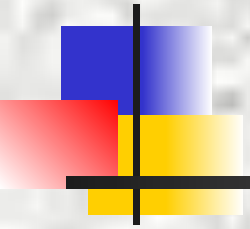
Основы языка *TURBO PASCAL*



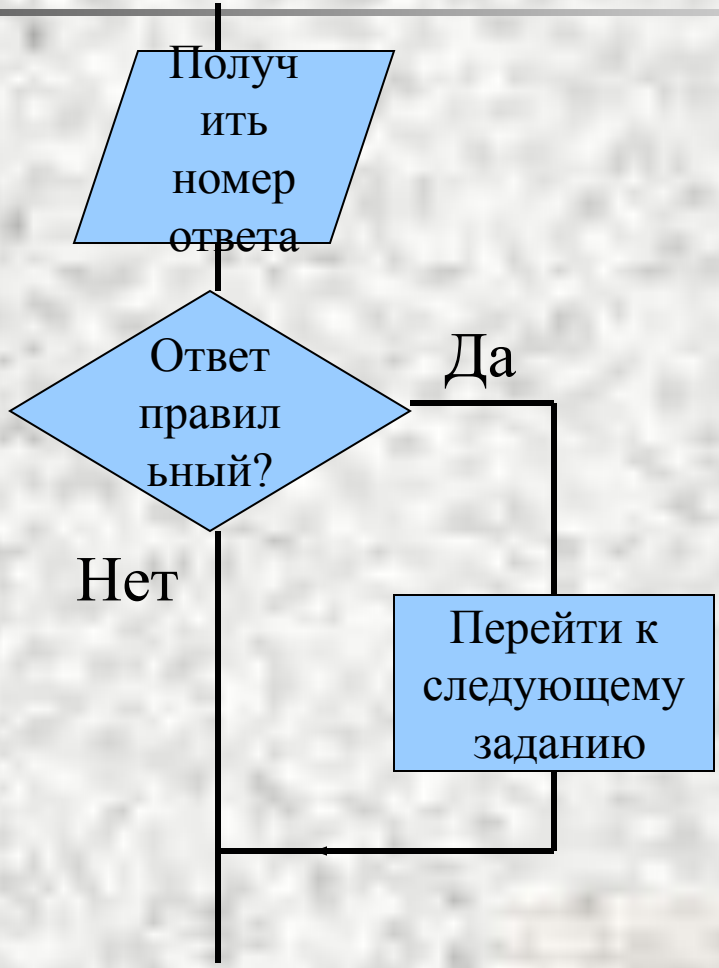
Содержание:

Алгоритм

Алгоритмы решения большинства задач не являются последовательными.действия (вычисления), которые необходимо выполнить, могут зависеть от определённого условия, например входных данных, или результатов, полученных во время выполнения программы. Например, в программе проверки знаний оценка за выбранный из нескольких вариантов ответ, добавляемая к общей сумме баллов, зависит от того, является ли ответ правильным. Фрагмент алгоритма решения этой задачи представлен на блок-схеме.

Назад
Далее
БЛОК-СХЕМА

Пример фрагмента блок-схемы ветвящегося алгоритма



Назад

Условие

В языке Pascal условие - это выражение логического типа (BOOLEAN), которое может принимать одно из двух значений: “истина” (TRUE) или ” ложь” (FALSE).

В Pascal имеется шесть операторов, позволяющих сравнивать между собой значения числовых переменных, а также значение переменной и константу (число).

Оператор	Описание	Значение выражения
>	Больше	TRUE, если первый операнд больше второго, иначе FALSE
<	Меньше	TRUE, если первый операнд меньше второго, иначе FALSE
=	Равно	TRUE, если первый операнд равен второму, иначе FALSE
< >	Не равно	TRUE, если первый операнд не равен второму, иначе FALSE
< =	Больше или равно	TRUE, если первый операнд больше или равен второму, иначе FALSE
< =	Меньше или равно	TRUE, если первый операнд меньше или равен второму, иначе FALSE

Далее
Назад
Меню

Использование операторов сравнения позволяет записывать простые условия. Из простых условий, которые являются выражениями логического типа, можно строить сложные условия с применением к ним, как к операндам, логических операторов: **NOT** - отрицание, **AND** - “логическое И”, **OR** - “логическое ИЛИ”.

При записи сложных условий важно учитывать то, что логические операторы имеют более высокий приоритет, чем операторы сравнения, поэтому простые условия следует брать в скобки.

Например:

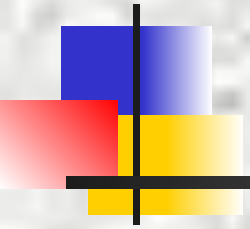
((a = b) AND (c = d)) OR (a > d)

Далее
Назад
Меню

ВЕТВЛЕНИЯ

Выбор действия в зависимости от выполнения условия может быть реализован при помощи инструкций **IF** и **CASE.**

Инструкция IF



Синтаксис инструкции IF:

```
if  условие
        then
          begin
{инструкции, выполняемые, если условие истинно}
          end
        else
          begin
{инструкции, выполняемые, если условие ложно}
          end;
```

Далее

Назад

Меню

Инструкция IF выполняется следующим образом:

1. Вычисляется значение условия (выражения логического типа).

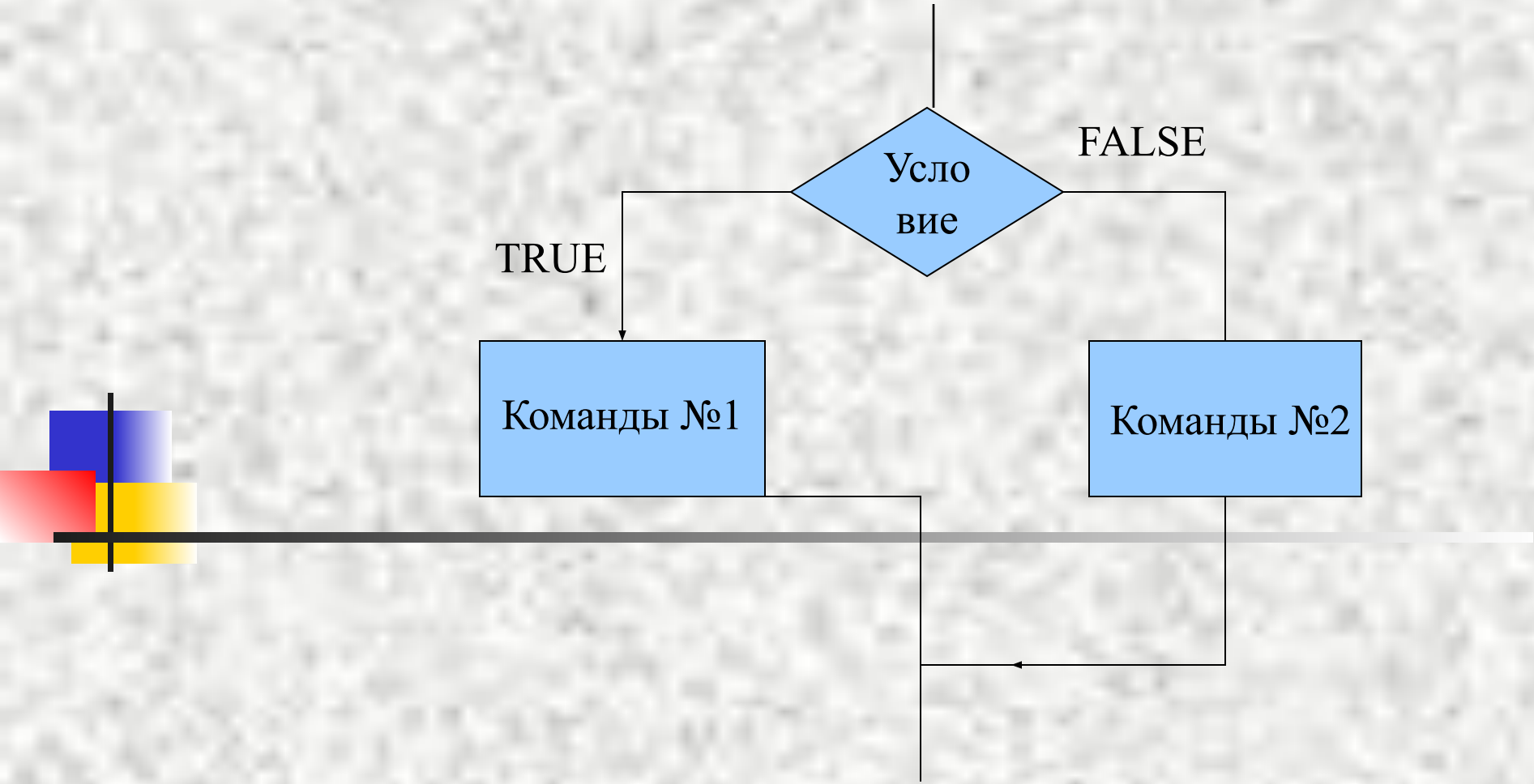
2. Если значение выражения условия равно **TRUE**, то выполняются инструкции следующие за словом **THEN.**

Если значение выражения условия равно **FALSE**, то выполняются инструкции, следующие за словом **ELSE.**

Примечание:

*Если при выполнении (невыполнении) условия надо выполнить только одну инструкцию, то слова **BEGIN** и **END (ELSE)** могут быть опущены.*

Назад
Далее
Меню
Блок-схема
Пример



Блок-схема составного оператора

IF - THEN - ELSE

Назад
Меню

Пример:

Если переменная t содержит обозначение типа соединения сопротивлений электрической цепи (t = 1 соответствует последовательному соединению, t = 2 - параллельному), а переменные r1 и r2 содержат величины сопротивлений, то инструкция

```
if  t = 1
then
z:=r1 + r2
else
z:=(r1 + r2)/(r1*r2);
```

вычисляет сопротивление цепи в зависимости от типа соединения сопротивлений.

Назад

Меню

Если какое-либо действие нужно выполнить только при выполнении условия, инструкция **IF** может быть записана так:

if *{условие}*

then

begin

{инструкции, выполняемые, если условие истинно}

end

Например, инструкция

```
if  n = m
then c:= c+ 1;
```

увеличивает значение переменной c только в том случае, если значения переменных n и m равны.

Далее
Назад
Меню

Часто в программе необходимо реализовать выбор более чем из двух вариантов. Множественный выбор можно реализовать при помощи двух инструкций **IF**, одна из которых «вложена» в другую.

```
if   {условие}
then
begin
{инструкции, выполняемые, если условие истинно}
end
else
if   {условие}
then  begin
{инструкции, выполняемые, если условие истинно}
end
else  begin
{инструкции, выполняемые, если условие ложно}
end;
end.
```

Обратите **внимание**, что после инструкций, расположенных перед else, символ «точка с запятой» не поставлен.

Далее
Назад
Меню

Инструкция CASE

Инструкция **CASE** позволяет реализовать множественный выбор и в общем виде записывается так:

```
case выражение of
список констант 1: begin
    {последовательность инструкций 1}
    end;
список констант 2: begin
    {последовательность инструкций 2}
    end;
список констант N: begin
    {последовательность инструкций N}
    end;
  else  begin
  последовательность инструкций, выполняемая в}
{случае, если выражение не попало ни в один из}
    {списков констант}
    end;
end;
```

Блок-схема

Далее

Назад

Меню

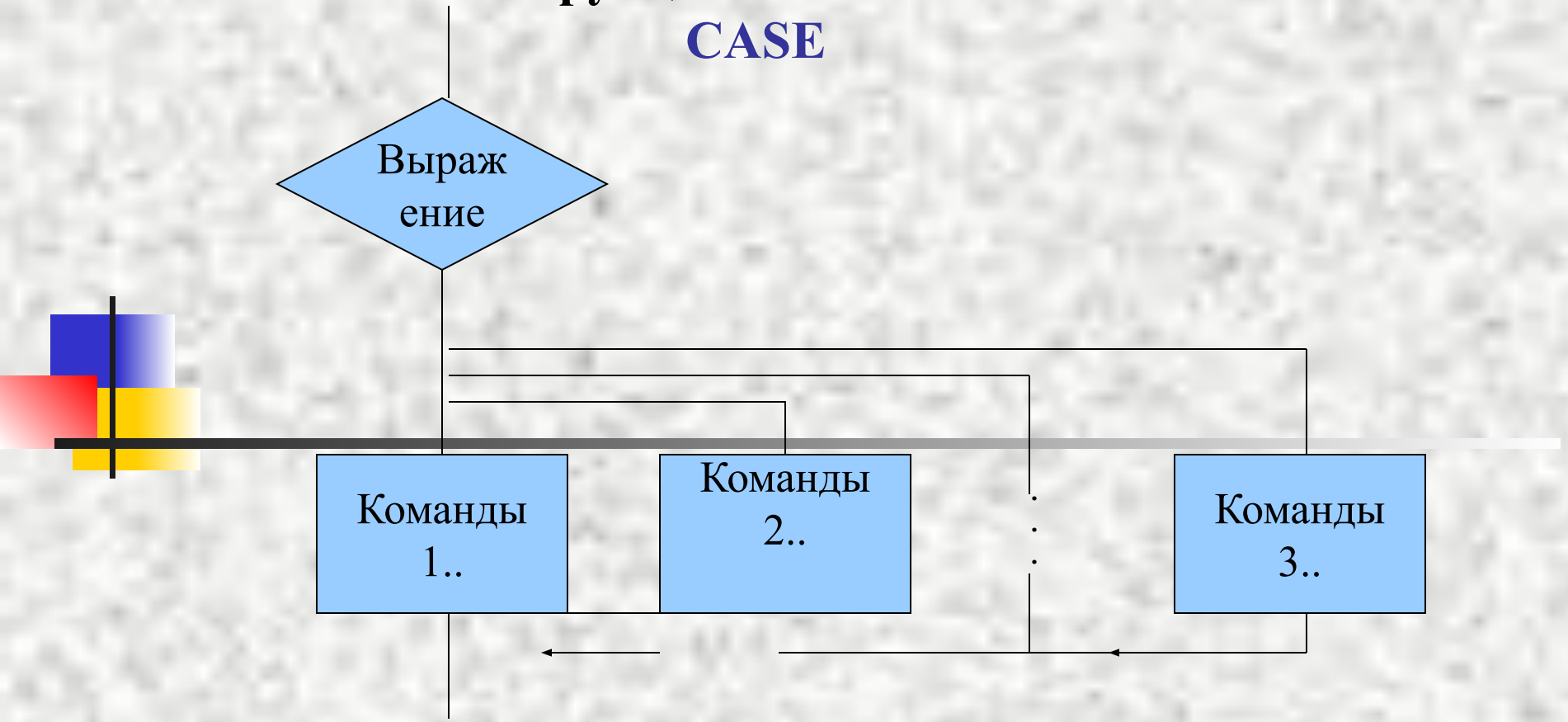
Пример

Выражение – блок инструкций, от значения которого зависит дальнейший ход программы (одна из последовательностей инструкций, которая должна быть выполнена).

Список констант - константы, разделенные запятыми. Если константы представляют диапазон чисел, то вместо списка можно указать первую и последнюю константу диапазона, разделив их двумя точками. Например, список 1, 2, 3, 4, 5, 6 может быть заменен диапазоном 1..6.

Назад

Блок-схема алгоритма, соответствующего инструкции CASE

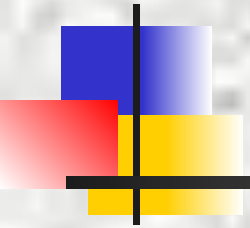


Назад

Меню

Пример:

```
1. case day of
1, 2, 3, 4, 5: write (‘Рабочий день.’);
6: write (‘Суббота!’);
7: write (‘Воскресенье!’);
end;
2. case day of
1..5: write (‘Рабочий день.’);
6: write (‘Суббота!’);
7: write (‘Воскресенье!’);
end;
3. case day of
6: write (‘Суббота!’);
7: write (‘Воскресенье!’);
else write (‘Рабочий день.’);
end;
```



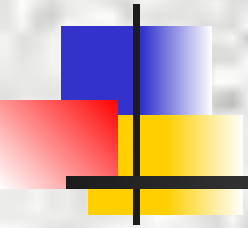
Назад

При выполнении инструкции CASE происходит следующее: вычисляется выражение оператора CASE, затем полученное значение последовательно сравнивается с константами из списков констант. Если значение выражения совпадает с константой из списка, то выполняется соответствующая этому списку последовательность инструкций, и на этом выполнение инструкции CASE завершается. Если значение выражения не совпадает ни с одной константой из всех списков, то выполняется последовательность инструкций идущих после ELSE. Синтаксис инструкции CASE позволяет использовать ELSE и соответствующую последовательность инструкций. В этом случае, если значение выражения не совпадает ни с одной из всех списков, то выполняется следующая за CASE инструкция.

Далее
Назад
Меню
Пример

Пример: Следующий фрагмент программы показывает использование инструкции CASE для организации меню. Программа выводит меню(список вариантов) и ожидает ввода номера задачи, которая должна быть выполнена.

```
var
  vybor: integer;
begin
writeln (‘1 - Максимальное число’);
writeln (‘2 - Минимальное число’);
writeln;
writeln (‘Введите номер задачи и нажмите <Enter>’);
readln (vybor);
case vybor of
1: begin
{вычисление максимального числа}
end;
2: begin
{вычисление минимального числа}
end;
end;
end.
```



Назад

Меню

ЦИКЛЫ

При решении многих задач некоторую последовательность действий приходится выполнять несколько раз. Например программа контроля знаний выводит вопрос, принимает ответ, добавляет оценку за ответ к сумме баллов, затем повторяет это действие ещё раз, и ещё до тех пор, пока не будут выведены все вопросы.

Такие повторяющиеся действия называются циклами и реализуются в программе с использованием инструкций циклов.

В языке Pascal циклические вычисления реализуются при помощи инструкций FOR, WHILE и REPEAT.

Далее
Назад
Меню

Инструкция FOR

Инструкция FOR используется, если надо выполнить некоторую последовательность действий несколько раз, причём число повторений заранее известно. Например вычислить значения функции в нескольких различных, отстоящих на равном расстоянии друг от друга точках, то есть построить таблицу значений функции. такие задачи решаются с использованием цикла с фиксированным числом повторений, который в языке Pascal реализуется при помощи инструкции FOR.

В общем виде инструкция выглядит так:

for счётчик_цикла:= начальное_значение_счётчика

to конечное_значение_счётчика do

begin

{последовательность операторов}

end

Примечание:

Если между begin и end находится только одна инструкция, то begin и end можно не писать.

Далее

Назад

Меню

Пример

Счетчик_цикла - имя переменной-счетчика циклов;

Начальное_значение_счетчика - выражение, определяющее начальной значение переменной-счетчика циклов;

Конечное_значение_счетчика -выражение, определяющее конечное значение переменной-счетчика циклов;

Назад

Пример:

```
1. for i:= 1 to 10 do
        begin
        write (‘-’);
        end;

2. for j:= i+1 to n do
        begin
        y:= 2*x*x - 10;
        x:= x + 0.5;
        end;
```

Назад
Меню

Обычно в качестве выражений, определяющих значения начального и конечного состояния счётчика циклов используют переменные или константы. В этом случае последовательность операторов, находящаяся между *begin* и *end*, будет выполнена (начальное_ значение_ счётчика - конечное_значение_счётчика + 1) раз.

Значение переменной_счётчика можно использовать в последовательности операторов между *begin* и *end*.

Например, в результате выполнения инструкции

```
for i:= 1 to 5 do
begin
writeln (i);
end;
```

на экран будут выведены числа 1, 2, 3, 4 и 5 - каждое число на отдельной строке.

Если в инструкции FOR вместо слова *to* записать *downto*, то после очередного цикла значение счётчика будет не увеличиваться, а уменьшаться.

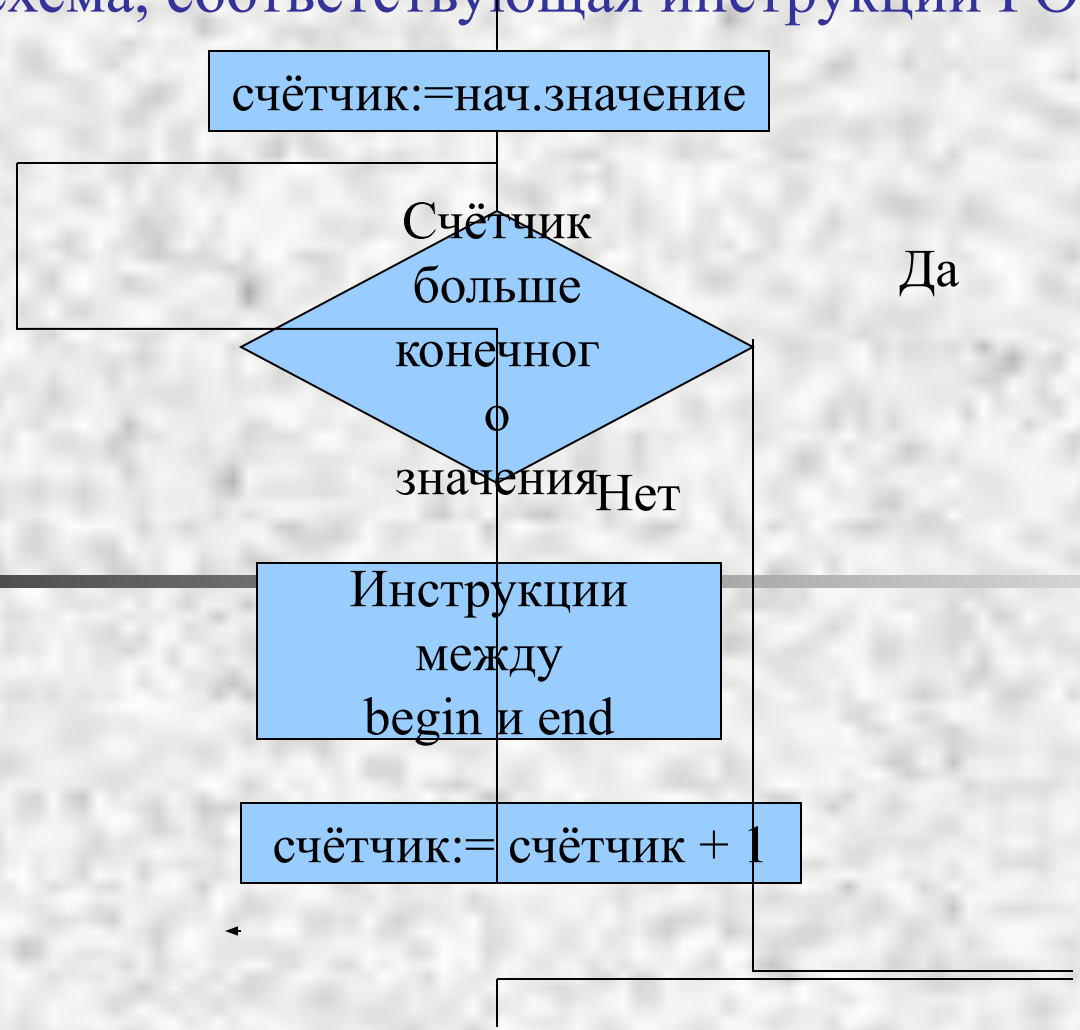
Например, инструкция

```
for j:= 10 downto 0 do
writeln (i);
```

выводит на экран числа от 10 до 0.

Далее
Назад
Меню
Блок-схема

Блок-схема, соответствующая инструкции FOR



Назад

Обратите внимание, что в случае, если начальное значение счётчика превышает конечное значение счётчика, то последовательность операторов между *begin* и *end* ни разу не будет выполнена.

Инструкция WHILE

Инструкция WHILE используется в программе, если надо провести некоторые повторные вычисления (цикл), однако число повторов заранее неизвестно и определяется самим ходом вычисления. Типичными примерами использования цикла WHILE являются вычисления с заданной точностью, поиск в массиве или в файле.

В общем виде инструкция выглядит:

Блок-схема

Пример

```
while условие do
begin
{последовательность  инструкций}
end;
```

где *условие* - выражение логического типа.

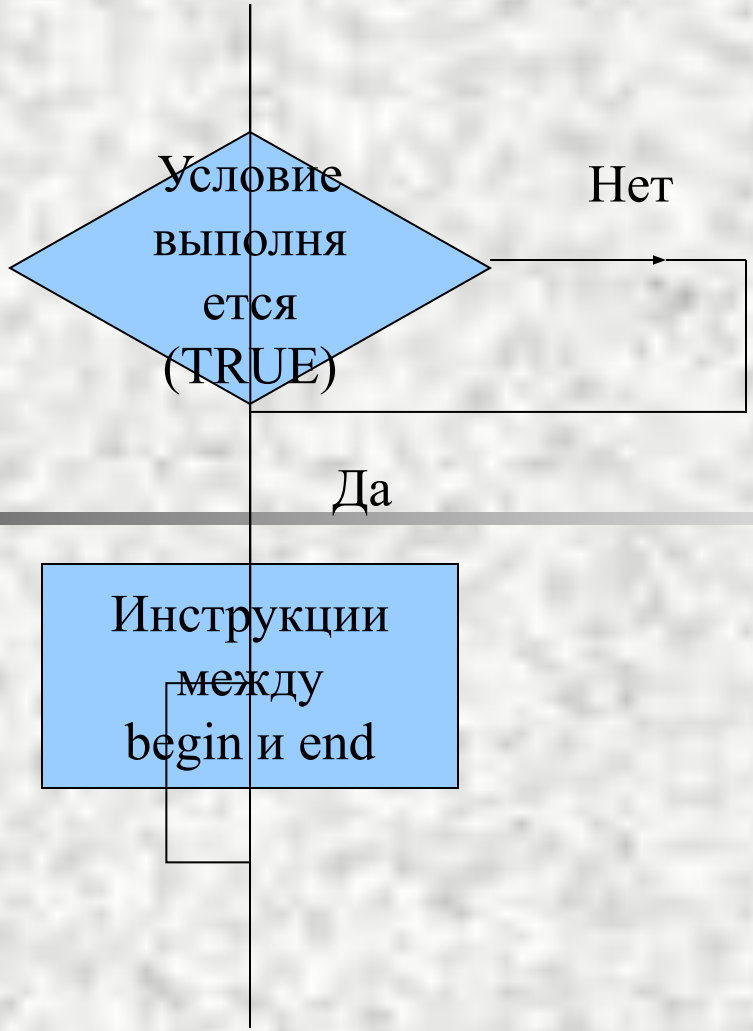
Инструкции, находящиеся между *begin* и *end*, выполняются до тех пор, пока *условие* истинно (значение выражения условие равно TRUE).

Обратите внимание:

- для того, чтобы последовательность инструкций между *begin* и *end* была выполнена хотя бы один раз, необходимо, чтобы перед выполнением инструкции WHILE условие было истинно;
- для того, чтобы цикл завершился, необходимо, чтобы последовательность инструкций между *begin* и *end* изменяла значения переменных, входящих в выражение *условие*.

Далее
Назад
Меню

Блок-схема, соответствующая инструкции WHILE



Назад

Меню

Пример: С клавиатуры вводится непустая последовательность различных натуральных чисел, завершающаяся 0. Определить порядковый номер наименьшего из них.

Обозначим через x очередное число, через i - его номер, через min - минимальное из введённых чисел и через k - его номер.

```
program nomer (input, output);
var x, i, min, k: integer;
begin
write (‘Введите произвольные два числа:’); readln (min, x);
k:= 1; i:= 2;
while x < > 0 do
begin
if  x < min then
begin min:= x; k:= 1 end;
write (‘Введите очередное число:’);
readln (x); i:= i + 1;
end;  {конец цикла while}
write (‘Номер минимального числа’, k);
end.
```

Назад

Меню

Инструкция REPEAT

Инструкция REPEAT, как и инструкция WHILE, используется в программе, если надо провести некоторые повторяющиеся вычисления (цикл), однако число повторов заранее не известно и определяется самим ходом вычисления.

В общем виде инструкция выглядит так:

Блок-схема

Пример

```
repeat
{последовательность инструкций}
until условие
```

где условие - выражение логического типа.

Инструкция выполняется следующим образом:

1. Выполняются инструкции, следующие за словом *repeat.*
2. Вычисляется значение условия. Если условие ложно (значение выражения условие равно FALSE), то повторно выполняются инструкции цикла. Если условие истинно (значение выражения условие равно TRUE), то выполнение цикла прекращается.

Таким образом, инструкции, находящиеся между *repeat* и *until*, выполняются до тех пор, пока условие ложно (значение выражения условие равно FALSE).

Далее
Назад
Меню

Пример:

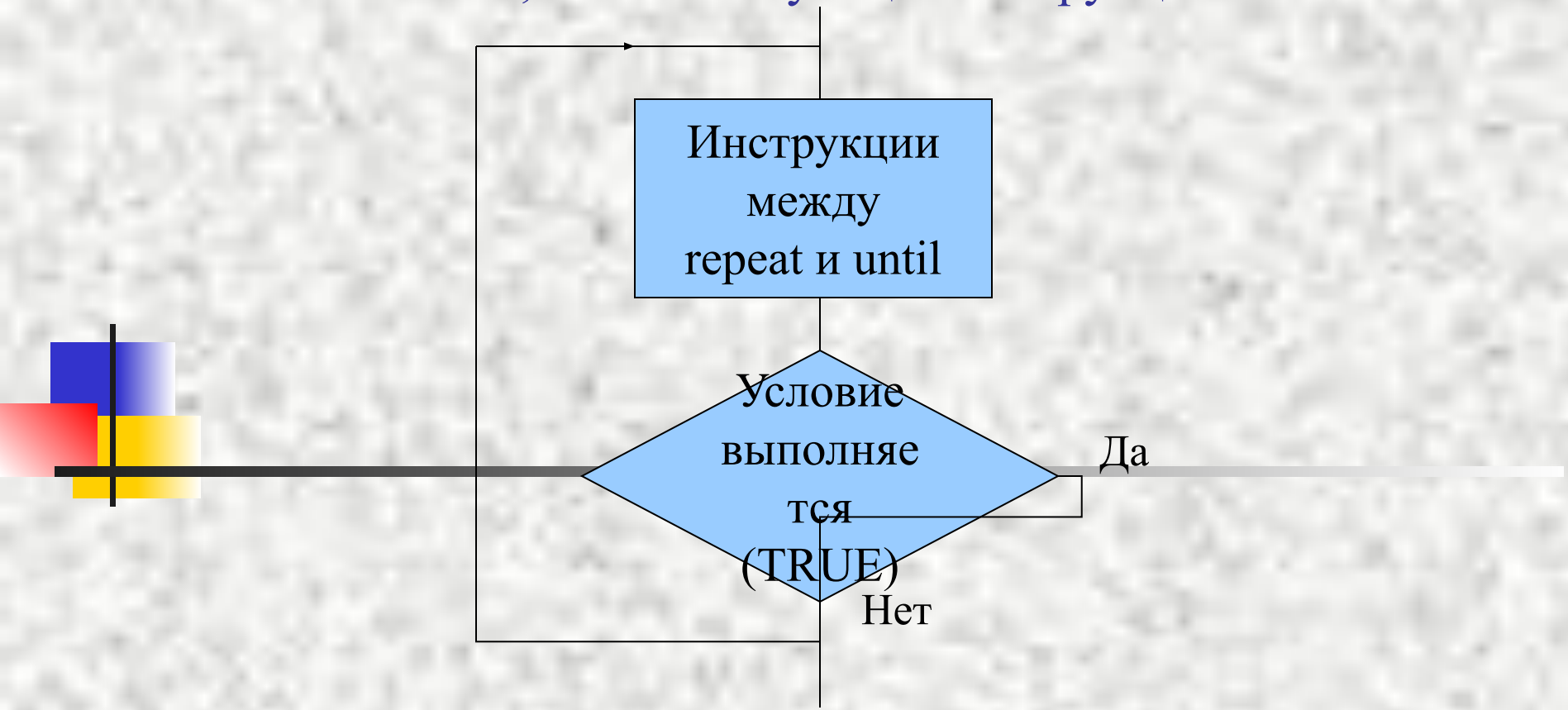
```
1. repeat
   writeln (i);
   i:= i + 1;
until i = 10;
```

```
2. repeat
   read (n);
if n < > then summ:= summ + n;
until n = 0;
```

Назад

Меню

Блок-схема, соответствующая инструкции REPEAT



Обратите внимание:

- последовательность инструкций между *repeat* и *until* всегда будет выполнена хотя бы один раз;
- для того, чтобы цикл завершился, необходимо, чтобы последовательность операторов между *repeat* и *until* изменяла значения переменных, входящих в выражение условие.

Назад
Меню

Инструкция полезна при разработке программ, обрабатывающих ввод с клавиатуры или из файла.

Пример программы, вычисляющий сумму положительных чисел, вводимых с клавиатуры:

```
var
  numb: integer; число, введённое с клавиатуры
  summ: integer; сумма введённых чисел
begin
  writeln (‘Вычисление суммы положительных целых чисел.’);
  summ:= 0;
  repeat
    write (‘-->’);
    readln (numb);
    if numb > 0 then summ:= summ + numb;
  until numb <= 0;
  writeln (‘Сумма введённых чисел’, summ);
end.
```

Программа заканчивает работу, как только вводиться неположительное число.

Назад
Далее
Меню

МАССИВЫ

Массив - это структура данных, которую можно рассматривать как набор переменных одинакового типа, имеющих общее имя. Массивы удобно использовать для хранения однородной по составу информации, например, элементов таблиц, коэффициентов уравнений, матриц.

Объявление массива

Перед использованием массив, как и любая переменная, должен быть объявлен в разделе объявления переменных. В общем виде объявление массива выглядит так:

Имя: array [нижний_индекс .. верхний_индекс] of тип

где

- Имя - имя переменной-массива;
- array - ключевое слово, обозначающее, что переменная является массивом;
- нижний_индекс и верхний_индекс - целые числа, определяющие диапазон изменения индексов (номеров) элементов массива и, неявно, количество элементов (размер) массива;
- тип - тип элементов массива.

Далее
Назад
Меню

Примеры объявления массивов:

```
temper: array [1..31] of real;
koef: array [0..2] of integer;
name: array [1..30] of string [25];
```

При объявлении массива удобно использовать именованные константы. Именованная константа объявляется в разделе описания констант, который располагают перед разделом объявления переменных. Начинается раздел объявления констант словом **CONST**. Например, массив названий команд участниц чемпионата по футболу можно объявить так:

```
const
  NT=18;  {число команд}
SN=25;  {предельная длина названия команды}
var
  team: array [1..NT] of string [SN];
```

Чтобы в программе использовать элемент массива, надо указать имя массива и номер элемента (индекс), заключив его в квадратные скобки. Индекс может быть константой или выражением целого типа. Например:

```
team [1]:= Зенит;
d:= koef [1] * koef [1] - 4 * koef [2] * koef [1];
readln (name [n + 1]);
writeln (temper [i]);
```

Далее

Назад

Меню

К *типичным действиям с массивами* можно отнести следующие:

- вывод массива;
- ввод массива;
- сортировка массива;
- поиск в массиве заданного элемента;
- поиск в массиве максимального или минимального элемента;

Вывод массива - это вывод на экран значений элементов массива. Если в программе необходимо вывести значения всех элементов массива, то для этого удобно использовать инструкцию FOR, переменная-счётчик которой может быть реализована как индекс элемента массива.

Например, программа, выводящая на печать номера и названия дней недели, хранящиеся в массиве day, может быть реализована так:

```
var
  day: array [1..7] of string [11];
  i: integer;
begin
day [1]:='Понедельник';
day [2]:='Вторник';
day [3]:='Среда';
day [4]:='Четверг';
day [5]:='Пятница';
day [6]:='Суббота';
day [7]:='Воскресенье';
for i:= 1 to 7 do writeln (i, '  ', day [i]);
                                  end.
```

Далее

Назад

Меню

Ввод массива

Под вводом массива понимается ввод значений элементов массива. Как и вывод массива, ввод удобно реализовать при помощи инструкции FOR. Чтобы пользователь программы знал, ввода какого элемента массива ожидает программа, следует организовать вывод подсказок перед вводом очередного элемента массива. В подсказке обычно указывают индекс элемента массива.

Пример:

Следующая программа запрашивает температуру воздуха в течении недели и запоминает введённые значения в массиве temper, затем вычисляет среднее значение. Для организации подсказок используется массив строк day.

Пример

Далее
Назад
Меню

Пример:

```
var
day: array [1..7] of string [11];  {название дней недели}
temper: array [1..7] of real;  {температура}
sum: real;  {сумма температур за неделю}
sredn: real;  {средняя температура за неделю}
i: integer;
begin
day [1]:='Понедельник';
day [2]:='Вторник';
day [3]:='Среда';
day [4]:='Четверг';
day [5]:='Пятница';
day [6]:='Суббота';
day [7]:='Воскресенье';
writeln ('Задайте температуру воздуха за неделю.');
for i:= 1 to 7 do
begin
write (day [i], '-->');
readln (temper [i]);
end;
{вычисление средней температуры за неделю}
sum:= 0;
for i:= 1 to 7 do
sum:= sum + temper [i];
sredn:= sum / 7;
writeln ('Средняя температура за неделю:', sredn: 6: 2);
end.
```

Далее

Назад

Меню

Сортировка массива

Под *сортировкой массива* подразумевается процесс перестановки элементов с целью упорядочивания их в соответствии с каким-либо критерием. Например, если имеется массив целых a , то после сортировки по возрастанию должно выполняться условие:

a [1] <= a [2] <= … <= a [SIZE]

SIZE - верхняя граница индекса массива.

Так как можно сравнивать переменные типов INTEGER, REAL, CHAR и STRING, то можно сортировать массивы этих типов.

Задача сортировки распространена в информационных системах и используется как предварительный этап задачи поиска, так как поиск в упорядоченном (отсортированном) массиве проводится намного быстрее, чем в неупорядоченном.

Существует много методов (алгоритмов) сортировки массивов. Здесь мы рассмотрим два метода:

- метод прямого выбора
- метод прямого обмена

Далее
Назад
Меню

Сортировка методом прямого выбора

Алгоритм сортировки массива по возрастанию методом прямого выбора может быть представлен так:

1. Просматривая массив от первого элемента, найти минимальный и поместить его на место первого элемента, а первый на место минимального.
2. Просматривая массив от второго элемента, найти минимальный и поместить его на место второго элемента, а второй на место минимального.
3. И так далее до последнего элемента.

В ***Примере*** представлена программа сортировки массива целых чисел по возрастанию. Для демонстрации процесса сортировки программа выводит массив после каждого обмена элементов.

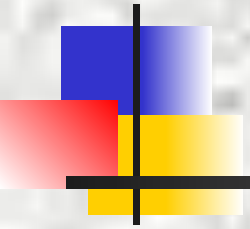
Далее
Назад
Меню

Пример:

```
const
SIZE = 5;
var
a: array [1..SIZE] of integer;
i: integer; {номер элемента, которого ведётся поиск минимального элемента}
min: integer; {номер минимального элемента в части массива от  i до верхней границы массива}
j: integer; {номер элемента, сравниваемого с минимальным}
buf: integer; {буфер, используемый при обмене элементов массива}
k: integer;
begin
writeln (‘Сортировка массива.’);
write (‘Введите’, SIZE: 3,’ целых в одной строке’);
writeln (‘через пробел и нажмите<ENTER>’);
for k:=1 to SIZE do read (a [k]);
begin
{поиск минимального элемента в части массива от f [i] до a [SIZE]}
min:= i;
for j:= i + 1 to SIZE do begin
if a [j] < a [min] then min:= j;
{поменяем местами a [min] и a [j]}
buf:= a [i];
a [i]:= a [min];
a [min]:= buf;
{выведем массив}
for k:= 1 to SIZE do write (a [k], ‘  ‘);
writeln;
                                    end;
end;
writeln (‘Массив отсортирован.’);
end.
```

Далее
Назад
Меню

Сортировка методом обмена (Метод пузырька)



В основе алгоритма лежит обмен соседних элементов массива. Каждый элемент массива, начиная с первого, сравнивается со следующим и если он больше следующего, то элементы меняются местами. Таким образом элементы с меньшим значением продвигаются к началу массива (всплывают), а элементы с большим значением - к концу массива (тонут), поэтому этот метод иногда называют “пузырьковым”. Этот процесс повторяется на единицу меньше раз, чем элементов в массиве.

Далее
Назад
Меню

Пример:

Программа сортировки массива целых чисел по возрастанию. Для демонстрации процесса сортировки программа выводит массив после каждого цикла обменов.

```
const
  SIZE
var
a: array [1..SIZE] of integer;
i: integer; {счетчик циклов}
k: integer; {текущий индекс элемента массива}
buf: integer;
begin
writeln (‘Сортировка массива пузырьковым методом.’);
write (‘Введите’,SIZE: 3,’целых в одной строке через пробел’);
writeln (‘и нажмите <ENTER>’);
for k:= 1 to SIZE do read (a [k]);
writeln (‘Сортировка.’);
for i:= 1 to SIZE - 1
do begin
for k:= 1 to SIZE - 1
do begin
if a [k] > a [k + 1]
then begin
{обменяем k-й и (k + 1)-й элементы}
buf:= a [k];
a [k]:= a [k + 1];
a [k + 1]:= buf;
end;
end;
for k:= 1 to SIZE do write (a [k], ‘  ’);
writeln (‘Массив отсортирован.’);
end.
```

Назад
Меню

Поиск в массиве

При решении многих задач возникает необходимость установить, содержит ли массив определенную информацию или нет. Например, проверить, есть ли в массиве фамилий студентов фамилия “Петров”. Задачи такого типа называются *поиском в массиве.*

Для организации поиска в массиве могут быть использованы различные алгоритмы. Наиболее простой - это алгоритм перебора. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

В **Примере** представлен текст программы поиска в массиве целых чисел. Перебор элементов массива осуществляет инструкция REPEAT, в теле которой инструкция IF сравнивает текущий элемент массива с образцом и присваивает переменной naiden значение TRUE, если текущий элемент равен образцу. Цикл завершается, если в массиве обнаружен элемент, равный образцу (naiden = TRUE), или если проверены все элементы массива. По завершении цикла, по значению переменной found можно определить, успешен поиск или нет.

Очевидно, что чем больше элементов в массиве и чем дальше расположен нужный элемент от начала массива, тем дольше будет программа искать нужный элемент.

Так как операции сравнения применимы как к числам, так и строкам, то данный алгоритм может использоваться для поиска как в числовых, так и в строковых массивах.

На практике довольно часто проводится поиск в массиве, элементы которого упорядочены по некоторому критерию. Например, массив фамилий, как правило, упорядочен по алфавиту, массив данных о погоде упорядочен по датам наблюдений.

Далее
Назад
Меню

Пример:

```
var
massiv: array [1..10] of integer; {массив целых}
obrazec: integer; {образец для поиска}
naiden: boolean; {признак совпадения с образцом}
i: integer;
begin
{ввод 10 целых чисел}
writeln ('Поиск в массиве.');
write ('Введите 10 целых в одной строке через пробел');
writeln ('и нажмите <ENTER>');
write ('-->');
for i:= 1 to 10 do read (massiv [i]);
{числа введены в массив}
write ('Введите образец для поиска (целое число) -->');
readln (obrazec);
{поиск простым перебором}
naiden:= FALSE; {совпадений нет}
i:= 1; {проверяем с первого элемента массива}
repeat
if massiv [i] = obrazec
then naiden:= TRUE  {совпадение с образцом}
else i:= i + 1;   {переход к следующему элементу}
until (i > 10) or (naiden); {завершим, если совпадение с образцом или}
{проверен последний элемент массива}
if naiden
then writeln ('Совпадение с элементом номер', i : 3, ' . ', 'Поиск успешен.')
else writeln ('Совпадений с образцом нет.');
end.
```

Назад
Меню

Поиск минимального (максимального) элемента массива

Алгоритм поиска минимального и максимального (максимального) элемента массива довольно очевиден: делается предположение, что первый элемент массива является минимальным (максимальным), затем остальные элементы массива сравниваются с этим элементом. Если обнаруживается, что проверяемый элемент меньше (больше) принятого за минимальный (максимальный), то этот элемент принимается за минимальный (максимальный) и продолжается проверка оставшихся элементов. Ниже представлена программа поиска минимального элемента в массиве целых чисел.

```
const
  GRANICA = 10;
var
  a: array [1 .. GRANICA] of integer; {массив целых чисел}
  min: integer; {номер минимального элемента массива}
  i: integer; {номер элемента сравниваемого с минимальным}
begin
  {здесь инструкции ввода массива}
  min:= 1; {пусть первый элемент минимальный}
  for i:= 2 to GRANICA do
    if a [i] < a [min] then min:= i;
  writeln ('Минимальный элемент массива:'., a [min]);
  writeln ('Номер элемента:', min);
end.
```

Далее

Назад

Меню

Многомерные массивы

Исходные данные для решения многих задач можно представить в табличной форме:

	Продукт1	Продукт2	Продукт3	Продукт4
Завод1				
Завод2				
Завод3				

В программе для хранения и обработки табличных данных можно использовать совокупность одномерных массивов. Например таблица результатов производственной деятельности нескольких филиалов фирмы может быть представлена так:

```
zavod1: array [1..4] of integer;
zavod2: array [1..4] of integer;
zavod3: array [1..4] of integer;
```

Для подобных случаев Pascal предоставляет более удобную структуру данных - двумерный массив.

В общем виде описание двумерного массива выглядит так:

```
Имя :array [НижняяГраницаИндекса1..ВерхняяГраницаИндекса1,
            НижняяГраницаИндекса2..ВерхняяГраницаИндекса2] of Тип
```

где *Имя* - имя массива; *array* - слово языка Pascal, показывающее, что описываемый элемент данных - массив;

НижняяГраницаИндекса1, ВерхняяГраницаИндекса1, НижняяГраницаИндекса2, ВерхняяГраницаИндекса2 - константы или выражения типа INTEGER, определяющие диапазон изменения индексов и, следовательно, число элементов массива;

Тип - тип элементов массива.

Далее
Назад
Меню

Приведенная выше таблица может быть представлена в виде двумерного массива так:

```
product : array [1..3, 1..4] of integer
```

Этот массив состоит из 12 элементов типа INTEGER.

Чтобы использовать элемент массива, нужно указать имя массива и индексы элемента. Первый индекс обычно соответствует номеру строки таблицы, второй - номеру колонки. Так элемент *product [2, 3]* содержит число продуктов третьего наименования, выпущенных вторым заводом.

Значения элементов двумерных массивов выводят на экран и вводят с клавиатуры, как правило, по строкам, т.е. Сначала все элементы первой строки, затем второй и т.д. Это удобно выполнять при помощи вложенных инструкций FOR. Следующий фрагмент программы выводит на экран значения элементов массива по строкам:

```
for i:= 1 to 3 do
begin
for j:= 1 to 4 do
write (product [i, j]);
writeln;
end;
```

Каждый раз, когда внутренний цикл завершается, внешний цикл увеличивает i на единицу, и внутренний цикл выполняется вновь. Таким образом, выводятся все компоненты массива product : product [1, 1], product [1, 2],... product [1, 4], product [2,1], product [2, 2],... product [2, 4] и т.д.

При описании массивов в программе удобно использовать именованные константы как значения верхних границ индексов массива.

Далее
Назад
Меню

Пример:

Представленная ниже программа обрабатывает результаты соревнований по легкой атлетике.

Клуб	Завоевано медалей:		
	Золотые	Серебряные	Бронзовые
Буревестник	4	4	4
Динамо	2	4	3
Зенит	6	4	4
Спартак	3	3	4

Программа считывает исходные данные, вводимые с клавиатуры, вычисляет общее количество медалей и затем расставляет клубы по порядку в соответствии с общим количеством медалей. Для представления данных о количестве медалей используется двумерный массив medal, количество строк которого на единицу больше, чем количество клубов, а количество столбцов на единицу больше, чем медалей. Дополнительный столбец используется для хранения общего количества медалей, которое вычисляется после ввода исходных данных, дополнительная строка - как буфер при обмене строк во время сортировки строк таблицы. ПРИМЕР

Далее
Назад
Меню

```
const
  NC = 4; {}
var
  club : array [1..NC + 1] of string [30]; {}
  medal: array [1.. NC + 1, 1..4] of integer; {}
  m, i, j : integer;
begin
  club [1] := ‘Буревестник’;
  club [1] := ‘Динамо       ’;
  club [1] := ‘Зенит        ’;
  club [1] := ‘Спартак      ’;
writeln (‘Для каждой команды в одной строке введите’);
writeln (‘через пробел’ число золотых, серебряных и’);
writeln (‘бронзовых медалей и нажмите <ENTER>’);
for i:= 1 to NC do
  begin
    write (club [i],’->’);
    readln (medal [i, 1], medal [i, 2], medal [i, 3]);
  end;
{подсчет общего количества медалей}
for i := 1 to NC do
  begin
    medal [i, 4] := 0;
    for j := 1 to 3 do
      medal [i, 4] := medal [i, 4] + medal [i, j];
  end;
{сортировка таблицы}
for i:= 1 to NC - 1 do
  begin
    {найти строку, в которой максимально общее число медалей}
    m := i;
    for j := i + 1 to NC do
      if medal [j, 4] > medal [m, 4] then m := j;
    {обменяем i-ю строку с m-й}
    club [NC + 1] := club [i];
    club [i] := club [m];
    club [m] := club [NC + 1];
    for j := 1 to 4 do
      begin
        medal [NC + 1, j] := medal [i, j];
        medal [i, j] := medal [m, j];
        medal [m, j] := medal [NC + 1, j];
      end;
  end;
{вывод итоговой таблицы}
writeln;
writeln (‘Клуб Золотые Серебряные Бронзовые Всего’);
writeln (‘** Итоговая таблица**’);
for i:= 1 to NC do
  begin
    write (i : 2, ‘. ‘, club [i]);
    for j := 1 to 4 do
      write (medal [i, j] : 11);
    writeln;
  end;
end.
```

Назад

Меню

Ошибки при использовании массивов

При использовании массивов наиболее распространенной ошибкой является превышение индексом значения выражения верхней границы индекса, указанной при объявлении массива. Если в качестве индекса используется константа и ее значение превышает верхнюю границу, то такая ошибка обнаруживается на этапе компиляции. Например во время компиляции программы

```
var
day : array [1..6] of string [11];
begin
day [1] := ‘Понедельник’;
day [7] := ‘Воскресенье’;
end.
```

будет выведено сообщение об ошибке для инструкции

day [7] := ‘Воскресенье’;

Если при обращении к элементу массива в качестве индекса используется переменная или выражение, то возможно возникновение ошибки времени выполнения программы (run time error). Например, в программе, представленной ниже, ошибок времени компиляции нет.

```
var
temper : array [1..12, 1..31] of real;
month, day: integer;
begin
writeln (‘’);
write ( ‘->’);
readln (day, month, t);
temper [month, day] := t;
end.
```

Далее
Назад
Меню

Однако если во время работы программы в ответ на запрос будет введена строка

-15 17 12

что соответствует -15 градусов 17 января, то при выполнении инструкции

temper [month, day] := t будет выведено сообщение

Run time error 104;

В программы, в которых возможны ошибки времени выполнения вследствие неправильного ввода исходных данных, следует добавлять инструкции проверки вводимых данных. Для приведенной выше программы это можно сделать, например, так:

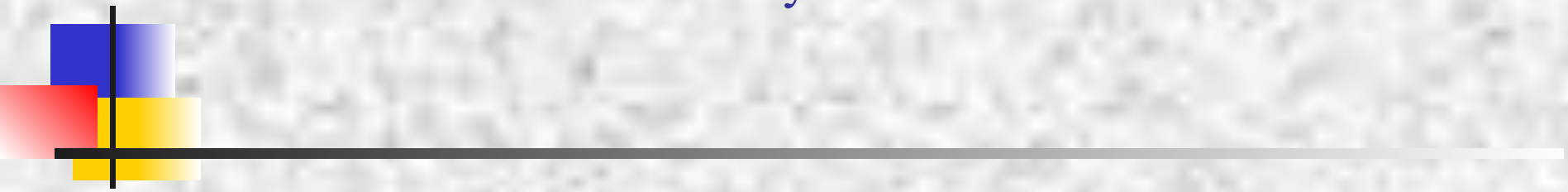
```
var
  temper : array [1..12, 1..31] of real;
  month, day : integer;
  t : real;
begin
  writeln (‘Веедите дату (число, номер месяца) и температуру воздуха’);
  write (‘->’);
  readln (day, month, t);
  if ((day . 31) OR (month > 12) OR (day < 1) OR (month < 1))
    then writeln (‘Неверные данные’);
    else temper [month, day] := t;
end.
```

Далее

Назад

Меню

Вы закончили знакомство с основами языка программирования ***Turbo Pascal***. Теперь Вы можете смело приступать к практическому применению полученных знаний.



ВЫХОД

Назад
Меню