);
  Writeln('Последовательность действий:');
  k:=0; move(n,'A','B','C');
  writeln('Минимальное число ходов:',k); readkey
End. {Hanoi}
```

Ниже приведен результат работы программы Hanoi при n=3.

```
 Введите количество дисков:3
 Последовательность действий:
 Снять диск №1 со столбика A и надеть на
столбик C
 Снять диск №2 со столбика A и надеть на
столбик B
 Снять диск №1 со столбика C и надеть на
столбик B
 Снять диск №3 со столбика A и надеть на
столбик C
 Снять диск №1 со столбика B и надеть на
столбик A
 Снять диск №2 со столбика B и надеть на
столбик C
 Снять диск №1 со столбика A и надеть на
столбик C
 Минимальное число ходов:7
```

Можно провести аналогию между рекурсивным методом программирования и методом математической индукции. Вначале задача решается для простого случая, например, для n=1. Затем предполагается, что существует решение для случая n-1 и решение для случая n сводится к решению для случая n-1. При этом процедура вызывает сама себя до тех пор, пока не будет достигнута элементарная ситуация.

Познакомимся с рекурсивным алгоритмом быстрой сортировки массива. Метод быстрой сортировки был разработан в 1962 г. профессором Оксфордского университета **К. Хоаром.**

Задача 3. «Быстрая сортировка». Упорядочите элементы одномерного массива по неубыванию методом быстрой сортировки.

♣ **Метод быстрой сортировки** заключается в следующем:

I **этап**. Обозначим через a и b границы сортируемой части массива **X**(n). Вначале a=1, b=n. В массиве **X** ищется элемент S, стоящий около середины. Элементы массива просматриваются поочередно слева направо и справа налево. При переборе элементов слева направо определяется позиция i элемента **X**[i] => S. При переборе элементов справа налево определяется позиция j элемента **X**[j] <= S. Если элемент **X**[i] находится слева от S, а элемент **X**[j] – справа, то найденные элементы **X**[i] и **X**[j] меняются местами. Встречный поиск продолжается до тех пор, пока индексы i и j не пересекутся, т.е. i>j. Например, пусть массив X состоит из 5 элементов:

5 1 4 2 3.

S=4. При i=1, j=5 переставляются крайние элементы: **3** 1 4 2 **5.**

Встречный поиск будет продолжен при i=2, j=4.

При i=3, j=4 переставляются 3-й и 4-й элементы: 3 1 **2 4** 5.

Текущие значения i и j изменяются на единицу. i=4, j=3.

I этап закончен, так как i>j.

После завершения I этапа все элементы, меньшие или равные среднему, окажутся слева от границы пересечения индексов i и j, а все элементы, которые больше или равны среднему, окажутся справа. Однако левая и правая части массива еще могут быть неотсортированными.

II **этап**. На втором этапе повторяются действия первого этапа для левой и правой частей массива. Если j оказывается больше a, то действия первого этапа реализуются для элементов массива с номерами от a до j. Если i оказывается меньше b, то действия первого этапа реализуются для элементов массива с номерами от i до b.

В нашем примере после первого этапа i=4, j=3. Значит, на втором уровне рекурсии сортировке будут подвержены элементы левой части массива с номерами от 1 до 3: 3 1 2|4 5 и правой части с номерами от 4 до 5.

На **втором уровне рекурсии для левой части** массива: 3 1 2 4|5: S=1.

При i=1, j=2 переставляются 1-й и 2-й элементы: **1 3** 2 4|5.

Текущие значения i и j изменяются на единицу. i=2, j=1.

Второй уровень рекурсии закончен, так как i>j.

После второго уровня рекурсии для левой части массива i=2, j=1. Значит, на третьем уровне рекурсии сортировке будут подвержены элементы с номерами от 2 до 3: правой части рассматриваемого участка:1|3 2|4 5.

На **третьем уровне рекурсии для левой части** массива: 1|3 2|4 5:

S=3. При i=2, j=3 переставляются 2-й и 3-й элементы: 1|**2 3**|4 5.

Текущие значения i и j изменяются на единицу. i=3, j=2.

Третий уровень рекурсии закончен, так как i>j. Дальнейший рекурсивный вызов не выполняется, так как после данного шага i=3, j=2, т.е. j=a, а i=b.

На **втором уровне рекурсии для правой части** массива: 1 2 3|4 5:

S=4. При i=4, j=4 переставляются 4-й и 4-й элементы: 1 2 3|**4** 5.

Текущие значения i и j изменяются на единицу. i=5, j=3.

Второй уровень рекурсии закончен, так как i>j.

Третьего уровня рекурсии для правой части массива не будет, так как j<a, а i=b.

```
Program Q_Sort;
Uses crt;
Const n=10;
Var x:array[1..n] of integer; i:integer;
 Procedure Quick ( a,b: integer);
  var s,r,i,j:integer;
  begin
    s:=x[(a+b) div 2]; i:=a; j:=b;
    while i<=j do
     begin
      while x[i]<s do i:=i+1;
      while x[j]>s do j:=j-1;
      if i<=j   then
          begin
            r:=x[i]; x[i]:=x[j]; x[j]:=r; i:=i+1; j:=j-1
          end;
     end;
    if a<j then Quick(a,j);
    if i<b then Quick(i,b);
  end; {Quick}
Begin
  Textbackground(7); Textcolor(blue); Clrscr;
  for i:=1 to n do read(x[i]); readln; Quick(1,n);
  for i:=1 to n do write(x[i],' '); readkey
End. {Q_Sort}
```

Рассмотрим математическую задачу, решение которой приводит к рекурсивному алгоритму.

Задача 4. Вычислите определитель заданной матрицы, пользуясь формулой разложения по первой строке (*матрица* **B** *получается вычеркиванием из* **A** *первой строки и* k*-го столбца*): $\det A = \sum_{k=1}^{n} (-1)^{k+1} A_{1,k} \det B_k$.

♣ Составим рекурсивную функцию det, вычисляющую определитель det **A** порядка n. Если n>1, то определитель det **A** раскладывается по первой строке. Минор B_{1k} элемента a_{1k} получается копированием элементов матрицы **A** и смещением элементов на одну строку вверх и на один столбец влево. При этом элементы первой строки и k-го столбца теряются, а порядок определителя уменьшается на единицу.

Если n=1, то определитель будет равен элементу **a**[1,1].

Константа n_max содержит порядок определителя. Тип Mas описывает квадратную матрицу размера n_max x n_max. Переменные i и j используются как счетчики циклов.

В функции det в переменной s накапливается сумма произведений элементов первой строки на их алгебраические дополнения. В переменной sign хранится результат возведения (-1) в степень k+1. ♠

```
Program Det_A;
Uses crt;
Const n_max=3;
Type Mas=array[1..n_max,1..n_max] of integer;
Var a:Mas;i,j:integer;
Function det(a:Mas; n:integer):integer;
var i,j,s,k,sign:integer; b:Mas;
begin
 if n>1
  then
   begin
    s:=0;
    for k:=1 to n do
     begin
      if odd(k+1)
        then sign:=-1 else sign:=1;
      b:=a;
      {вычеркивание 1-й строки и k-го столбца}
      for i:=1 to n-1 do
        for j:=1 to n do b[i,j]:=b[i+1,j];
      for i:=1 to n-1 do
       for j:=k to n-1 do b[i,j]:=b[i,j+1];
      s:=s+sign*a[1,k]*det(b,n-1);
     end;
    det:=s
   end
  else   det:=a[1,1];
end;{det}
Begin
    Textbackground(7); Textcolor(blue); Clrscr;
    for i:=1 to n_max do
     for j:=1 to n_max do read(a[i,j]);
    writeln('det=',det(a,n_max)); readkey
End.{Det_A}
```

*Подготовьтесь к ответам на **все(!)** контрольные вопросы и выполните **все(!)** контрольные задания. Дорогу осилит идущий!*

Контрольные вопросы и задания

1. Метод последовательной детализации. Функции

1. Что такое алгоритмический блок?
2. Что такое блочное программирование? Зачем оно применяется?
3. Что называют подпрограммой?
4. В чем суть метода последовательной детализации?

5. В чем заключается метод программирования сверху вниз?
6. В чем заключается метод программирования снизу вверх?
7. Каким методом программирования целесообразнее решать новую задачу? Объясните, почему.
8. Какие языки программирования называются процедурно-ориентированными?
9. Является ли Turbo Pascal процедурно-ориентированным? Почему?
10. Какие средства структурирования программ имеются в Паскале?
11. Какие стандартные процедуры и функции Вам известны?
12. Что такое модуль? Приведите пример стандартного модуля и процедур из него.
13. В каком случае подпрограмма называется функцией?
14. Какое количество значений возвращает функция?
15. Как определить тип значения, возвращаемого функцией?
16. Существуют ли ограничения на тип возвращаемого функцией значения?
17. Допустимо ли следующее описание функции:
    function f (x:real) : string [5]?
18. Приведите несколько способов вызова функции из основной программы.
19. Как в общем виде описывается функция?
20. Что такое параметр функции? Что называют фактическим и формальным параметром функции?
21. Что называют областью видимости данных?
22. Какие данные называются локальными? Глобальными?
23. Почему понятия “локальные” и “глобальные” являются условными?
24. Назовите четыре правила определения зоны действия идентификаторов процедур и функций.
25. Может ли имя локальной переменной совпадать с именем глобальной? Какая из двух одноименных переменных будет действовать в процедуре? В основной программе?
26. Можно ли создавать пользовательские функции с именами стандартных функций (*например, можно ли создать свою функцию* sin())?
27. Напишите программу, печатающую все числа Мерсенна типа integer. Используйте в программе функцию, определяющую, является ли аргумент простым числом. Напишите алгоритм и блок-схему.
28. Что называют временем жизни локальных данных?
29. В книге n страниц (n - *входное данное*). Составьте алгоритм, блок-схему и программу, которая будет находить, сколько цифр понадобится для того, чтобы занумеровать все страницы данной книги. Используйте две пользовательские функции.
30. Почему блоки сравнивают с “черными ящиками” с односторонне зеркальной крышкой?
31. Напишите функцию, вычисляющую sh(x), и для заданного значения переменной x вычислите следующее выражение:
    $sh(x) + tg(x+1) - tg^2(2 + sh(x - 1))$.

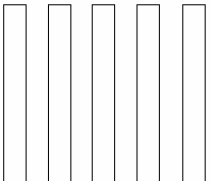
32. Напишите, используя данный фрагмент программы:
```
type страна = (Швейцария, Нидерланды, Бельгия, Ирландия, Норвегия);
   столица = (Берн, Амстердам, Брюссель, Дублин, Осло);
```
функцию, которая по заданной стране определяет ее столицу.
33. Для заданного натурального числа N определите первые N простых чисел.
34. Напишите функцию, проверяющую, является ли заданная литера гласной русской буквой.
35. Заданный массив целых чисел делится на три части двумя элементами: максимальными минимальным. Определите сумму элементов в каждой части массива. Используйте функции для нахождения индексов минимального и максимального элементов и подсчета суммы элементов в указанной части массива.
36. Пусть дана прямоугольная матрица A(nxm), элементами которой являются целые числа. Определите, в какой строке матрицы находится наибольшее количество симметричных чисел. Составьте функцию, проверяющую симметричность числа.
37. Для заданной строки текста определите слова, которые содержат символы, отличные от букв. Напишите функцию, определяющую тип символа строки.

2. Процедуры. Способы передачи параметров

38. Дайте определение процедуры.
39. Как в общем виде описывается процедура?
40. Что называют параметром процедуры?
41. Какие параметры называются формальными? Фактическими?
42. Какое количество значений возвращает процедура?
43. Каким образом осуществляется обмен данными между вызывающим и вызываемым блоками?
44. Могут ли формальные параметры быть константами? Выражениями? Именами других функций или процедур? Объясните ответ.
45. Могут ли фактические параметры быть переменными? Константами? Выражениями? Именами других функций? Именами других процедур? Почему?
46. Могут ли имена формальных параметров совпадать/ не совпадать с именами фактических параметров?
47. Как определить область видимости формального параметра?
48. Назовите два основания классификации способов передачи параметров.
49. Что происходит при передаче параметров по значению?
50. Что происходит при передаче параметров по ссылке?
51. Назовите шесть теоретически возможных способов передачи параметров. Какие из них реализованы в Паскале?
52. Как происходит передача входного параметра по значению? Можно ли изменять значение такого формального параметра? Изменится ли при этом фактический параметр? Изменится ли одноименный фактический параметр?
53. Что такое параметры-значения? Какого вида и типа могут быть фактические

параметры-значения?

54. Как происходит передача входного параметра по ссылке? Можно ли изменять значение такого формального параметра? Можно ли передавать параметр другим процедурам и функциям?
55. Что такое параметры-константы? Как описать формальный параметр-константу? Какого вида и типа могут быть фактические параметры-константы? Когда используют параметры-константы?
56. Как происходит передача входного и выходного параметра по ссылке? Можно ли изменять значение такого формального параметра? Изменится ли при этом соответствующий фактический параметр? Почему? Будет ли доступен формальный параметр в основной программе?
57. Что такое параметры-переменные? Как описать формальный параметр-переменную? Могут ли фактические параметры-переменные быть константами, выражениями? Какого типа могут быть фактические параметры-переменные?
58. Существуют ли процедуры без параметров? Как осуществляется обмен данными между основной программой и подпрограммой без параметров?
59. Что такое открытые параметры-массивы? Зачем они нужны?
60. Как описать открытый параметр-массив? Приведите пример.
61. Как работают функции Low(),High(),SizeOf()?
62. Какие ограничения накладываются на фактические параметры-массивы?
63. Как работают процедуры Exit и Halt?
64. Напишите программу с процедурой для решения следующей задачи: “Аня нарвала яблок и поровну раздала своим сестрам Оле, Маше и Свете, а что осталось, съела. Оля свои яблоки поделила между тремя сестрами, а что осталось, съела. То же самое сделали Маша и Света. Сколько яблок оказалось у каждой сестры?”
65. Составьте программу для решения задачи о дележе в общем случае, то есть для дележа яблок между n (входное данное!) лицами. Дележ осуществляется по тем же правилам, что и в предыдущей задаче.
66. Вычислите число e с точностью до 50-го знака после запятой. Проведите декомпозицию задачи, напишите алгоритм, блок-схему и программу.
67. Напишите программу, определяющую количество десятичных знаков, выделяемых для представления действительного числа.
68. Составьте программу, которая напечатает число p с заданной точностью.
69. Для заданной вещественной матрицы определите, образуют ли ее элементы упорядоченную последовательность при их переборе по спирали. Для определения факта упорядоченности части строки (столбца) используйте функции. Напишите алгоритм с описанием особых случаев, блок-схему и программу.
70. Напишите программу к предыдущей задачи, используя функции с открытыми параметрами-массивами.
71. Для заданной вещественной матрицы определите, образуют



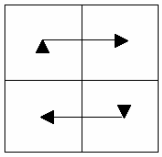
ли ее элементы упорядоченную последовательность при их переборе по схеме, указанной справа. Для определения факта упорядоченности части строки (столбца) используйте функции.

72. Пусть даны вещественные числа a, b, e (a<b, e>0). С точностью e вычислите интеграл $\int_a^b f(x)dx$, $f(x)=\frac{\sin x}{x}$, используя формулу трапеций $\int_a^b f(x)dx=\frac{h}{2}\left(f(a_0)+2f(a_1)+2f(a_2)+\ldots+2f(a_{n-1})+f(a_n)\right)$. Для обеспечения нужной точности воспользуйтесь правилом Рунге.
73. Определите все общие делители двух заданных натуральных чисел.
74. Напишите процедуру сложения двух дробей, результатом которой является несократимая правильная дробь. Используйте функцию для нахождения наибольшего общего делителя.
75. В двух целочисленных последовательностях $a_1, a_2, \ldots, a_n$ и $b_1, b_2, \ldots, b_m$ замените все элементы, следующие за элементом с максимальным значением, на значение минимального элемента. Используйте в процедуре открытый параметр-массив.
76. **Индивидуальное(!) задание**, которое передается преподавателю перед началом собеседования по этой теме:
Номер индивидуального задания определяет преподаватель!
Опишите постановку задачи, создайте математическую модель ее решения, разработайте блок-схему и работающую программу, проведите тестирование и отладку программы, обдумайте полученные результаты.

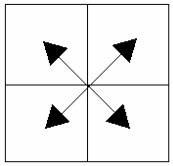
Индивидуальные задания

1. Составьте процедуру “сжатия” исходной последовательности символов, которая заменяет последовательность, состоящую из одинаковых символов, текстом вида x(k), где x - символ последовательности, k - число вхождений. Определите для указанной последовательности коэффициент сжатия (*отношение исходной длины последовательности к полученной*).
2. Пусть даны две матрицы A (mxn), B(mxn), состоящие из вещественных чисел. Необходимо получить матрицу **C** (mxn), где элемент C_{ij} равен сумме элементов i-й строки матрицы **A**, которые отсутствуют в j-м столбце матрицы B. Напишите функцию вычисления C_{ij}, использующую функцию проверки наличия числа в j-м столбце матрицы B.
3. Пусть даны две вещественные матрицы порядка n. Получите новую матрицу следующим способом (*для нахождения минимального элемента в указанной строке используйте функцию*): умножением минимального элемента каждой строки первой матрицы на наибольший элемент соответствующего столбца второй матрицы.

4. Пусть даны две вещественные матрицы порядка n. Получите новую матрицу следующим способом (*для нахождения произведения элементов в указанной строке используйте функцию*): прибавлением к элементам каждого столбца первой матрицы произведения элементов соответствующих строк второй матрицы.
5. Пусть дана вещественная квадратная матрица порядка 2n. Получите новую матрицу, переставляя ее блоки размером n так, как показано на рисунке справа. Для обмена четырех заданных фрагментов матрицы напишите функцию.



6. Пусть дана вещественная квадратная матрица порядка 2n. Получите новую матрицу, переставляя ее блоки размером n так, как показано на рисунке справа. Для обмена четырех заданных фрагментов матрицы напишите функцию.



7. Пусть дана прямоугольная матрица A(mxn), элементами которой являются целые числа. Замените все положительные четные числа на числа, являющиеся их «перевертышами». Составьте подпрограмму, получающую для заданного числа его «перевертыш» (*число* **a** *будем считать «перевертышем» числа* b, *если, читая число* **a** *справа налево, получаем число* b).
8. Напишите функцию, которая в заданной строке определяет количество вхождений в нее заданной подстроки. Для заданных строки и слова определите количество вхождений слова в строку.
9. Для заданного текста определите пару слов, буквенный состав которых наиболее схож. Используйте функцию для определения сходства буквенного состава двух слов.
10. Пусть дана матрица A(nxn). Упорядочьте строки по неубыванию сумм цифр элементов этой строки. Воспользуйтесь функцией, определяющей для каждого числа сумму его цифр.
11. Пусть дано n треугольников, заданных координатами своих вершин. Найдите пару треугольников, максимально удаленных друг от друга.
12. Пусть дано n прямоугольников, заданных координатами левой верхней и правой нижней вершины. Стороны прямоугольников параллельны осям координат. Определите пару прямоугольников с максимальной площадью пересечения. Напишите функцию для определения площади пересечения двух прямоугольников.
13. Пусть дано n отрезков на интервале (**A**, **B**). Определите часть интервала, который покрывается наибольшим количеством отрезков. Напишите функцию для определения количества отрезков, покрывающих заданный интервал.
14. Составьте процедуру деления с остатком для многочленов. Вычислите:

$$\frac{a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n}{\left(b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m\right)\left(b_m + b_{m-1} x + b_{m-2} x^2 + \ldots + b_0 x^m\right)},\ 2m < n.$$

15. Найдите все числа, которые равны сумме факториалов своих цифр. Например,

1!+4!+5! = 145. Определите верхнюю границу множества таких чисел. Используйте функцию, вычисляющую сумму факториалов цифр числа.

16. Уплотните матрицу A(nxn) влево и вверх. Для выявления нулевых строк и столбцов используйте подпрограмму.

3. Рекурсивные подпрограммы

77. Что такое рекурсивные подпрограммы?
78. В каком случае решение задачи можно выразить рекурсивным алгоритмом? Приведите пример.
79. Что такое стек? Как организовано заполнение стека и выбор элементов из стека?
80. Как связаны стек и рекурсия?
81. Что такое условие выхода из рекурсии? Для чего оно нужно?
82. Может ли рекурсивная процедура вызывать себя бесконечное число раз?
83. Можно ли рекурсивный алгоритм заменить нерекурсивным?
84. Что такое глубина рекурсии?
85. Что такое текущий уровень рекурсии?
86. Что такое рекурсивный спуск? Рекурсивный возврат?
87. Приведите три разных формы рекурсивной процедуры/ функции.
88. Пусть дана строка текста. Напечатайте этот текст в обратном порядке.
89. Для чего служит функция eoln?
90. Напишите алгоритм и программу для решения задачи: «Ханойские башни».
91. В чем заключается метод быстрой сортировки элементов массива?
92. Упорядочите элементы одномерного массива по неубыванию методом быстрой сортировки.
93. Вычислите определитель заданной матрицы, пользуясь формулой разложения по первой строке (*матрица* **B** *получается вычеркиванием из* **A** *первой строки и* k*-го столбца*): $\det A = \sum_{k=1}^{n} (-1)^{k+1} A_{1,k} \det B_k$.
94. По вещественному числу a >0 вычислите величину: $\frac{\sqrt[3]{a} - \sqrt[6]{a^2+1}}{1+\sqrt[7]{3+a}}$. Корни $y = \sqrt[k]{x}$ вычисляйте с точностью **E**=0,00001 по следующей итерационной формуле: $y_0 = 1;\ y_{n+1} = y_n + \frac{\left(\frac{x}{y_n^{k+1}} - y_n\right)}{k}$, n=0,1,2,... , приняв за ответ приближение y_{n+1}, для которого $|y_{n+1} - y_n| < E$.
95. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int_a^b x^m \ln^n x dx = \begin{cases} \dfrac{x^{m+1}}{m+1}\ln^n x - \dfrac{n}{m+1}\int_a^b x^m \ln^{n-1} x dx, n > 1, \\ x^{m+1}\left[\dfrac{\ln x}{m+1} - \dfrac{1}{(m+1)^2}\right], n = 1. \end{cases}$$

96. Получите количество размещений из 10 элементов 1, 2,..., 10 по 3 в каждом. Размещением называется выборка из n указанных элементов m неповторяющихся элементов.
97. Напишите рекурсивную функцию для нахождения биномиальных коэффициентов (для заданного $M \geq i \geq j > 0$ вычислите все C_j^i):

$$C_m^n = \begin{cases} 1, \quad m = 0, n > 0 \quad \vee \quad m = n \geq 0, \\ 0, \quad m > n > 0, \\ C_{m-1}^{n-1} + C_m^{n-1}, 0 < m < n. \end{cases}$$

98. Опишите рекурсивную функцию, которая по заданным границам a и b, заданной функции $f(x)$ и заданной точности e методом деления отрезка пополам находит с точностью e корень уравнения $f(x) = 0$ на отрезке $[a,b]$. Считать, что $e > 0$, $a < b$, $f(a) \cdot f(b) < 0$, $f(x)$ - непрерывная и монотонная на $[a,b]$.
99. Постройте синтаксический анализатор для понятия *идентификатор*. `<Идентификатор> ::= <буква>|<идентификатор> (<цифра>|<буква>).`

Литература

1. Абрамов С.А. Начала информатики / С.А.Абрамов, Е.В.Зима. – М. : Наука, 1990.
2. Дагене В.А. и др. 100 задач по программированию / В.А.Дагене и др. - М.: Просвещение, 1993.
3. Епанешников А.М. Программирование в среде Turbo Pascal 7.0 / А.М. Епанешников, В.А.Епанешников. – 3-е изд. – М.: Диалог-МИФИ, 1996.
4. Зубов В.С. Программирование на языке Turbo Pascal / В.С.Зубов. – М. : Филинъ, 1997.
5. Кассера В. TurboPascal 7.0 / В.Кассера, Ф. Кассера. - 2003.
6. Марченко А.И. Программирование в среде Turbo Pascal 7.0 / А.И.Марченко, Л.А.Марченко. - М.: Бином Универсал, К. : ЮНИОР, 1997.
7. Пильщиков В.Н. Сборник упражнений по языку Паскаль / В.Н.Пильщиков. - М. : Наука, 1989.
8. Программирование на языке Паскаль: задачник/под ред. Усковой О.Ф. – СПб: Питер, 2002.
9. Фаронов В.В. Turbo Pascal 7.0. Начальный курс / В.В.Фаронов. – М. : Нолидж, 1998.
10. Хершель Р. Турбо Паскаль / Р.Хершель. – Вологда : МП МИК, 1991.

Составители: Васильев Валерий Викторович,
Хливненко Любовь Владимировна

Редактор Тихомирова О.А.