Уважаемые студенты групп!

Вашему вниманию представлена лекция на тему «Логические выражения. Оператор условного перехода. Оператор безусловного перехода. Оператор выбора».

Задание

1. Прочитать внимательно лекцию.
2. Законспектировать лекцию в рабочую тетрадь не менее 3-5 страницы рукописного текста. В конспекте лекции обязательно должно быть приведены примеры.
3. Решить приведенные в лекции в контрольных вопросах задачи.

С уважением Ганзенко Ирина Владимировна
!!! Если возникнут вопросы обращаться по телефону 0721134803 (вацап), +79591134803 (телеграмм)
disobuch.ganzenko2020@mail.ru

СЛЕДУЮЩЕЕ ЗАНЯТИЕ ЛАБОРАТОРНОАЯ!!!!
Конспект лекции предоставить преподавателю до 10.02.2023

Лекция: Логические выражения. Оператор условного перехода. Оператор безусловного перехода. Оператор выбора

План

1 Процесс ветвления

Алгоритмы решения большинства задач не является последовательными. Действия (вычисления), которые необходимо выполнить, могут зависеть от определенной условия, например, от исходных данных, или результатов, полученных при выполнении программы. Таким образом, разветвленный алгоритм предусматривает выбор одного из нескольких последовательностей действий в зависимости от исходных данных или промежуточных результатов.

Вычислительный процесс называется ***разветвленным,*** если в зависимости от выполнения определенных условий он реализуется одним из нескольких, заранее предусмотренных (возможных), направлений. Каждый отдельное направление называется ветвью вычисления.

Разветвленные программы могут быть реализованы одним из трех способов: с использованием ***операторов перехода, условного оператора или оператора выбора***. Для этого существуют специальные инструкции (операторы) передачи управления, позволяющие перейти с одного места программы в другое (передать управление) и изменить последовательный порядок выполнения ее операторов (инструкций).

Если такой переход осуществляется только при выполнении определенного условия, он называется условным, а соответствующий ему оператор - оператором условного перехода. Если переход выполняется в любом случае, он называется безусловным, а соответствующий ему оператор - оператором безусловного перехода. Если в зависимости от значения какого-либо выражения необходимо выполнить один из нескольких последовательных операторов, использующих оператор выбора (варианта).

2 Логическое выражение

Логическое выражение - это средство записи условий для поиска нужных данных. Логическое выражение может принимать значения **true** (истинность) или **false** (ложь). Логические выражения бывают простые и составные. Простой - это два арифметических выражения, соединенные символом отношения, а составной - это простые логические выражения, соединенные названиями логических операций: and (и), not (нет) и or (или).

And (и) - если каждое условие, что входит в сложное условие, истинная, тогда всё сложное условие типа «и» тоже истинна. Если хотя бы одно условие, входит в сложное условие, ошибочно, тогда всё сложное условие типа «и» тоже ошибочно.

Or (или) - если хотя бы одно условие, что входит в сложное условие, истинная, всё сложное условие типа «или» истинна. Если ни одно условие, что входит в сложное условие, ошибочна, тогда всё сложное условие типа «или» тоже ошибочно.

Not (нет) - если каждое условие, что входит в сложное условие, истинная, тогда всё сложное условие типа «нет» ошибочно. Если каждое условие, что входит в сложное условие, ошибочна, тогда всё сложное условие типа «нет» истинна.

Рассмотрим определение логических операций.

Выражение	**Значение**	**Выражение**	**Значение**
not true	false	**not** false	true
true **and** true	true	true **or** true	true
true **and** false	false	true **or** false	true

false **and** true	false	false **or** true	true
false **and** false	false	false **or** false	false

3 Справочная информация по операторам ветвления языка Pascal

Оператор IF

При разработке любой программы необходим механизм, позволяющий выполнить тот или иной участок программного кода, основываясь на некотором условии. Для реализации такого рода алгоритмов в Pascal предусмотрен оператор ветвления IF. Основная форма вызова данного оператора:

if <условие> then <оператор>;
<остальные операторы программы>

Принцип действия данного оператора следующий: осуществляется проверка заданного условия. В том случае, если условие выполняется (дает результат «ИСТИНА», т.е. «TRUE»), то осуществляется переход к оператору, расположенному справа от THEN. После окончания работы этого оператора управление переходит к остальным операторам программы, которые расположены после символа «;». Если же условие не выполняется, то оператор, расположенный справа от THEN, пропускается и управление сразу переходит к остальным операторам.

Очень часто, в случае выполнения заданного условия, требуется выполнить не один, а сразу несколько операторов. В этом случае эти несколько операторов следует разместить внутри операторных скобок BEGIN..END.

Внимание! В программе может находиться несколько участков кода, размещенных внутри BEGIN..END, причем эти участки могут быть вложенными. Самый внешний BEGIN..END определяет начало и окончание программы, а остальные BEGIN..END выполняют роль операторных скобок.

Следует учитывать, что если некоторый набор операторов, состоящий из нескольких строк, находится внутри операторных скобок BEGIN..END, то всю конструкцию (с точки зрения оператора IF) следует рассматривать как «составной оператор»:

if A = B then
begin
  <оператор 1>;
  <оператор 2>;
  ...

<оператор N>
end;

Условие – это любое выражение, результатом которого является логическое (Boolean) значение: **True** или **False** (Да или Нет). При составлении условия можно использовать операторы сравнения:

«=» – равно	«<>» – не равно	«>» – больше
«<» – меньше	«>=» – больше или равно	«<=» – меньше или равно

Операция (A = B) вернет **True** в том случае, если переменные A и B равны между собой; в противном случае вернет **False**;

Операция (A <> B) вернет **True** только в том случае, если переменные A и B не равны между собой.

Операция (A > B) вернет **True** в том случае, если переменная A имеет значение большее, чем значение переменной B.

Операция (A >= B) вернет **True** в том случае, если переменная A **равна** переменной B, **либо** имеет значение большее, чем у переменной B.

Выражение ((A < -1) **or** (A > 1)) вернет **True**, если значение переменной A **меньше** -1 **или больше** 1, т.е. оно не лежит в диапазоне [-1 ÷ 1].

Выражение ((A = B) **and** (A > C * C)) вернет **True**, если переменные A и B равны между собой и в то же время значение переменной A превышает значение переменной C, взятое в квадрате. Если не будет выполнено хотя бы одно из условий, то выражение вернет **False**.

Операция «**not**» инвертирует результат логического выражения, указанного справа, т.е. (**not True**) вернет **False**, а (**not False**) вернет **True**.

Выражение (**not** (A = B)) вернет True в том случае, если A не равно B.

Внимание! Программа на языке Pascal всегда должна иметь наименование, указанное после ключевого слова PROGRAM, секцию VAR с объявленными переменными и **внешние** операторные скобки BEGIN..END, в которых должны располагаться все необходимые операторы. Для краткости изложения материала эти ключевые слова в некоторых дальнейших примерах пропущены! Также пропущен код, осуществляющий присвоение значения некоторым переменным.

Ниже представлен один из таких примеров. В нем осуществляется ввод числа A; если введенное число отрицательное, то его значение заменяется на ноль.

```
Readln(A);          {ввод числа A}
if A < 0 then A := 0; {заменяем на 0, если число отрицательное}
Writeln('A: ', A);    {выводим на экран значение переменной A}
```

В примере ниже осуществляется проверка делителя на 0. При равенстве нулю выдается сообщение об ошибке и происходит выход из программы.

```
program Delenie;
var
  Delimoe, Delitel, Res: Real;
begin
  Delimoe := 100;
  Readln(Delitel);            {ввод значения делителя}
  if Delitel = 0 then         {проверка на равенство нулю}
  begin                       {начало составного оператора}
    Writeln('Ошибка: на ноль делить нельзя!');
    Exit;                     {досрочный выход из программы}
  end;                        {составной оператор закончился}
  Res := Delimoe / Delitel;   {осуществляем деление}
  Writeln('Результат: ', Res); {вывод результата на экран}
end.
```

Очень часто возникает необходимость отреагировать не только на выполнение условия, но и на его невыполнение. Для этого к конструкции IF..THEN добавляется ключевое слово ELSE, определяющее начало альтернативной ветви выполнения программного кода. В этом случае конструкция IF..THEN выглядит следующим образом:

```
if <условие> then
  <оператор1>
else
  <оператор2>;
```

```
if <условие> then
begin
  <группа операторов 1>;
end
else
begin
  <группа операторов 2>;
end;
```

Следует отметить, что перед ключевым словом ELSE **не должна** стоять точка с запятой.

На рисунке 2.1 приведен пример схемы разветвляющегося алгоритма.

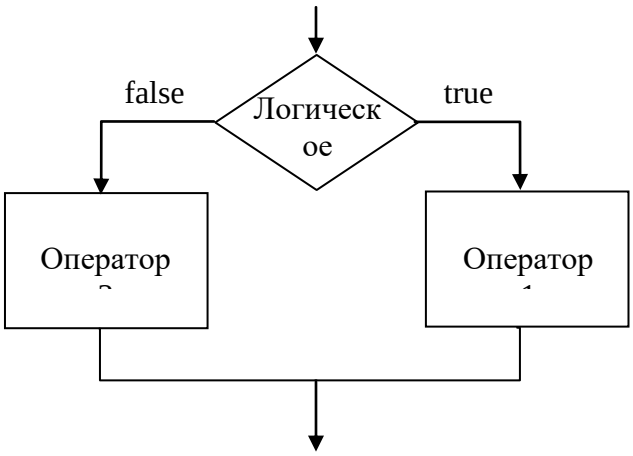

Рисунок 2.1 – Схема алгоритма

В представленном ниже примере осуществляется поиск наибольшего значения среди X и Y и сохранение найденного значения в переменную Max:

```
if X > Y then
  Max := X
else
  Max := Y;
Write('Максимум: ', Max);
```

```
if X >= Y then
begin
  Max := X;
  Write('X больше или равен Y');
end
else
begin
  Max := Y;
  Write('Y больше X');
end;
Write('Максимум: ', Max);
```

Кроме того, оператор IF..THEN может быть вложенным, причем уровень вложенности не ограничивается, например:

```
if X < -3 then
  Y := X + 1
else if (X > 3) and (X < 10) then
  Y := X * X
else if X >= 10 then
begin
  Y := Sqrt(X);
  Writeln('Y: ', Y);
end
else
  Y := Y * Y;
```

Следует отметить, что подобные конструкции на практике могут быть весьма громоздкими. Для улучшения читабельности кода рекомендуется чаще пользоваться операторными скобками BEGIN..END с необходимым выравниванием. Важно помнить, что ключевое слово ELSE относится только к одному, ближайшему оператору IF..THEN, расположенному выше по коду.

Константы

Перед тем, как познакомиться с оператором CASE, необходимо дать определение понятию «константа». Константой в языке Pascal является некоторое значение (например, числовое), заданное непосредственно в тексте программы (т.е. пользователь вашей программы его не вводит). Например, в операторе «A := 100» переменной A присваивается явно заданное значение

«100», т.е. константа. В языке Pascal различают два вида констант: именованные и неименованные. Для того чтобы константа была именованной, ее необходимо указать в секции CONST в разделе описаний программы: сначала указывается имя константы, затем символ «=», далее указывается необходимое значение, например:

```
program ConstExample;
const
  MinLimit = 1;     {Минимальный лимит}
  MaxLimit = 100;   {Максимальный лимит}
  Pi = 3.14;        {Число Пи}
.......
begin
  A := MaxLimit;    {Это более осмысленно, чем A := 100}
  if B < MinLimit then...
.......
end;
```

После того, как константа объявлена, ее имя можно использовать в программе вместо числового значения, например «A := MaxLimit». В некоторых случаях это позволяет улучшить читабельность программы, а также упростить ее дальнейшую разработку. Значение именованной константы невозможно изменить при выполнении программы (в отличие от переменной).

В данной лабораторной работе рекомендуется использовать константы для обозначения постоянных параметров, которые не требуется вводить пользователю вашей программы. Например, стоимость 1 кВт/час является величиной постоянной, поэтому вы можете ее объявить с помощью именованной константы:

```
const
  KiloWattCost = 3.45;     {Стоимость 1 кВт/час }
```

Оператор выбора CASE

В том случае, если задана некоторая переменная порядкового типа (целочисленная, логическая или символьная) и на каждое возможное ее значение программа должна отреагировать индивидуально, рекомендуется использовать оператор выбора CASE. Логика работы оператора CASE аналогична логике IF..THEN, однако, использование оператора CASE в **некоторых случаях** позволяет значительно улучшить читабельность кода.

Оператор CASE имеет следующий формат:

case <переменная или выражение порядкового типа> **of**

```
  <константа или список констант 1> : <оператор 1>;
  <константа или список констант 2> : <оператор 2>;
  ……————————————————————
  <константа или список констант n> : <оператор N>;
else
  <альтернативная ветвь: оператор или группа операторов>
end;
```

Логика работы оператора CASE следующая: сначала программа определяет значение переменной или выражения порядкового типа (например, целочисленное). Далее отыскивается константа, совпадающая с указанным значением, после чего выполняется оператор, расположенный после символа «:». Если программе не удалось найти константу, совпадающую с заданным значением, то выполняется оператор из альтернативной ветви, расположенной после ключевого слова ELSE. Ключевое слово ELSE не является обязательным (его следует указывать, когда в этом возникает необходимость).

В приведенном ниже примере пользователь вводит степень числа N от 1 до 3. Программа возводит переменную X в степень N. Отдельно обрабатывается случай, когда N равен нулю. Во всех остальных случаях устанавливается значение 0.

```
  Write('Введите значение n: ');
  Readln(n); {ожидаем, когда пользователь введет n}
  case n of
    0: {демонстрация использования операторных скобок begin...end}
      begin
        Writeln('Сообщение: любое число в степени 0 равно единице!');
        Y := 1;
      end;
    1: Y := X;
    2: Y := X * X;
    3: Y := X * X * X;
  else {альтернативная ветвь кода}
    {здесь дополнительный begin...end не требуется}
    Writeln('Вы ввели недопустимое число!');
    Y := 0;
  end; {конец оператора case}
  Writeln('Результат: ', Y); {вывод результата на экран}
```

Кроме одиночных констант могут быть заданы списки и/или диапазоны значений. Например:

```
  case n of
    0, 2..4 : Y := A * B; {оператор будет выполнен для n: 0, 2, 3, 4}
```

```
  1, 5   : Y := A / B;
  6      : Y := (A + B) * (A - B);
end;
```

Следует отметить, что при использовании оператора CASE действует ряд ограничений:

– значения констант не должны дублироваться; это ограничение действует также при использовании диапазонов;

– тип констант должен соответствовать типу заданной переменной; если переменная целочисленная, то и все константы должны быть целыми;

– заданная переменная должна иметь порядковый тип (например, Integer, Byte, Char, Boolean); она не может быть объявлена как Real (дробный тип) или **string** (строка).

Оператор GOTO

Оператор безусловного перехода GOTO (англ.: **перейти к**) позволяет прервать выполнение текущего участка кода и перейти к другому участку, если он отмечен меткой безусловного перехода. Метка безусловного перехода объявляется в разделе LABEL и должна соответствовать требованиям, предъявляемым к идентификаторам (в порядке исключения допускается вместо наименования метки использовать целочисленные значения). После объявления метки в разделе LABEL ее можно указать в любом месте (но только один раз) в тексте программы. Для того чтобы перейти на заданную метку, следует вызвать оператор GOTO <имя_метки>, причем количество операторов перехода на одну и ту же метку в программе не ограничено:

```
……————————————————————
goto M1; {переходит вниз на метку M1}
<операторы>;
M1:
<операторы>;
……————————————————————
goto M1; {переходит вверх на метку M1}
```

```
……————————————————————
label
  M1;
var
  X, Y: Real;
begin
  Readln(X);
  ……————————————————————
  goto M1;
  ……————————————————————
M1:
  Y := X * 2 – 3 / X;
  Writeln('Y=', Y);
  ……————————————————————
end.
```

Следует отметить, что в современном программировании использование оператора GOTO не приветствуется, поскольку

злоупотребление данным оператора приводит к сильному «запутыванию» кода. Для избежания использования оператора GOTO следует применять другие методы, например циклы и подпрограммы (см. следующие лабораторные работы).

Пример программы с разветвленной структурой

Составить программу вычисления функции:

$$y = \begin{cases} \frac{1}{x} & \text{при } x > 0 \\ x^2 & \text{при } x < 0 \end{cases}$$

```
program Lab2;
label
  M1, M2; {объявление меток}
var
  n: Integer;
  X, Y: Real;
  Flag: Boolean;      {Признак выполнения пункта N1}
begin
  Writeln('Программа вычисления функции. Автор: Иванов И.И.');
  { Вывод на экран меню }
  Writeln('Введите цифру для выполнения действия:');
  Writeln('1 - Ввод данных');
  Writeln('2 - Вычисление функции и вывод результатов');
  Writeln('3 - Завершение работы программы');
  Flag := False; { Первоначальная инициализация флага }
M1:
  Write('Введите номер пункта меню: ');
  Readln(n); { Ввод номера пункта меню}
  case n of
    1: { Ввод данных }
      begin
      M2:
        Write('Введите значение аргумента X: ');
        Readln(X);
        { Проверка допустимости значения аргумента }
        if X = 0 then
        begin
          Writeln('X не может быть равным 0 по условию');
          goto M2; { переход к M2 для повторного ввода данных }
        end;
        Flag := True; {Пункт №1 выполнен, установка флага в True}
      end;
```

```
  2: { Вычисление значения функции }
    begin
      if not Flag then {Если пункт №1 не выполнен}
        Writeln ('Данные не введены, выполните пункт №1');
      else
      begin {пункт №1 был выполнен}
        { Операторы вычисления и вывода значения функции  }
        if X > 0 then {если X положительный}
          Y := 1 / X
        else {иначе X < 0}
          Y := X * X;
        Writeln('При X  =  ', X:7:2, ' Y  =  ', Y:7:2);
      end;
    end;
  3: Exit; { Выход из программы }
 end; { end case }
 goto M1; { переход в режим выбора пункта меню }
end. { Конец программы}
```

Данный пример требует пояснения. Программа начинается с объявления меток безусловного перехода (M1, M2) и объявления переменных, в том числе логической однобайтной переменной **Flag: Boolean**.

Как ранее было сказано, логическая переменная может иметь всего два значения: **True** или **False**. В начале работы программы переменной **Flag** присваивается значение **False**.

Это необходимо, поскольку если не выполнить этого присвоения, то в начале работы программы значение переменной **Flag** не определено, т.е. она случайным образом может быть равна False или True. Следует обратить внимание, что для пункта №2 оператора CASE осуществляется проверка переменной Flag (**if not** Flag **then** …), а поскольку осуществляется обращение к переменной в режиме чтения, то значение переменной обязательно должно быть присвоено **заранее**.

В приведенном примере проверка (**if not** Flag **then** …) будет препятствовать выполнению операторов вычисления до тех пор, пока пользователь в пункте №1 не введет допустимое значение аргумента X (Flag в этом случае будет выставлен в **True**).

4 Контрольные вопросы

1. Что такое разветвленный вычислительный процесс?
2. Напишите пример оператора условного перехода.
3. Как записывается и выполняется оператор условного перехода в полной и сокращенной формах?
4. Что такое составной оператор?

5. Можно ли в операторе ветвления использовать составные операторы?

6. Какие особенности использования символа «;» в операторе условного перехода?

7. Что такое логическое условие?

8. Напишите пример оператора безусловного перехода.

9. Как записывается и для чего используется оператор безусловного перехода?

10. Для чего предназначены метки, как они описываются?

11. Что считается плохим стилем программирования относительно оператора goto и почему?

12. Напишите пример оператора выбора?

13. В каких случаях используется оператор выбора?

14. Что такое селектор и требования к его использованию?

15. Какие из следующих операторов ветвления являются:

1) правильными _____

2) неправильными _____

а) if a <b then a = a + 1 else b = b-1;

б) if (x <5) and (y> 3) then s = s + 1 else s = s-1;

в) if a <> b then b = a;

г) if 9 then k = k + 1.

16. Какие значения будут иметь переменные a и b в результате выполнения оператора ветвления:

if a <b then a = b else b = a,

если перед выполнением a = 0.5, b = -1.7

Ответ a = _____ b = _____

17. Какая задача решается в результате выполнения оператора:

if x <y then max = y else max = x?

ответ

18. При каких значениях a i b, условие a> b будет:

1) истинная _____ 2) ошибочное _____

а) a = 2, b = 5, б) a = 5, b = 2; в) a = 2, b = 2.

19. Найдите значение величины C после выполнения оператору разветвления:

if b> = 6 then c = 5 else c = 12, если

1) b = 2, c = _____ 2) b = 16, c = _____ 3) b = 6, c = _____

20. Какое значение величины d, если после выполнения оператору

if d <= 6 then c = 5 else c = 10;

1) c = 5, d = _____ 2) c = 10; d = _____

а) d = 4; б) d = 10; в) d = 2.

21. При каком значении k, после выполнения оператору варианта

case k of

k1: writeln ('Отлично');

k2: writeln ('Хорошо');

k3: writeln ('Средне');

k4: writeln ('Удовлетворительно');
end;

1) Хорошо k = _____ а) k1 = 5
2) Средне k = _____ б) k2 = 4
3) Удовлетворительно k = _____ в) k3 = 3
4) Отлично k = _____ г) k4 = 4

22. Какое значение будет иметь переменная S после выполнения указания разветвления

```
Label 1,2;
begin
s = 1;
m = 3;
1: if k> = m then goto 2;
s = s + m;
k = k + 1;
goto 1;
2: end.
```

если

1) k = 2 s = _____ а) s = 5 д) s = 6
2) k = 3 s = _____ б) s = 1 е) s = 4
3) k = 1 s = _____ в) s = 10
4) k = 5 s = _____ г) s = 7
5) k = 0 s = _____ г) s = 3