Программирование на ЯВУ

1. Структурное программирование. Типовые структуры.

Структурное программирование — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 70-х годах XX века Э. Дейкстрой, разработана и дополнена Н. Виртом.

В соответствии с данной методологией
Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:

1. последовательное исполнение — однократное выполнение операций в том порядке, в котором они записаны в тексте программы;
2. ветвление — однократное выполнение одной из двух или более операций, в зависимости от выполнения некоторого заданного условия;
3. цикл — многократное исполнение одной и той же операции до тех пор, пока выполняется некоторое заданное условие (условие продолжения цикла).

В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.
Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) могут оформляться в виде т. н. подпрограмм (процедур или функций). В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция вызова подпрограммы. При выполнении такой инструкции выполняется вызванная подпрограмма, после чего исполнение программы продолжается с инструкции, следующей за командой вызова подпрограммы.
Разработка программы ведётся пошагово, методом «сверху вниз».

Сначала пишется текст основной программы, в котором, вместо каждого связного логического фрагмента текста, вставляется вызов подпрограммы, которая будет выполнять этот фрагмент. Вместо настоящих, работающих подпрограмм, в программу вставляются «заглушки», которые ничего не делают. Полученная программа проверяется и отлаживается. После того, как программист убедится, что подпрограммы вызываются в правильной последовательности (то есть общая структура программы верна), подпрограммы-заглушки последовательно заменяются на реально

работающие, причём разработка каждой подпрограммы ведётся тем же методом, что и основной программы. Разработка заканчивается тогда, когда не останется ни одной «затычки», которая не была бы удалена. Такая последовательность гарантирует, что на каждом этапе разработки программист одновременно имеет дело с обозримым и понятным ему множеством фрагментов, и может быть уверен, что общая структура всех более высоких уровней программы верна. При сопровождении и внесении изменений в программу выясняется, в какие именно процедуры нужно внести изменения, и они вносятся, не затрагивая части программы, непосредственно не связанные с ними. Это позволяет гарантировать, что при внесении изменений и исправлении ошибок не выйдет из строя какая-то часть программы, находящаяся в данный момент вне зоны внимания программиста.

Теорема Бёма — Якопини — положение структурного программирования, согласно которому любой исполняемый алгоритм может быть преобразован к структурированному виду, то есть такому виду, когда ход его выполнения определяется только при помощи трёх структур управления: последовательной (англ. sequence), ветвлений (англ. selection) и повторов или циклов (англ. repetition, cycle).

2. Концепция типов в Турбо Паскале

В математике принято классифицировать переменные в соответствии с некоторыми важными характеристиками. Производится строгое разграничение между вещественными, комплексными и логическими переменными, между переменными, представляющими отдельные значения и множество значений и так далее.

При обработке данных на ЭВМ такая классификация еще более важна. В любом алгоритмическом языке каждая константа, переменная, выражение или функция бывают определенного типа.

В языке ПАСКАЛЬ существует правило: тип явно задается в описании переменной или функции, которое предшествует их использованию. Концепция типа языка ПАСКАЛЬ имеет следующие основные свойства:
- любой тип данных определяет множество значений, к которому принадлежит константа, которые может принимать переменная или выражение, или вырабатывать операция или функция;
- тип значения, задаваемого константой, переменной или выражением, можно определить по их виду или описанию;
- каждая операция или функция требует аргументов фиксированного типа и выдает результат фиксированного типа.

Отсюда следует, что транслятор может использовать информацию о типах для проверки вычислимости и правильности различных конструкций.

Тип определяет:
- возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;
- внутреннюю форму представления данных в ЭВМ;
- операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

Обязательное описание типа приводит к избыточности в тексте программ, но такая избыточность является важным вспомогательным средством разработки программ и рассматривается как необходимое свойство современных алгоритмических языков высокого уровня. В языке ПАСКАЛЬ существуют скалярные и структурированные типы данных. К скалярным типам относятся стандартные типы и типы, определяемые пользователем.

Стандартные типы включают целые, действительные, символьный, логические и адресный типы. Типы, определяемые пользователем, - перечисляемый и интервальный. Структурированные типы имеют четыре разновидности: массивы, множества, записи и файлы. Кроме перечисленных, TURBO PASCAL включает еще два типа - процедурный и объектный.

Из группы скалярных типов можно выделить порядковые типы, которые характеризуются следующими свойствами:
- все возможные значения порядкового типа представляют собой ограниченное упорядоченное множество;
- к любому порядковому типу может быть применена стандартная функция Ord, которая в качестве результата возвращает порядковый номер конкретного значения в данном типе;
- к любому порядковому типу могут быть применены стандартные функции Pred и Succ, которые возвращают предыдущее и последующее значения соответственно;
- к любому порядковому типу могут быть применены стандартные функции Low и High, которые возвращают наименьшее и наибольшее значения величин данного типа.

В языке ПАСКАЛЬ введены понятия эквивалентности и совместимости типов. Два типа T1 и T2 являются эквивалентными (идентичными), если выполняется одно из двух условий:
- T1 и T2 представляют собой одно и то же имя типа;
- тип T2 описан с использованием типа T1 с помощью равенства или последовательности равенств.

Например:

```
type
  T1 = Integer;
  T2 = T1;
  T3 = T2;
```

Менее строгие ограничения определены совместимостью типов. Например, типы являются совместимыми, если:
- они эквивалентны;
- являются оба либо целыми, либо действительными;
- один тип - интервальный, другой - его базовый;

- оба интервальные с общим базовым;

один тип - строковый, другой - символьный.

В ТУРБО ПАСКАЛЬ ограничения на совместимость типов можно обойти с помощью приведения типов. Приведение типов позволяет рассматривать одну и ту же величину в памяти ЭВМ как принадлежащую разным типам. Для этого используется конструкция

Имя_Типа(переменная или значение). Например,

Integer('Z')

представляет собой значение кода символа 'Z' в двухбайтном представлении целого числа, а Byte(534) даст значение 22, поскольку целое число 534 имеет тип Word и занимает два байта, а тип Byte занимает один байт, и в процессе приведения старший байт будет отброшен.

3. Оператор условного перехода и программирование разветвляющихся алгоритмов

В Паскале имеется возможность нелинейного хода программы, т.е. выполнения операторов не в том порядке, в котором они записаны. Такую возможность нам предоставляют разветвляющиеся алгоритмы. Они могут быть реализованы одним из трех способов: с использованием операторов перехода, условного оператора или оператора выбора.

1. Оператор перехода

Оператор перехода имеет вид

GOTO <метка>.

Он позволяет передать управление непосредственно на нужный оператор программы. Перед этим оператором должна располагаться метка отделенная от него двоеточием. В Турбо Паскале в качестве меток выступают либо целые числа от 0 до 9999, либо идентификаторы. Все метки должны быть описаны в разделе объявления меток следующим образом:

label <список меток через запятую> ;

2. Условный оператор IF

Условный оператор включает в себя операторы, которые выполняются или не выполняются в зависимости от записанного в операторе условия. Оператор имеет вид:

IF "условие" Then "оператор1" Else "оператор2";

где "условие" - выражение логического типа;

"оператор1" выполняется, если условие верно (True),

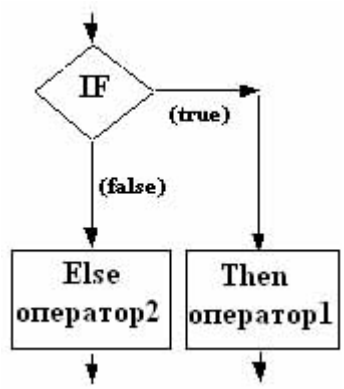
"оператор2" выполняется, если условие не верно (False).

Например, вычисление квадратного корня из числа "a" проводится при условии a>=0, операторами:

```
IF  a>=0 Then b := Sqrt(a)
Else
  begin
    WriteLn('a<0');
    Readln;
    Halt
  end;
```

Оператор Halt прекращает выполнение программы.

Схема выполнения условного оператора имеет вид:



В условном операторе может отсутствовать блок Else оператор2; т. е. условный оператор может иметь вид:

IF "условие" Then "опертор1";

например: *IF a<0 Then a:=abs(a);*
Приведем пример программы определения весовой категории в зависимости от веса спортсмена.

```
PROGRAM  VES;              { определение весовой категории спортсмена }
                          Условная схема программы
CONST A1='легкая категория';
      A2='средняя категория';
      A3='тяжелая категория';
      A4='сверхтяжелая категория';
var
  V : integer;
BEGIN
  Write('введите вес спортсмена V = '); ReadLn(v);
  if V<62 then WriteLn(A1)          {вложенный условный оператор }
  else if V<75 then WriteLn(A2)
       else if V<88 then WriteLn(A3)
            else WriteLn(A4)
  writeln('Нажмите Enter');
  readln;
END.
```

Условный оператор может применяться для идентификации (распознавания) объекта по определенным признакам составляющих его элементов. Например, если объектом является треугольник, то элементами объекта могут быть: 1) три его угла (a, b, c); 2) три его стороны (a1, b1, c1); и т. д.

Признаками являются значения элементов по которым производится идентификация, например, для углов: 1) один угол > 90 - (один признак); 2) три угла < 90 - (три признака); и т. д.
В результате идентификации объект получает имя. Например, треугольник - остроугольный, либо тупоугольный и т. д.
Если идентификация проводится по одному признаку для нескольких элементов, то несколько условий связываются служебным словом "or", например:
If (a > 90) or (b > 90) or (c > 90) then writeln ('Треугольник - тупоугольный');
Если идентификация проводится по нескольким признакам, число которых равно числу элементов, то несколько условий связываются служебным словом "and", например:
If (a < 90) and (b < 90) and (c < 90) then
writeln('Треугольник - остроугольный');

Если имя объекта составное, то добавляются признаки для идентификации второй части имени и применяются вложенные условные операторы, например, для равнобедренного треугольника:

```
If (a < 90)and (b < 90) and(c < 90) then
   If (a=b) or (b=c) or (a=c) then
     writeln('Треугольник - остроугольный и равнобедренный')
   else writeln('Треугольник - остроугольный');
```

Условный оператор можно применять для контроля правильности вводимых данных, например:

```
If (a+b+c)<>180 then
begin
writeln('Сумма углов <> 180');
end;
```

Если для идентификации объекта достаточно меньшего числа признаков, чем число элементов, то условия, связанные "and" группируются, а группы соединяются служебным словом "or". Например, четырехугольник имеет элементами четыре стороны (a, b, c, d), а его имя устанавливается по двум признакам (равенство двух пар сторон), тогда можно использовать операторы:

```
If ((a=b) and (c=d)) or ((a=c) and (b=d)) or ((a=d) and (b=c)) then
   writeln('Параллелограмм');
```

3. Оператор выбора CASE

Оператор служит для выбора одного из помеченных вариантов действия (операторов), в зависимости от значения "параметра". Оператор имеет вид:

```
Case "параметр" Of
"список помеченных операторов"
Else "оператор"
End;
```

Здесь "параметр" - выражение или переменная порядкового типа.
Из "списка помеченных операторов" выполняется оператор с меткой, включающей значение "параметра", иначе оператор после слова Else.
Конструкция Else "оператор" может отсутствовать, тогда "оператор" будет иметь вид: Begin "операторы" end;

Пример операторов для определения порядка целого числа N от 0 до 999:

```
case N of
0..9     : writeln('однозначное');
10..99   : writeln('двузначное');
100..999 : writeln('трехзначное')
else writeln('Число "N" не входит в указанный диапазон')
end;
```

4. Оператор цикла с постусловием

Оператор цикла с постусловием организует выполнение цикла, состоящего из любого количества операторов неизвестное заранее количество раз. Выход из цикла осуществляется, если некоторое логическое выражение окажется истинным. Так как истинность логического оператора проверяется в конце, тело цикла выполняется хотя бы один раз.

Структура оператора:

Repeat
<оператор_1>;
<оператор_2>;
. . .
<оператор_N>;
Until <>;

В этой структуре:

<оператор_1>;<оператор_2>;. . . <оператор_N>; — тело цикла.

<условие> — логическое выражение, ложность которого проверяется после выполнения тела цикла.

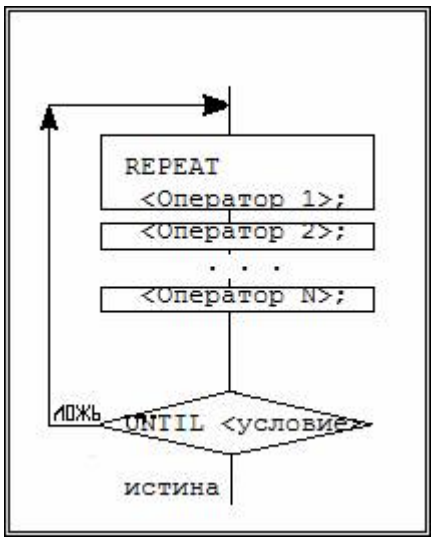
Порядок выполнения оператора:

Выполняются операторы, следующие за служебным словом Repeat. После этого проверяется условие. Если условие ложно, то происходит возвращение к выполнению операторов, следующих за служебным словом Repeat и снова проверяется условие. Если условие истинно, то выполнение тела цикла прекращается.

Оператор цикла с постусловием "звучит" так:

Повторять тело цикла пока не выполнится условие

В цикле Repeat тело цикла выполняется по крайней мере один раз.



Пример программы:
Вычисление суммы S=1+1/2+1/3+…+1/50

```
Program primer;
Var n:Integer;
 s:Real;
Begin
s:=0;
n:=1;
Repeat
s:=s+1/n;
n:=n+1;
Until n>50;
writeln(s);
End.
```

5. Оператор цикла с предусловием

Оператор цикла с предусловием организует выполнение одного (возможно составного) оператора неизвестное число раз. Выход из цикла осуществляется, если некоторое логическое выражение окажется ложным. Так как истинность логического выражения проверяется вначале, то тело цикла может не выполниться ни разу.
Структура оператора

While <условие> do <оператор>;

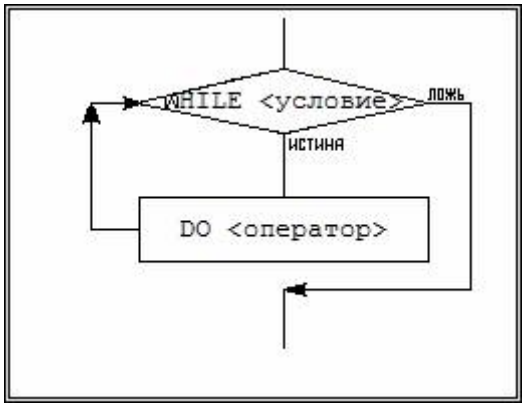
В этой структуре:

<условие> — логическое выражение, истинность которого проверяется вначале выполнения циклического оператора;

<оператор> — любой выполняемый оператор языка (в том числе и составной, т.е. последовательность операторов, заключенная в операторные скобки Begin — End).

Порядок выполнения оператора:

Пока условие истинно выполняется оператор, следующий за служебным словом do. Как только условие становится ложно, выполнение оператора цикла прекращается.



Пример применения:
Вычислить сумму S=1+1/2+1/3+…+1/50.

Program prim;
Var s:real

```
    i:Integer;
Begin
s:=0;
i:=1;
While i<=50 do
        Begin
        s:=s+1/n;
        n:=n+1;
        End;
writeln(s);
End.
```

6. Оператор цикла с заголовком

Оператор цикла с параметром организует выполнение одного оператора заранее известное количество раз.

Структура оператора:

Существуют два варианта оператора

Вариант первый

For i:=<start> to <finish> do <onepamop>;

Вариант второй

For i:=<start> downto <finish> do <onepamop>;

В этих структурах:

i — параметр цикла;

<start> — начальное значение параметра;

<finish> — конечное значение параметра;

<оператор> — тело цикла;

Тип переменной цикла i и значений <start > и <finish> должен быть порядковым!
Порядок выполнения оператора:

1. Вычисляются и запоминаются начальное — start, и конечное — finish, значения параметра цикла. Start и finish могут быть представлены в виде конкретного значения (в этом случае нет необходимости в вычислениях) или в виде выражения, значение которого вычисляется в начале выполнения цикла.

2. Параметру цикла i присваивается значение start.

3. Значение параметра цикла i сравнивается со значением finish. Оператор "тело цикла" будет выполняться при выполнении следующего условия:
   первый вариант оператора: i<=finish;
   второй вариант оператора: i>=finish;
В противном случае происходит прекращение выполнения циклического оператора.

4. Параметру цикла присваивается:
   первый вариант оператора: следующее большее значение;
   второй вариант оператора: следующее меньшее значение.

5. Выполняется пункт 3 данной схемы.

Часто говорят, что первый вариант оператора цикла с параметром — цикл с возрастающим параметром; второй вариант — с убывающим параметром.

Если при первой же проверке, параметр цикла не будет удовлетворять условий пункта 3, то тело цикла не выполниться ни разу.

Телом цикла может быть только один оператор. для того, чтобы в теле цикла с параметром выполнить несколько операторов, их необходимо объеденить операторными скобками Begin и End.

После прекращения выполнения оператора, значение параметра цикла не определено, за исключением случаев, когда выход из оператора был осуществлен с помощью GoTo или стандартной процедуры Break.

Пример применения:
Вычислить сумму S=1+1/2+1/3+…+1/50.

```
program prim;
Var i:Integer;
    s:Real;
Begin
s:=0;
For i:=1 to 50 do s:=s+1/I;
writeln(s);
End.
```

7. Алгоритмы вычисления конечных сумм и произведений

Очень часто встречаются задачи, связанные с вычислением суммы конечного числа слагаемых:

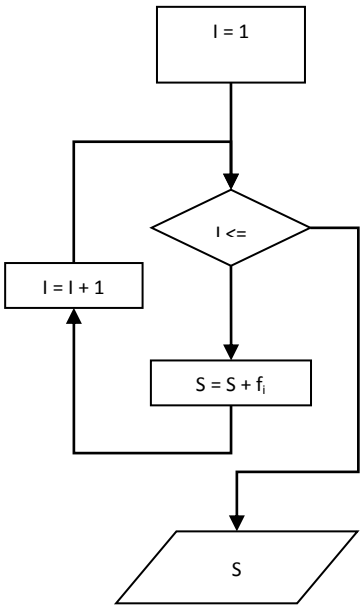
$S = f_1 + f_2 + \ldots + f_n$

Для обозначения этой суммы целесообразно использовать знак суммы $\sum$. Тогда эту сумму можно будет записать в виде:

$$S = \sum_{i=1}^{n} f_i$$

Здесь i - индекс суммирования, который меняется от 1 (начальное значение) до n (конечное значение) с шагом 1, f_i - выражение общего члена. Из выражения f_i при значении i = 1 получится первое слагаемое в сумме, при i = 2 - второе... и, наконец, при i = n последнее слагаемое.

Для указанной суммы вычисления можно организовать в виде циклического алгоритма, когда после каждого шага цикла номер слагаемого увеличивается на единицу, а сумма изменяется на величину i - ого слагаемого.



•**Пример 3.** Составьте программу вычисления суммы.

$$S = 1 + x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \frac{x}{100}^{100}$$

Указание: знаки, начиная со второго слагаемого, в сумме чередуются (+, - , +, - и т.д.). Закон чередования знаков (+, - или -, +) можно описать с помощью дополнительного множителя $(-1)^i$ либо $(-1)^{i+1}$. Первое слагаемое нельзя описать с помощью общей формулы, справедливой для всех членов, поэтому необходимо вынести его за знак суммы.

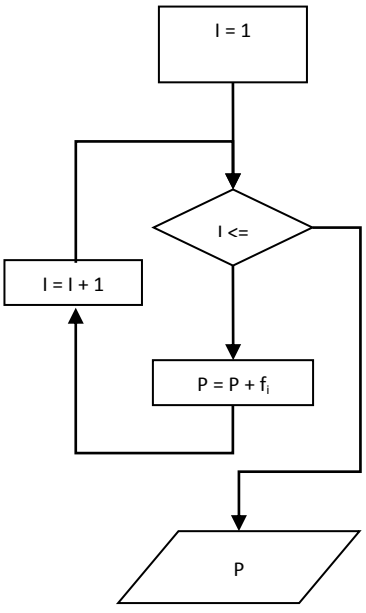
Имеем S = $1+\sum_{i=1}^{100}(-1)^{i+1}\frac{x^i}{i}$

```
REM Вычисление суммы
INPUT “Введите значение x = ”; x
 S = 1
 FOR I =1 TO 100
 S = S + (-1)^(I+1)*X^I/I
 NEXT I
 PRINT “S = ”; S
 END •
```

Для вычисления произведения конечного числа сомножителей воспользуемся символом Π

$$P = f_1 * f_2 * \dots f_3 = \prod_{i=1}^{n} f_i$$

При такой записи целочисленный индекс i меняется от 1 до числа n с шагом 1, где n - количество сомножителей в произведении. При i = 1 получаем первый сомножитель, при i = 2 — второй и т.д., при i = n — n - ый сомножитель. Как и сумму, произведение можно вычислить с помощью



циклического процесса.

• Составьте программу вычисления произведения.

$$P = \frac{(x-2)(x-4)(x-8)...(x-128)}{(x-1)(x-3)(x-7)...(x-127)}$$

Имеем

$$P = \prod_{i=1}^{7} \frac{(x-2^i)}{(x-2^i+1)}$$

```
INPUT  “Введите число x=”; x
P = 1
FOR  I = 1 TO 7
T=x–2^I
P= P*T/(T+1)
NEXT I
PRINT “P= ”; P
END •
```

8. Программирование итерационных циклов

Смотри 4 , 5

9. Вложенные циклы

Если телом цикла является циклическая структура, то такие циклы называют вложенными. Цикл, содержащий в себе другой цикл, называют внешним, а цикл, содержащийся в теле другого цикла, называют внутренним. Внешний и внутренний циклы могут быть трех видов: циклами с предусловием while, циклами с постусловием repeat или циклами с параметром for.

Правила организации внешнего и внутреннего циклов такие же, как и для простого цикла каждого из видов. Но при программировании вложенных циклов необходимо соблюдать следующее дополнительное условие: все операторы внутреннего цикла должны полностью располагаться в теле внешнего цикла.

Рассмотрим пример простой задачи, решение которой предполагает использование вложенных циклов, — задачи вывода на экран таблицы умножения. С использованием цикла for вариант решения данной задачи может быть следующим:

```
program Tab;
var
I,J : byte;
begin
for I:=1 to 10 do {Внешний цикл}
for J:=1 to 10 do {Внутренний цикл}
Writeln (I, ' * ', J, ' = ', I*J); {Тело внутреннего цикла}
end.
```

Проанализируем действие данной программы. В разделе описания переменных описываются переменные I, J целого типа byte, выполняющие функции управляющих переменных циклов for.

Выполнение программы начинается с внешнего цикла. При первом обращении к оператору внешнего цикла for вычисляются значения начального (1) и конечного (10) парамет-ров цикла и управляющей переменной I присваивается начальное значение 1.

Затем циклически выполняется следующее:

1. Проверяется условие 1<= 10.

2. Если оно соблюдается, то выполняется оператор в теле цикла, т. е. выполня-ется внутренний цикл.

2.1. При первом обращении к оператору внутреннего цикла for вычисляются значения начального (1) и конечного (10) параметров цикла и управляющей переменной J присваивается начальное значение 1.

Затем циклически выполняется следующее:

2.2. Проверяется условие J<=10.

2.3. Если оно удовлетворяется, то выполняется оператор в теле цикла, т. е. оператор Writeln (I, ' * ', J, ' = ', I*J), выводящий на экран строку таблицы умножения в соответствии с текущими значениями переменных I и J.

2.4. Затем значение управляющей внутреннего цикла J увеличивается на единицу и оператор внутреннего цикла for проверяет условие J<=10. Если условие соблюдается, то выполняется тело внутреннего цикла при неизменном значении управляющей переменной внешнего цикла до тех пор, пока выполняется условие J<= 10.

Если условие J<=10 не удовлетворяется, т. е. как только J станет больше 10, оператор тела цикла не выполняется, внутренний цикл завершается и управление в программе передается за пределы оператора for внутреннего цикла, т. е. на оператор for внешнего цикла.

3. Значение параметра цикла I увеличивается на единицу, и проверяется условие 1<=10. Если условие 1<=10 не соблюдается, т. е. как только I станет больше 10, оператор тела цикла не выполняется, внешний цикл завершается и управление в программе передается за пределы оператора for внешнего цикла, т. е. на оператор end, и программа завершает работу.

Таким образом, на примере печати таблицы умножения наглядно видно, что при вложении циклов изменение управляющей переменной вложенного цикла проходит полный цикл от начального до конечного значения при неизменном значении управляющей переменной внешнего цикла, затем значение управляющей переменной внешнего цикла изменяется на единицу и опять изменение параметра внутреннего цикла претерпевает изменения от начального до конечного значения с шагом, равным единице. И так до тех пор, пока значение параметра внешнего цикла не станет больше конечного значения, определенного в операторе for внешнего цикла.

10. Одномерные массивы. Типовые задачи.

Самой распространенной структурой, реализованной практически во всех языках программирования, является массив.

Массивы состоят из ограниченного числа компонент, причем все компоненты массива имеют один и тот же тип, называемый базовым. Структура массива всегда однородна. Массив может состоять из элементов типа integer , real или char , либо других однотипных элементов. Из этого, правда, не следует делать вывод, что компоненты массива могут иметь только скалярный тип.

Другая особенность массива состоит в том, что к любой его компоненте можно обращаться произвольным образом. Что это значит? Программа может сразу получить нужный ей элемент по его порядковому номеру (индексу).

Индекс массива

Номер элемента массива называется индексом. Индекс – это значение порядкового типа, определенного, как тип индекса данного массива. Очень часто это целочисленный тип (integer , word или byte), но может быть и логический и символьный.

Описание массива в Паскале. В языке Паскаль тип массива задается с использованием специального слова array (англ. – массив), и его объявление в программе выглядит следующим образом:

Type < имя _ типа >= array [I] of T;

где I – тип индекса массива, T – тип его элементов.

Можно описывать сразу переменные типа массив, т.е. в разделе описания переменных:

Var a,b: array [I] of T;

Обычно тип индекса характеризуется некоторым диапазоном значений любого порядкового типа : I 1 .. I n . Например, индексы могут изменяться в диапазоне 1..20 или ‘ a ’..’ n ’.

При этом длину массива Паскаля характеризует выражение:

ord (I n)- ord (I 1)+1.

Вот, например, объявление двух типов: vector в виде массива Паскаля из 10 целых чисел и stroka в виде массива из 256 символов:

Type

Vector=array [1..10] of integer;

Stroka=array [0..255] of char;

С помощью индекса массива можно обращаться к отдельным элементам любого массива, как к обычной переменной: можно получать значение этого элемента, отдельно присваивать ему значение, использовать его в выражениях.

Опишем переменные типа vector и stroka :

Var a: vector;

c: stroka;

далее в программе мы можем обращаться к отдельным элементам массива a или c . Например, a [5]:=23; c [1]:=' w '; a [7]:= a [5]*2; writeln (c [1], c [3]).

Найти произведение элементов одномерного массива, состоящего из n элементов. Элементы вводятся с клавиатуры.

```
Program proisveden;
Var a: array[1..100] of integer;
    i, n, p: integer;
Begin
 Write ('Сколько элементов? ');  Readln (n);
 p:=1;
 For i:=1 to n do
   begin
     write ('введите число'); readln (a[i]);
     p:=p*a[i];
   end;
   writeln('произведение элементов равно:  ',p);
End.
```

11. Программирование задач обработки двумерных массивов (матриц).

Двумерный массив в Паскале трактуется как одномерный массив, тип элементов которого также является массивом (массив массивов). Положение элементов в двумерных массивах Паскаля описывается двумя индексами. Их можно представить в виде прямоугольной таблицы или матрицы.

Рассмотрим двумерный массив Паскаля размерностью 3*3, то есть в ней будет три строки, а в каждой строке по три элемента:

Каждый элемент имеет свой номер, как у одномерных массивов, но сейчас номер уже состоит из двух чисел – номера строки, в которой находится элемент, и номера столбца. Таким образом, номер элемента определяется пересечением строки и столбца. Например, а 21 – это элемент, стоящий во второй строке и в первом столбце.

Описание двумерного массива Паскаля.

Существует несколько способов объявления двумерного массива Паскаля.

Мы уже умеем описывать одномерные массивы, элементы которых могут иметь любой тип, а, следовательно, и сами элементы могут быть массивами. Рассмотрим следующее описание типов и переменных:

Пример описания двумерного массива Паскаля

```
Type

Vector = array [1..5] of <тип_элементов>;

Matrix= array [1..10] of vector;

Var m: matrix;
```

Мы объявили двумерный массив Паскаля m, состоящий из 10 строк, в каждой из которых 5 столбцов. При этом к каждой i -й строке можно обращаться m [i], а каждому j -му элементу внутри i -й строки – m [i , j].

Определение типов для двумерных массивов Паскаля можно задавать и в одной строке:

```
Type
```

Matrix= array [1..5] of array [1..10] of < тип элементов >;

или еще проще:

type

matrix = array [1..5, 1..10] of <тип элементов>;

Обращение к элементам двумерного массива имеет вид: M [i , j]. Это означает, что мы хотим получить элемент, расположенный в i -й строке и j -м столбце. Тут главное не перепутать строки со столбцами, а то мы можем снова получить обращение к несуществующему элементу. Например, обращение к элементу M [10, 5] имеет правильную форму записи, но может вызвать ошибку в работе программы.

В двумерном массиве, состоящем из n целых чисел, найти сумму элементов в каждой строке. Размер произвольный.

```
Program summastrok;
Var a: array[1..50,1..50] of integer;
    i, j, n, m,S: integer;
Begin
Write('сколько строк?'); Readln(m);
Write('сколько столбцов?'); Readln(n);
For i:=1 to m do
 For j:=1 to n do
     begin
         write('a[',i,',',j,')=');    readln (a[i,j]);
    end;
For i:=1 to m do
 begin
   S:=0;
   For j:=1 to n do
```

```
      S:=S+a[i,j];
    Writeln(‘сумма элементов в ‘,i,’ строке равна  ‘,S);
  end;
End.
```

12. Записи

Запись представляет собой совокупность ограниченного числа логически связанных компонент, принадлежащих к разным типам. Компоненты записи называются полями, каждое из которых определяется именем. Поле записи содержит имя поля, вслед за которым через двоеточие указывается тип этого поля. Поля записи могут относиться к любому типу, допустимому в языке Паскаль, за исключением файлового типа.

Описание записи в языке Паскаль осуществляется с помощью служебного слова record, вслед за которым описываются компоненты записи. Завершается описание записи служебным словом end.

Например, телефонный справочник содержит фамилии и номера телефонов, поэтому отдельную строку в таком справочнике удобно представить в виде следующей записи:

```
type TRec = Record
      FIO: String[20];
      TEL: String[7]
    end;
var rec: TRec;
```

Описание записей возможно и без использования имени типа, например:

```
var rec: Record
        FIO: String[20];
        TEL: String[7]
       end;
```

Обращение к записи в целом допускается только в операторах присваивания, где слева и справа от знака присваивания используются имена записей одинакового типа. Во всех остальных случаях оперируют отдельными полями записей. Чтобы обратиться к отдельной компоненте записи, необходимо задать имя записи и через точку указать имя нужного поля, например:

```
rec.FIO, rec.TEL
```

Такое имя называется составным. Компонентой записи может быть также запись, в таком случае составное имя будет содержать не два, а большее количество имен.

Обращение к компонентам записей можно упростить, если воспользоваться оператором присоединения with.

Он позволяет заменить составные имена, характеризующие каждое поле, просто на имена полей, а имя записи определить в операторе присоединения:

with rec do оператор;

Здесь rec - имя записи, оператор - оператор, простой или составной. Оператор представляет собой область действия оператора присоединения, в пределах которой можно не использовать составные имена. Например для нашего случая:

```
with rec do begin
          FIO:='Иванов А.А.';
          TEL:='2223322';
        end;
```

Такая алгоритмическая конструкция полностью идентична следующей:

```
rec.FIO:='Иванов А.А.';
rec.TEL:='2223322';
```

Инициализация записей может производиться с помощью типизированных констант:

```
type
  RecType = Record
      x,y: Word;
       ch: Char;
      dim: Array[1..3] of Byte
    end;
const
```

```
Rec: RecType = ( x: 127;
                 y: 255;
                 ch: 'A';
                 dim: (2, 4, 8) );
```

Подробнее..

Особой разновидностью записей являются записи с вариантами, которые объявляются с использованием зарезервированного слова case. С помощью записей с вариантами вы можете одновременно сохранять различные структуры данных, которые имеют большую общую часть, одинаковую во все структурах, и некоторые небольшие отличающиеся части.

Например, сконструируем запись, в которой мы будем хранить данные о некоторой геометрической фигуре (отрезок, треугольник, окружность).

```
type
TFigure = record
  type_of_figure: string[10];
  color_of_figure: byte;
  ...
  case integer of
    1: (x1,y1,x2,y2: integer);
    2: (a1,a2,b1,b2,c1,c2: integer);
    3: (x,y: integer; radius: word);
end;
var figure: TFigure;
```

Таким образом, в переменной figure мы можем хранить данные как об отрезке, так и о треугольнике или окружности. Надо лишь в зависимости от типа фигуры обращаться к соответствующим полям записи.

Заметим, что индивидуальные поля для каждого из типов фигур занимают тем не менее одно адресное пространство памяти, а это означает, что одновременное их использование невозможно.

В любой записи может быть только одна вариантная часть. После окончания вариантной части в записи не могут появляться никакие другие поля. Имена полей должны быть уникальными в пределах той записи, где они объявлены.

13. Технология модульного программирования. Процедуры.

Стандартный Паскаль не предусматривает механизмов раздельной компиляции частей программы с последующей их сборкой перед выполнением. Вполне понятно стремление разработчиков коммерческих компиляторов Паскаля включать в язык средства, повышающие его модульность.

Модуль Паскаля – это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы инициирующей части.

Основным принципом модульного программирования является принцип «разделяй и властвуй». Модульное программирование – это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определенным правилам.

Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

В языке Паскаль, как и в большинстве языков программирования, предусмотрены средства, позволяющие оформлять вспомогательный алгоритм как подпрограмму. Это бывает необходимо тогда, когда какой-либо подалгоритм неоднократно повторяется в программе или имеется возможность использовать некоторые фрагменты уже разработанных ранее алгоритмов. Кроме того, подпрограммы применяются для разбиения крупных программ на отдельные смысловые части в соответствии с модульным принципом в программировании.

Для использования подалгоритма в качестве подпрограммы ему необходимо присвоить имя и описать алгоритм по правилам языка Паскаль. В дальнейшем, при необходимости вызвать его в программе, делают вызов подпрограммы упоминанием в нужном месте имени соответствующего подалгоритма со списком входных и выходных данных. Такое упоминание приводит к выполнению входящих в подпрограмму операторов, работающих с указанными данными. После выполнения подпрограммы работа

продолжается с той команды, которая непосредственно следует за вызовом подпрограммы.

В языке Паскаль имеется два вида подпрограмм - процедуры и функции.

Процедуры и функции помещаются в раздел описаний программы. Для обмена информацией между процедурами и функциями и другими блоками программы существует механизм входных и выходных параметров. Входными параметрами называют величины, передающиеся из вызывающего блока в подпрограмму (исходные данные для подпрограммы), а выходными - передающиеся из подрограммы в вызывающий блок (результаты работы подпрограммы).

Одна и та же подпрограмма может вызываться неоднократно, выполняя одни и те же действия с разными наборами входных данных. Параметры, использующиеся при записи текста подпрограммы в разделе описаний, называют формальными, а те, что используются при ее вызове - фактическими.

Описание и вызов процедур и функций

Структура описания процедур и функций до некоторой степени похожа на структуру Паскаль-программы: у них также имеются заголовок, раздел описаний и исполняемая часть. Раздел описаний содержит те же подразделы, что и раздел описаний программы: описания констант, типов, меток, процедур, функций, перменных. Исполняемая часть содержит собственно операторы процедур.

Формат описания процедуры имеет вид:

```
procedure имя процедуры (формальные параметры);
  раздел описаний процедуры
begin
  исполняемая часть процедуры
end;
```

Формат описания функции:

```
function имя функции (формальные параметры):тип результата;
```

раздел описаний функции

begin

исполняемая часть функции

end;

Формальные параметры в заголовке процедур и функций записываются в виде:

var имя праметра: имя типа

и отделяются друг от друга точкой с запятой. Ключевое слово var может отсутствовать (об этом далее). Если параметры однотипны, то их имена можно перечислять через запятую, указывая общее для них имя типа. При описании параметров можно использовать только стандартные имена типов, либо имена типов, определенные с помощью команды type.Список формальных параметров может отсутствовать.

Вызов процедуры производится оператором, имеющим следующий формат:

имя процедуры(список фактических параметров);

Список фактических параметров - это их перечисление через запятую. При вызове фактические параметры как бы подставляются вместо формальных, стоящих на тех же местах в заголовке. Таким образом происходит передача входных параметров, затем выполняются операторы исполняемой части процедуры, после чего происходит возврат в вызывающий блок. Передача выходных параметров происходит непосредственно во время работы исполняемой части.

Вызов функции в Турбо Паскаль может производиться аналогичным способом, кроме того имеется возможность осуществить вызов внутри какого-либо выражения. В частности имя функции может стоять в правой части оператора присваивания, в разделе условий оператора if и т.д.

Для передачи в вызывающий блок выходного значения функции в исполняемой части функции перед возвратом в вызывающий блок необходимо поместить следующую команду:

имя функции := результат;

При вызове процедур и функций необходимо соблюдать следущие правила:

количество фактических параметров должно совпадать с количеством формальных;

соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу.

Заметим, что имена формальных и фактических параметров могут совпадать. Это не приводит к проблемам, так как соответствующие им переменные все равно будут различны из-за того, что хранятся в разных областях памяти. Кроме того, все формальные параметры являются временными переменными - они создаются в момент вызова подпрограммы и уничтожаются в момент выхода из нее.

Рассмотрим использование процедуры на примере программы поиска максимума из двух целых чисел.

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

procedure MaxNumber(a,b: integer; var max: integer);

begin

  if a>b then max:=a else max:=b;

end;

begin

  write('Введите x,y ');

  readln(x,y);

  MaxNumber(x,y,m);

  MaxNumber(2,x+y,n);

  writeln('m=',m,'n=',n);

end.
```

Аналогичную задачу, но уже с использованием функций, можно решить так:

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

function MaxNumber(a,b: integer): integer;

  var max: integer;
```

```
begin
  if a>b then max:=a else max:=b;
  MaxNumber := max;
end;
begin
  write('Введите x,y ');
  readln(x,y);
  m := MaxNumber(x,y);
  n := MaxNumber(2,x+y);
  writeln('m=',m,'n=',n);
end.
```

14. Технология модульного программирования. Функции.

Смотри 13

15. Технология модульного программирования. Модули.

Структура модулей Паскаля

Всякий модуль Паскаля имеет следующую структуру:

Unit <имя_модуля>;

interface <интерфейсная часть>;

implementation < исполняемая часть >;

begin

<инициирующая часть>;

end .

Здесь UNIT – зарезервированное слово (единица); начинает заголовок модуля;

<имя_модуля> - имя модуля (правильный идентификатор);

INTERFACE – зарезервированное слово (интерфейс); начинает интерфейсную часть модуля;

IMPLEMENTATION – зарезервированное слово (выполнение); начинает исполняемую часть модуля;

BEGIN – зарезервированное слово; начинает инициирующую часть модуля; причем конструкция begin <инициирующая часть> необязательна;

END – зарезервированное слово – признак конца модуля.

Таким образом, модуль Паскаля состоит из заголовка и трех составных частей, любая из которых может быть пустой.

Стандартные модули Паскаля

В Турбо Паскале имеется 8 стандартных модулей, в которых содержится множество различных типов, констант, процедур и функций. Этими модулями являются SYSTEM, DOS, CRT, GRAPH, OVERLAY, TURBO3, GRAPh3. Модули Паскаля GRAPH , TURBO 3, GRAPH 3 выделены в отдельные TPU -файлы, а остальные входят в состав библиотечного файла TURBO . TPL . Лишь один модуль Паскаля SYSTEM подключается к любой программе автоматически, все остальные становятся доступны только после указания их имен в списке подключаемых модулей.

Модуль Паскаля SYSTEM. В него входят все процедуры и функции стандартного Паскаля, а также встроенные процедуры и функции, которые не вошли в другие стандартные модули (например, INC , DEC , GETDIR и т.п.). Модуль Паскаля SYSTEM подключается к любой программе независимо от того, объявлен ли он в предложении USES или нет, поэтому его глобальные константы, переменные, процедуры и функции считаются встроенными в Турбо Паскаль.

Модуль Паскаля PRINTER делает доступным вывод текстов на матричный принтер. В нем определяется файловая переменная LST типа TEXT , которая связывается с логическим устройством PRN. После подключения данного модуля Паскаля можно выполнить, например, такое действие:

16. Использование процедур и функций в качестве параметров в подпрограммах.

Во многих задачах, особенно в задачах вычислительной математики, необходимо передавать имена процедур и функций в качестве параметров. Для этого в TURBO PASCAL введен новый тип данных - процедурный или функциональный, в зависимости от того, что описывается.

Описание процедурных и функциональных типов производится в разделе описания типов:

```
type
    FuncType = Function(z: Real): Real;
    ProcType = Procedure (a,b: Real; var x,y: Real);
```

Функциональный и процедурный тип определяется как заголовок процедуры и функции со списком формальных параметров, но без имени. Можно определить функциональный или процедурный тип без параметров, например:

```
type
    Proc = Procedure;
```

После объявления процедурного или функционального типа его можно использовать для описания формальных параметров - имен процедур и функций.

Кроме того, необходимо написать те реальные процедуры или функции, имена которых будут передаваться как фактические параметры. Эти процедуры и функции должны компилироваться в режиме дальней адресации с ключом {$F+}.

Пример. Составить программу для вычисления определенного интеграла

$$\int_{tk}^{2t}$$

$$I = \int_{tn}^{} \frac{S}{\sqrt{1-\sin 2t}}\,dt$$

по методу Симпсона. Вычисление подинтегральной функции реализовать с помощью функции, имя которой передается как параметр. Значение определенного интеграла по формуле Симпсона вычисляется по формуле:

ISimps=2*h/3*(0.5*F(A)+2*F(A+h)+F(A+2*h)+2*F(A+3*h)+...
+2*F(B-h)+0.5*F(B))

где A и B - нижняя и верхняя границы интервала интегрирования,
N - число разбиений интервала интегрирования,
h=(B-A)/N, причем N должно быть четным.

```
Program INTEGRAL;
 type
   Func= function(x: Real): Real;
 var
   I,TN,TK:Real;
   N:Integer;
{$F+}
 Function Q(t: Real): Real;
  begin
    Q:=2*t/Sqrt(1-Sin(2*t));
  end;
{$F-}
 Procedure Simps(F:Func; a,b:Real; N:Integer; var INT:Real);
  var
    sum, h: Real;
```

```
    j:Integer;
  begin
    if Odd(N) then N:=N+1;
    h:=(b-a)/N;
    sum:=0.5*(F(a)+F(b));
    for j:=1 to N-1 do
      sum:=sum+(j mod 2+1)*F(a+j*h);
      INT:=2*h*sum/3
  end;
begin
  WriteLn(' ВВЕДИ TN,TK,N');
  Read(TN,TK,N);
  Simps(Q,TN,TK,N,I);
  WriteLn('I=',I:8:3)
end.
```

17. Множественный тип данных.

Множественный тип данных напоминает перечислимый тип данных. Вместе с тем, множество - набор элементов, не организованных в порядке следования. В математике множество - любая совокупность элементов произвольной природы. Понятие множества в программировании значительно уже математического понятия.

Определение. Под множеством в Паскале понимается конечная совокупность элементов, принадлежащих некоторому базовому типу.

В качестве базовых типов могут использоваться: перечислимые типы данных, символьный и байтовый типы или диапазонные типы на их основе.

Такие ограничения связаны с формой представления множества в языке и могут быть сведены к тому, что функция Ord для используемого базового типа должна быть в пределах от 0 до 255.

Множество имеет зарезервированное слово set of и вводится следующим описаниемType

< имя типа > = set of < имя базового типа >;

Var

< идентификатор,... >:< имя типа >;

Объединением 2-х множеств называется третье множество, которое содержит элементы, которые принадлежат хотя бы одному из множеств операндов, при этом каждый элемент входит в множество только один раз.

Объединение множеств записывается как операция сложения.

```
Type
  Symbol = set of char;
Var
  SmallLatinLetter, CapitalLatinLetter, LatinLetter : Symbol;
Begin
```

. . . . . .

SmallLatinLetter :=['a'..'z'];

CapitalLatinLetter := ['A'..'Z'];

LatinLetter := SmallLatinLetter+CapitalLatinLetter;

. . . . . .

End.

Разностью 2-х множеств является третье множество, которое содержит элементы 1-го множества, не входящие во 2-е множество.a: = a-['d'];

Если в вычитаемом множестве есть элементы, отсутствующие в уменьшаемом, они не влияют на результат.Summer := WarmSeason-Spring-Autumn;

Summer := WarmSeason-May-September;

Пересечением множеств называется множество, содержащее элементы одновременно входящие в оба множества операндов. Операция обозначается знаком умножения.

Summer := WarmSeason*Vacation;

18. Типизированные файлы.

Определение 1.
Файл представляет собой последовательность компонент одного и того же типа. Число компонент не фиксировано. В каждый момент доступна только одна компонента. Говорят, что на эту компоненту установлен указатель файла.

Определение 2.
Если выполнилась операция записи в n-ю компоненту файла, то указатель автоматически продвигается к (n+1)-й компоненте, то есть для записи становится доступной уже только (n+1)-я компонента.

Определение 3.
Длиной файла называется число записанных компонент. Файл, не содержащий компонент, называется пустым, его длина равна нулю. Читать файл можно также только последовательно по одной компоненте.

Для того, чтобы определять готовность файла к чтению либо к записи информации , существует стандартная функция EOF(F), где F-имя файла. Если указатель файла продвинулся за конец файла (готовность к записи), то эта функция принимает значение TRUE, во всех остальных случаях значение FALSE.
Общий вид описания типа FILE:
TYPE R = FILE OF TC;
Здесь R - идентификатор типа, TC - тип компонент (может быть любым, кроме типа FILE).

Файлы могут быть разных типов: состоять из целых компонент, либо вещественных, либо записей и т.д. Как и другие переменные, каждую переменную-файл надо описать в разделе VAR. Вводя имя переменной файла (имя файла), надо указать, какого типа файл. Этот тип должен быть обозначен каким-либо именем и описан в разделе TYPE.
Например, файл F вещественных чисел:
TYPE N = FILE OF REAL;
VAR F:N;(*описание переменной файла*)
Файл может быть описан и непосредственно при описании переменной, например:
VAR F:FILE OF REAL;

В первом случае введено имя файла F и соответствующий тип обозначен N; во втором введено имя файла F, а его тип имени не имеет и поэтому в разделе TYPE не описывается.

Возможно описание переменной без указания типа компонент:
VAR F:FILE;

Замечение редактора: в этом случае файл называется нетипизированным. Для работы с такими файлами используются отличные от указанных ниже процедуры и функции.

Для того, чтобы ассоциировать переменную F с именем файла, с которым будем работать, в паскале используется функция Assign.
Общий вид функции:
Assign (F,'My_file.alr');
Здесь F - имя переменной типа FILE с которой теперь связан файл My_file.alr. Далее в программе, когда мы будем обращаться к переменной F, например записывать данные , то мы будем работать с файлом My_file.alr. Вместе с файлом может указываться и путь.
Пример.
Assign (F,'C:\Tp7\Bin\My_file.alr');
1. Открытие, чтение и запись в файл.
Процедура RESET.
RESET открывает только уже существующие файлы на чтение и запись.
Общий вид:
RESET (F); (* Считается, что файл My_file.alr существует *)
Пример.

```
TYPE NF:FILE OF INTEGER;
VAR F:NF:STRING;
TT:integer;
BEGIN
        ASSIGN(F,'TITOV.PRS');
        RESET(F);
        WHILE NOT EOF(F) DO
        BEGIN
                READ(F,TT);
                WRITE(TT);
        END;
        CLOSE(F);
```

END.

2. Процедура REWRITE.

REWRITE открывает существующий файл на перезапись т.е. с потерей всей предыдущей хранящейся в нем информации, или если файла не существует, то создает новый файл с тем именем, с которым ассоциирована файловая переменная.

Общий вид:

REWRITE(F);

Пример. Надо создать файл для записи массива целых чисел.

```
VAR F:FILE OF INTEGER;I:INTEGER;
BEGIN
    ASSIGN(F,'D:\TP7\MYMASS.MAS');
    REWRITE(F);
    FOR I:=1 TO 100 WRITELN(F,SQR(I));
    CLOSE(F);
END.
```

Ввод и вывод осуществляется с помощью процедур READ и WRITE

Общий вид: READ(F,P1,P2,P3,:,PN);

P1,P2,P3,:,PN - Переменные в которые считываются значения из файла.

WRITE(F,T1,T2,T3,:,TN);

T1,T2,T3,:,TN - Переменные, значения которых запишутся в файл.

3. После использования процедур RESET и REWRITE файл НЕОБХОДИМО закрыть. Это делается с помощью процедуры CLOSE.

Общий вид:

CLOSE(F);

Пример.

PROGRAM T20;

```
VAR F:FILE OF INTEGER;U: STRING;
BEGIN
        ASSIGN(F,'C:\AUTOEXEC.BAT');
        RESET(F);
    WRITELN ('Содержание файла Autoexec.bat')
    WHILE  NOT EOF(F) DO
        BEGIN
            READLN(F,U);
            WRITELN(U);
        END;
    WRITE('****Конец файла****');
    CLOSE(F);
END.
```

19. Текстовые файлы.

Особое место в языке ПАСКАЛЬ занимают текстовые файлы, компоненты которых имеют символьный тип. Для описания текстовых файлов в языке определен стандартный тип Text:

```
var   TF1, TF2: Text;
```

Текстовые файлы представляют собой последовательность строк, а строки - последовательность символов. Строки имеют переменную длину, каждая строка завершается признаком конца строки.

С признаком конца строки связана функция EOLn(var T:Text):Boolean,

где T - имя текстового файла. Эта функция принимает значение TRUE, если достигнут конец строки, и значение FALSE, если конец строки не достигнут. Для операций над текстовыми файлами, кроме перечисленных, опредtлены также операторы обращения к процедурам:

ReadLn(T) - пропускает строку до начала следующей;

WriteLn(T) - завершает строку файла, в которую производится запись, признаком конца строки и переходит к началу следующей. Для работы с текстовыми файлами введена расширенная форма операторов ввода и вывода. Оператор
Read(T,X1,X2,...XK) эквивалентен группе операторов

```
begin
   Read(T,X1);
   Read(T,X2);
   .....
   Read(T,XK)
end;
```

Здесь T - текстовый файл, а переменные X1, X2,...XK могут быть либо переменными целого, действительного или символьного типа, либо строкой.

При чтении значений переменных из файла они преобразуются из текстового представления в машинное.

Оператор Write(T,X1,X2,...XK) эквивалентен группе операторов

```
begin

  Write(T,X1);

  Write(T,X2);

  .....

  Write(T,XK)

end;
```

Здесь T - также текстовый файл, но переменные X1,X2,...XK могут быть целого, действительного, символьного, логического типа или строкой. При записи значений переменных в файл они преобразуются из внутреннего представления в текстовый.

К текстовым файлам относятся стандартные файлы INPUT, OUTPUT.

Рассмотренные ранее операторы ввода - вывода являются частным случаем операторов обмена с текстовыми файлами, когда используются стандартные файлы ввода - вывода INPUT, OUTPUT.

Работа с этими файлами имеет особенности:

-имена этих файлов в списках ввода - вывода не указываются;

-применение процедур Reset, Rewrite и Close к стандартным файлам ввода - вывода запрещено;

-для работы с файлами INPUT, OUTPUT введена разновидность функции EOLn без параметров.

TURBO PASCAL вводит дополнительные процедуры и функции, применимые только к текстовым файлам, это SetTextBuf, Append, Flush, SeekEOLn, SeekEOF.

Процедура SetTextBuf(var f: Text; var Buf; BufSize: Word) служит для увеличения или уменьшения буфера ввода - вывода текстового файла f. Значение размера буфера для текстовых файлов по умолчанию равно 128 байтам. Увеличение размера буфера сокращает количество обращений к диску. Рекомендуется изменять разиер буфера до открытия файла. Буфер файла начнется с первого байта переменной Buf. Размер буфера задается в необязательном параметре BufSize, а если этот параметр отсутствует, размер буфера определяется длиной переменной Buf.

Процедура Append(var f: Text) служит для специального открытия выходных файлов. Она применима к уже существующим физическим файлам и открывает из для дозаписи в конец файла.

Процедура Flush(var f: Text) применяется к открытым выходным файлам. Она принудительно записывает данные из буфера в файл независимо от степени его заполнения.

Функция SeekEOLn(var f: Text): Boolean возвращает значение True, если до конца строки остались только пробелы. Функция SeekEOF(var f: Text): Boolean возвращает значение True, если до конца файла остались строки, заполненные пробелами.

20. Файлы прямого доступа.

TURBO PASCAL позволяет применять к компонентным и бестиповым файлам, записанным на диск, способ прямого доступа. Прямой доступ означает возможность заранее определить в файле блок, к которому будет применена операция ввода - вывода. В случае бестиповых файлов блок равен размеру буфера, для компонентных файлов блок - это одна компонента файла.

Прямой доступ предполагает, что файл представляет собой линейную последовательность блоков. Если файл содержит n блоков, то они нумеруются от 1 через 1 до n. Кроме того, вводится понятие условной границы между блоками, при этом условная граница с номером 0 расположена перед блоком с номером 1, граница с номером 1 расположена перед блоком с номером 2 и, наконец, условная граница с номером n находится после блока с номером n.

Реализация прямого доступа осуществляется с помощью функций и процедур FileSize, FilePos, Seek и Truncate.

Функция FileSize(var f): Longint возвращает количество блоков в

открытом файле f.

Функция FilePos(var f): Longint возвращает текущую позицию в файле f. Позиция в файле - это номер условной границы. Для только что открытого файла текущей позицией будет граница с номером 0. Это значит, что можно записать или прочесть блок с номером 1. После чтения или записи первого блока текущая позиция переместится на границу с номером 1, и можно будет обращаться к ьлоку с номером 2. После прочтения последней записи значение FilePos равно значению FileSize.

Процедура Seek(var f; N: Longint) обеспечивает назначение текущей позиции в файле (позиционирование). В параметре N должен быть задан номер условной границы, предшествующей блоку, к которому будет производиться последующее обращение. Например, чтобы работать с блоком 4, необходимо задать значение N, равное 3. Процедура Seek работает с открытыми файлами.

Процедура Truncate(var f) устанавливает в текущей позиции признак конца файла и удаляет (стирает) все последующие блоки.

Пример. Пусть на НМД имеется текстовый файл ID.DAT, который содержит числовые значения действительного типа по два числа в каждой строке - значения аргумента и функции соответственно. Количество пар чисел не более 200. Составить программу, которая читает файл, значения аргумента и функции записывает в одномерные массивы, подсчитывает их количество, выводит на экран дисплея и записывает в файл компонентного типа RD.DAT.

```
Program F;
 var
  rArg, rF: Array[1..200] of Real;
    inf: Text;
    outf: File of Real;
    n, l: Integer;
 begin
   Assign(inf,'ID.DAT');
   Assign(outf,'RD.DAT');
   Reset(inf);
   Rewrite(outf);
   n:=0;
   while not EOF(inf) do
     begin
       n:=n+1;
       ReadLn(inf,rArg[n],rF[n])
     end;
   for l:=1 to n do
    begin
     WriteLn(l:2,rArg[l]:8:2,rF[l]:8:2);
     Write(outf,rArg[l], rF[l]);
    end;
   close(outf)
 end.
```