Вспомогательные алгоритмы

Содержание

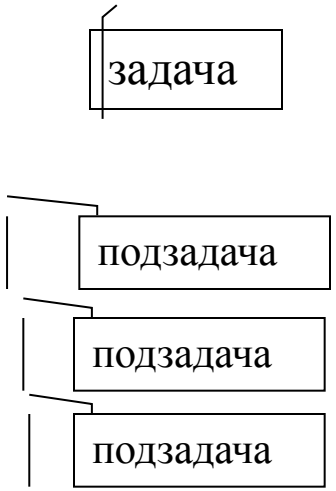
1. Вопросы для подготовки к зачету
2. Метод пошаговой детализации
3. Подпрограммы
   - 3.1. Преимущества использования подпрограмм
   - 3.2. Организация подпрограмм в Паскале
   - 3.3. Процедуры
   - 3.4. Задача «Две лодки»
   - 3.5. Задание
   - 3.6. Функции
4. Локальные и глобальные переменные
5. Рекурсия

Вопросы для подготовки к зачету

1. В чем заключается метод пошаговой детализации?
2. Что такое подпрограмма? Для чего она нужна? Какие преимущества она дает?
3. Что такое процедура? Как она оформляется? Как осуществляют вызов процедур?
4. Что такое функция? Как она оформляется? Как вызывают функцию?
5. Какие бывают параметры? Может ли быть подпрограмма без параметров?
6. Как оформляют (называют) параметры для передачи исходных данных и получения результата?
7. В чем состоит главное отличие функций и процедур? Как выбрать какой вид подпрограммы в данном случае предпочтительней?
8. Что такое локальные и глобальные переменные?

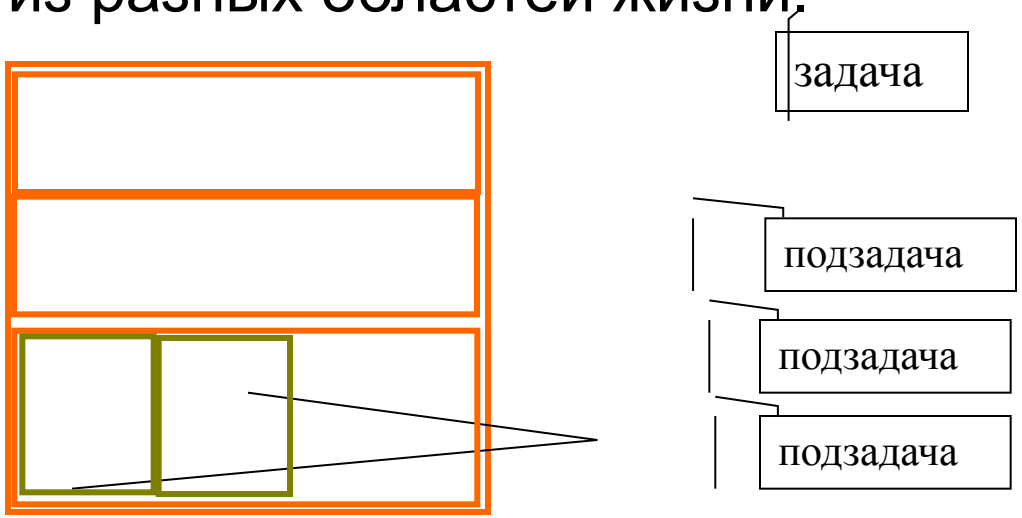
Метод пошаговой детализации

Эффективным методом построения алгоритмов является метод пошаговой детализации (последовательного построения). При этом сложная задача разбивается на ряд более простых задач. Для каждой подзадачи составляется свой, с точки зрения решения основной задачи ***вспомогательный алгоритм***.



Метод пошаговой детализации

Эти подзадачи могут, в свою очередь, потребовать разбиения на еще более простые задачи и т.д. В результате некоторые ***вспомогательные*** алгоритмы могут стать ***основными*** по отношению к вспомогательных более низкого уровня. Процесс пошаговой детализации заканчивается, когда задачи очередного уровня окажутся совсем простыми. Метод пошаговой детализации универсален. Он применим для решения задач из разных областей жизни.



Подпрограммы

При записи программы для компьютера вспомогательные алгоритмы обычно оформляют как подпрограммы. Правила обращения к ним и возврата в основную программу определяются конкретным языком программирования.

Подпрограмма - это последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы.

Подпрограммы общего назначения могут объединяться в библиотеки подпрограмм, а иногда образовывать набор стандартных функций.

Метод пошаговой детализации путем разбиения задачи на подзадачи лежит в основе так называемого структурного программирования.

Преимущества использования подпрограмм

Концепция процедуры позволяет выделить подзадачу как явную подпрограмму, делает структуру задачи ***более понятной*** и дает возможность вести разработку одновременно ***коллективу*** программистов.

Подпрограмма является одним из фундаментальных инструментов, оказывающих влияние ***на стиль, качество и надежность разработки*** программных средств.

Использование подпрограмм приводит к улучшению “читаемости” программы: как правило, в коротких блоках разбираться легче, чем в длинных. При разделении этапов разработки на подпрограммы программу будет легче ***передавать*** другому человеку и ***легче ее проверить***.

Организация подпрограмм в Паскале

Function

В Паскале существует два вида подпрограмм – ***функции и процедуры***. Через имя функции возвращается значение определенного типа, поэтому обращение к ней размещают в выражении. В отличии от функции, с именем процедуры не связано возвращение значения, поэтому вызов процедуры представляет собой оператор.

Procedure

Процедуры

Процедура – это специальным образом оформленная последовательность операторов, которой присвоено имя. Процедура хранится в разделе описаний программы, программа по отношению к ней выступает как внешняя. Процедура может быть вызвана для выполнения в исполнительной части программы из любых ее точек по имени.

Описание процедуры

Описание процедуры выглядит как программа, но вместо заголовка программы употребляется заголовок процедуры. Структура процедуры копирует структуру программы в целом, за исключением завершающей процедуру точки с запятой вместо точки после последнего *End* программы. Порядок следования разделов описаний подчиняется тем же правилам, по которым строится вся программа.

```
Заголовок процедуры;
Раздел описаний
Begin
    операторы;
end;
```

Общий вид описания процедуры

Пояснение: квадратные скобки в описании показывают необязательность этой части оператора.

Procedure <Имя_Процедуры>[(ПараметрЗначение2 : ТипЗначение2; . . .
Var параметрПеременная2 :ТипПеременная2;. . .)] ;

Формальные параметры

[Label <Метки_Используемые_в_Процедуре>]
[Const <Константы_Используемые_в_Процедуре>]
[Type <Типы_Используемые_в_Процедуре> **]**
[Var <Переменные_Используемые_в_Процедуре>**]**
[<Описание вложенных процедур и (или) функций>]
[Procedure . . .]
[Function . . .]

Декларативная часть процедуры

Begin
< Операторы >
End;

Исполнительная часть процедуры

Вызов процедуры

Процедура может быть активирована (вызвана) в исполнительной части программы по ее имени. При вызове процедура будет содержать список параметров, если он присутствовал при описании этой процедуры в разделе описаний. Типы используемых параметров при вызове процедуры не указываются.

<Имя_Процедуры> [(< Список_Параметров >)] ;

Например,

необходимо написать процедуру вычисления корней квадратного уравнения $ax^2 + bx + c = 0$. Заголовок этой процедуры будет выглядеть следующим образом:

```
Procedure Korni (A,B,C :Real;Var x1,x2: Real);
```

Формальные параметры

где *A*, *B* и *C* – коэффициенты квадратного уравнения, *x1* и *x2* – корни этого уравнения.

Различают параметры-значения (они используются для передачи в подпрограмму исходных данных) и параметры-переменные (процедура может получать значения от вызывающей программы, а также возвращать в программу новые значения). Перед параметрами – переменными ставят слово VAR.

Вызов процедуры осуществляется по ее имени.

```
Korni ( A1, B1, C1, x1, x2 );
```

Фактические параметры (или аргументы)

Параметры при вызове отделяются друг от друга запятыми, а весь список заключается в скобки. Аргументы подпрограммы должны соответствовать формальным параметрам по количеству, порядку следования и типам.

Задача «Две лодки»

Две моторные лодки равномерно двигались по реке в направлении к озеру, в которое река впадает. Поравнявшись, они начали двигаться равноускоренно. Какая из лодок раньше дойдет до озера?

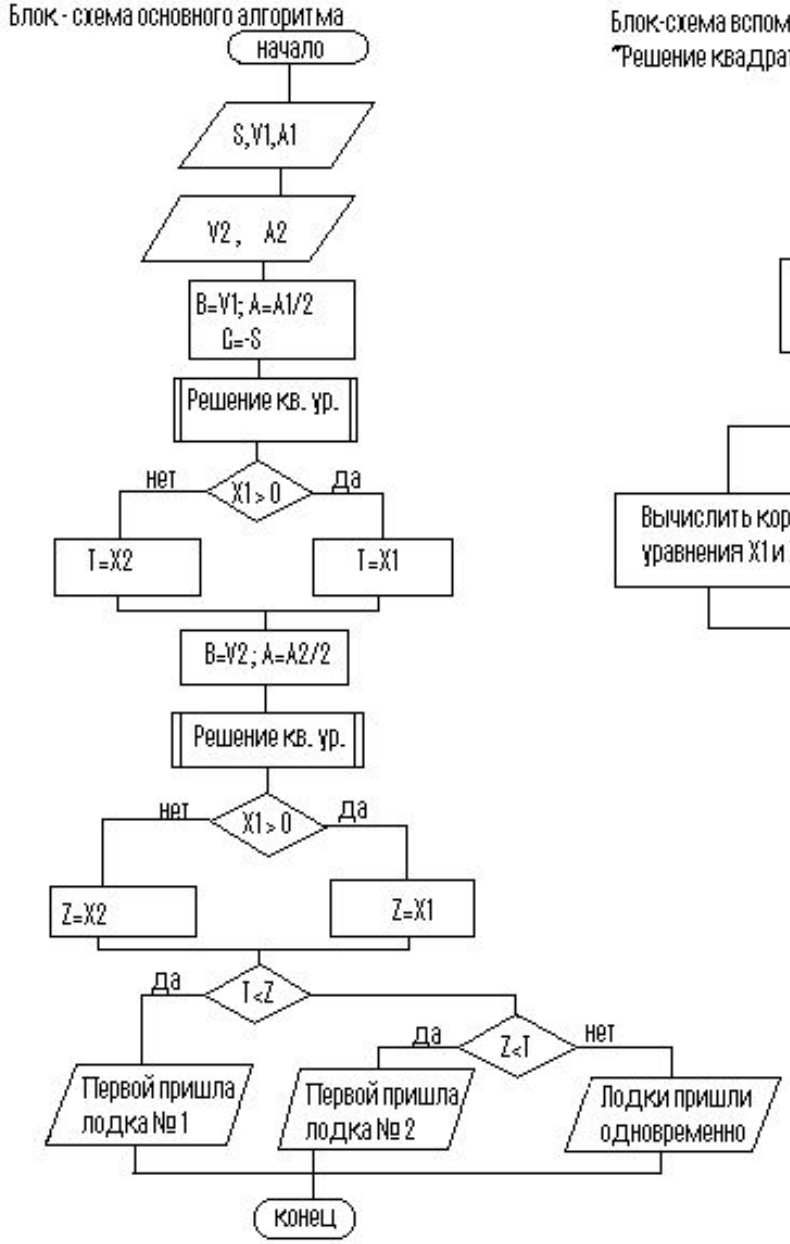
Построение математической модели

Лодки будем считать точками, реку и движение лодок – прямолинейными. Исходными данными являются начальные скорости лодок (обозначим их V1 и V2), ускорения лодок (A1 и A2), расстояние до озера (S). Результатом является сообщение, какая лодка раньше дойдет до озера или что лодки придут одновременно. Время находится из квадратного уравнения

$$vt+at^2/2=S$$

Для решения этого уравнения составим вспомогательный алгоритм. Поскольку скорости и ускорения положительны, лодки обязательно доплывут до озера (квадратное уравнение имеет два корня, по теореме Виета - один из них отрицательный, другой положительный, из соображений физического смысла выберем в качестве t положительный корень).См. блок - схему алгоритма.

Блок-схема к задаче о двух лодках



Программа к задаче «Две лодки»

```
program lodki;
var v2,v1,t2,t1,s,a2,a1,d,z1,z2:real;
procedure korni(a,b,c:real;var x,y:real);
begin
  d:=b*b-4*a*c;
  if d>=0 then
        begin
    d:=sqrt(d);
    x:=(-b-d)/(2*a);
    y:=(-b+d)/(2*a);
        end
  else writeln('Нет корней');
end;

begin {начало}
write('Введите расстояние до озера ');
readln(s);
write('Введите скорость первой и
второй лодки ');
readln(v1,v2);
write('Введите ускорение первой и
второй лодки ');
readln(a1,a2);
korni(a1/2,v1,-s,t1,t2);
if t1<0 then t1:=t2;
korni(a2/2,v2,-s,z1,z2);
if z1<0 then z1:=z2;
if z1<t1 then writeln('Лодка 2 пришла
раньше')
      else if t1<z1 then writeln('Лодка 1
пришла раньше')
           else writeln('Обе лодки
пришли одновременно');
readln;
end.
```

Выполнить

Задания

1. Напишите программу поиска наименьшего значения из трех заданных величин, используя не более двух сравнений, перестановку значений оформить в виде процедуры SWAP(X,Y)

2. Напишите программу, в которой вычисляются поэлементные суммы одномерных массивов **A+B, B+C, A+C, A+B+C.** Элементы всех массивов целого типа. Размерность всех массивов одинакова. Ввод, суммирование и вывод массива оформите в виде процедур.

3. Напишите программу сортировки массива методом «пузырька», используйте в ней процедуру SWAP.

Задания

4. Составить программу сравнения значений минимальных элементов двух одномерных целочисленных массивов разных размеров. В программу включить процедуры заполнения, распечатки, поиска минимальных элементов.

Текст программы

5. Отсортировать по возрастанию одномерный массив целых значений методом последовательных минимумов. Использовать процедуры заполнения, распечатки, поиска минимального и перестановки.

Текст программы

Функции

Функция - это подпрограмма, определяющая единственное целое, символьное, логическое, вещественное, строковое или ссылочное значение, являющееся результатом работы функции.

Описание функции

Все сказанное ранее о процедурах справедливо и для функций. Вместе с тем имеется ряд отличий. Одно из них чисто внешнее: заголовок начинается с зарезервированного слова ***Function.***

Function <Имя_Функции>[(< Список_Парам>)]:<Тип_Результата>;

Как и у процедуры, список параметров функции может быть пустым, в этом случае отсутствуют и сами скобки.

Содержательная часть функции, как и у процедуры, представляет собой блок, который содержит раздел описаний и исполнительную часть.

Особенности функции

1. В заголовке функций ***явно*** указывается тип результата, вычисленного с помощью этой функции. Вызов функции используется в выражениях, следовательно, при анализе выражения обязательно знать тип результата, вычисляемого функцией.
2. В теле функции обязательно должен присутствовать оператор присваивания, ***в левой части*** которого стоит идентификатор этой функции. Такое присваивание “возвращает” результат функции. Если такой оператор отсутствует, то значение, вычисляемое функцией, не определено, что приводит к аварийному завершению программы.
3. Если имя функции появляется где-либо в выражении внутри самой функции, то речь идет об ее ***рекурсивном*** выполнении.

Пример

```
{Описание функции выбора большего из двух аргументов.}
Function MAX (A,B:real): real;
Begin
    If A > B then MAX:= A
        Else MAX:= B
End;
```

Задача. Сравнить значения минимальных элементов двух массивов разных размеров

```
program poisk_min_v_dvux_massivax;
uses crt;
const n=255;
type massiv=array [1..n] of integer;
var a,b:massiv;na,nb:byte;
    mina,minb:integer;
procedure zap(var x:massiv;k:byte);
...
procedure print(var x:massiv;k:byte);
…
function min(var x:massiv;k:byte):integer;
var i:byte; m:integer;
begin
  m:=x[1];
  for i:=2 to k do
    if x[i]<m then m:=x[i];
  min:=m;
end;
```

```
begin
  clrscr; randomize;
  write('Кол-во эл-тов в массиве A: ');
  readln(na);
  write('Кол-во эл-тов в массиве B: ');
  readln(nb);
  zap(a,na); zap(b,nb);
  writeln('Massiv A:');
  print(a,na);
  writeln('Massiv B:');
  print(b,nb);
  if min(a,na)<min(b,nb) then
       writeln(' Наименьший в массиве A')
  else   if min(b,nb)<min(a,na) then
        writeln(' Наименьший в массиве B')
        else  writeln('Минимальные равны');
  readln;
end.
```

Задача. Сравнить значения минимальных элементов двух массивов разных размеров. Используем функцию

```
program poisk_min_v_dvux_massivax;
uses crt;
const n=255;
type massiv=array [1..n] of integer;
var a,b:massiv;na,nb:byte;
    mina,minb:integer;
procedure zap(var x:massiv;k:byte);
...
procedure print(var x:massiv;k:byte);
…
function min(var x:massiv;k:byte):integer;
var i:byte; m:integer;
begin
  m:=x[1];
  for i:=2 to k do
    if x[i]<m then m:=x[i];
  min:=m;
end;
```

```
begin
  clrscr; randomize;
  write('Кол-во эл-тов в массиве A: ');
  readln(na);
  write('Кол-во эл-тов в массиве B: ');
  readln(nb);
  zap(a,na); zap(b,nb);
  writeln('Massiv A:');
  print(a,na);
  writeln('Massiv B:');
  print(b,nb); mina:=min(a,na);minb:= min(b,nb);
  if mina<minb then
       writeln('Наименьший в массиве A')
  else   if minb<mina then
         writeln(' Наименьший в массиве B')
         else  writeln('Минимальные равны');
  readln;
end.
```

Локальные и глобальные переменные

Если переменные описаны в тексте функции или процедуры, то их называют ***локальными переменными***, т. к. они доступны только внутри этой подпрограммы. Для основной программы они не существуют.

Переменные или константы, определенные в разделе описаний основной программы перед описанием других подпрограмм доступны всем программным единицам и являются ***глобальными***.

Любой подпрограмме доступны глобальные переменные, за исключением тех, с чьими именами совпадают имена ее параметров или локальных переменных

Рекурсия

Рекурсия – это способ организации вычислительного процесса, при котором процедура или функция в процессе выполнения входящих в ее состав операторов обращается сама к себе. Это возможно, т.к. при каждом обращении под параметры и локальные переменные память резервируется в специальной области, называемой **стеком.**

Важным моментом при написании рекурсивных подпрограмм является организация выхода из подпрограммы, т.е в подпрограмме должно быть условие выполнение которого не повлечет за собой нового вызова рекурсивной функции или процедуры.

Рекурсия

Рекурсивные подпрограммы имеют одну из форм: прямую рекурсию и косвенную рекурсию. В первом случае подпрограммы сама себя вызывает, во втором – вызов происходит через вызов другой подпрограммы, которая обращается к вызывающей подпрограмме. (Пример. Если A, B – подпрограммы , то схема косвенной рекурсии может быть такой A □ B □ A).

Рекурсия

В случае косвенной рекурсии возникает проблема: как и где описывать вызывающий модуль. По правилам языка Паскаль каждая подпрограмма должна быть описана ***до ее вызова.*** Но если А вызывает В, а В вызывает А, то получается замкнутый круг. Для подобных ситуаций принято следующее правило: Один из рекурсивных модулей описывается ***предварительно*** следующим образом:

PROCEDURE P(список параметров); FORWARD;

Где **FORWARD** – ключевое слово, которое указывает, что текст процедуры P помещен ниже. Список параметров и тип результата (для FUNCTION) включается только в это предварительное описание и опускается в заголовке соответствующей п/п.

Задания

1. Вычисление факториала;
2. Вычисление Чисел Фибоначчи;
3. Выдать в обратном порядке цифры целого положительного числа N;
4. Вычислить положительную степень числа X *(на дом)*

Рекурсивные подпрограммы

```
Program recursi;
uses crt;
var l,m:integer;
function Factorial (n:integer):integer;
  {рекурсивная функция вычисления факториала}
  begin
    if N=0 then
    Factorial:= 1 else Factorial:=Factorial(n-1)*N
  end;
function Fib(k:integer):integer;
  {рекурсивная функция вычисления к- го числа Фибоначчи
  возвращает -1, если число не существует(к<=0)}
  begin
    if (k=1) or (k=2) then Fib:=1
       else
         if k>2 then
           Fib:=Fib(k-2)+Fib(k-1)
              else
           Fib:=-1;
   end;
```

```
procedure Revers(n:integer);
    {Выдает на экран в обратном
порядке цифры числа n}
    begin
      write (n mod 10);
      if (n div 10) <> 0 then
           Revers(n div 10)
    end;
```

Вычисление факториала

```
function Factorial (n:integer):longint;
  {рекурсивная функция вычисления факториала}
  begin
    if N=0 then
    Factorial:= 1 else Factorial:=Factorial(n-1)*N
  end;
```

Функция вычисления к- го числа Фибоначчи

```
function Fib(k:integer):integer;
  {рекурсивная функция вычисления к- го числа}
  begin
    if (k=1) or (k=2) then Fib:=1
       else
          Fib:=Fib(k-2)+Fib(k-1)
   end;
```

Выдает на экран в обратном порядке цифры числа n

```
procedure Revers(n:integer);
  {Выдает на экран в обратном порядке цифры числа n}
  begin
    write (n mod 10);
    if (n div 10) <> 0 then
          Revers(n div 10)
  end;
```

Задача. Сравнить значения минимальных элементов двух массивов разных размеров. Используем процедуру

```
program poisk_min_v_dvux_massivax;
uses crt;
const n=255;
type massiv=array [1..n] of integer;
var a,b:massiv;na,nb:byte;
    mina,minb:integer;
procedure zap(var x:massiv;k:byte);
...
procedure print(var x:massiv;k:byte);
...
procedure min_v_massive
(var x:massiv;var min:integer;k:byte);
var i:byte;
begin
  min:=x[1];
  for i:=2 to k do
    if x[i]<min then min:=x[i];
end;
```

```
begin
  clrscr; randomize;
  write(''Кол-во эл-тов в массиве A: ');
  readln(na);
  write('Кол-во эл-тов в массиве B: ');
  readln(nb);
  zap(a,na); zap(b,nb);
  writeln('Massiv A:');
  print(a,na);
  writeln('Massiv B:');
  print(b,nb);
  min_v_massive(a,mina,na);
  min_v_massive(b,minb,nb);
  if mina<minb then writeln('Наименьший в
массиве A')
  else   if minb<mina then writeln('Наименьший
в массиве B')
        else  writeln('Минимальные равны');
  readln;
end.
```

Задача. Сортировки массива. Используем процедуры

```
program sotirovka_massiva_min;
uses crt;
const n=255;
type massiv=array [1..n] of integer;
var a:massiv;na,i,nmin:byte;
    mina:integer;
procedure zap(var x:massiv;k:byte);
...
procedure print(var x:massiv;k:byte);
...
procedure min_v_massive(var
x:massiv;var min:integer;var
n_min:byte;nach,k:byte);
var i:byte;
begin
  min:=x[nach];n_min:=nach;
  for i:=nach+1 to k do
    if x[i]<min then begin
min:=x[i];n_min:=i;end;
end;
```

```
procedure swap(var x,y:integer);
var p:integer;
begin
  p:=x;x:=y;y:=p;
end;
begin
  clrscr; randomize;
  write('kol-vo elemetov v massive A: ');
  readln(na);
  zap(a,na);
  writeln('Massiv A:');
  print(a,na);
  for i:=1 to n-1 do
    begin
      min_v_massive(a,mina,nmin,i,na);
      swap(a[i],a[nmin]);
    end;
  writeln('Otsortirovan massiv A:');  print(a,na);
  readln;
end.
```