Syntax:

Syntax for the if-then-else statement is:

if condition then S1 else S2;

Where, **S1** and **S2** are different statements. **Please note that the statement S1 is not followed by a semicolon**. In the if-then-else statements, when the test condition is true, the statement S1 is executed and S2 is skipped; when the test condition is false, then S1 is bypassed and statement S2 is executed.

For example,

if color = red then

writeln('You have chosen a red car')

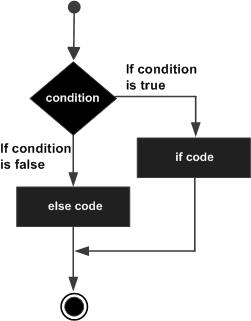
else

writeln('Please choose a color for your car');

If the boolean expression **condition** evaluates to true, then the if-then block of code will be executed, otherwise the else block of code will be executed.

Pascal assumes any non-zero and non-nil values as true, and if it is either zero or nil, then it is assumed as false value.

Flow Diagram:



Example:

Let us try a complete example that would illustrate the concept:

program ifelseChecking;

var

{ local variable definition }

a : integer;

begin

a := 100;

(\* check the boolean condition \*)

if( a < 20 ) then

(\* if condition is true then print the following \*)

writeln('a is less than 20' )

else

(\* if condition is false then print the following \*)

writeln('a is not less than 20' );

writeln('value of a is : ', a);

end.

When the above code is compiled and executed, it produces the following result:

a is not less than 20

value of a is : 100

The if-then-else if-then-else Statement

An if-then statement can be followed by an optional else if-then-else statement, which is very useful to test various conditions using single if-then-else if statement.

When using if-then , else if-then , else statements there are few points to keep in mind.

* An if-then statement can have zero or one else's and it must come after any else if's.
* An if-then statement can have zero to many else if's and they must come before the else.
* Once an else if succeeds, none of the remaining else if's or else's will be tested.
* No semicolon (;) is given before the last else keyword, but all statements can be compound statements.

Syntax:

The syntax of an if-then-else if-then-else statement in Pascal programming language is:

if(boolean\_expression 1)then

S1 (\* Executes when the boolean expression 1 is true \*)

else if( boolean\_expression 2) then

S2 (\* Executes when the boolean expression 2 is true \*)

else if( boolean\_expression 3) then

S3 (\* Executes when the boolean expression 3 is true \*)

else

S4; ( \* executes when the none of the above condition is true \*)

Example:

The following example illustrates the concept:

program ifelse\_ifelseChecking;

var

{ local variable definition }

a : integer;

begin

a := 100;

(\* check the boolean condition \*)

if (a = 10) then

(\* if condition is true then print the following \*)

writeln('Value of a is 10' )

else if ( a = 20 ) then

(\* if else if condition is true \*)

writeln('Value of a is 20' )

else if( a = 30 ) then

(\* if else if condition is true \*)

writeln('Value of a is 30' )

else

(\* if none of the conditions is true \*)

writeln('None of the values is matching' );

writeln('Exact value of a is: ', a );

## IF THEN (ELSE) Command Syntax in Pascal

In Pascal (as in many other programming languages) exists the command IF-THEN (ELSE). The syntax of this command should be:

**BASIC SYNTAX OF THE IF-THEN COMMAND IN PASCAL**

**IF (condition) THEN command**

If we also want to specify what should happen if the condition is not valid, the command IF-THEN includes the part ELSE which is executed if the condition is not valid.

**SYNTAX OF THE IF-THEN (ELSE) COMMAND IN PASCAL**

**IF (condition) THEN command-1 ELSE command-2**

## Simple Conditions

Take a look at the following simple example:

**IF-THEN IN PASCAL**

**Program** IsNumberGreaterThanFive;  
 **var** a:integer;  
 **begin**  
  **readln**(a);  
  **IF** (a>5) **THEN** **writeln**('a is greater than 5');  
  **readln**;  
 **end**.

If we want to have more exact information we use the complete IF-THEN-ELSE commands:

**IF-THEN-ELSE IN PASCAL**

**Program** IsNumberGreaterOrLowerThanFive;  
 **var** a:integer;  
 **begin**  
  **readln**(a);  
  **IF** (a>5) **THEN**  
    **writeln**('a is greater than 5')  
  **ELSE**  
    **writeln**('a is not greater than 5');  
  **readln**;  
 **end**.

If we want to be even more precise and evaluate both inequality and equality, we have to nest another IF-THEN in one of the branches of IF-THEN-ELSE:

**NESTED IF-THEN IN PASCAL**

**Program** IsNumberGreaterLowerThanFiveOfEqual;  
 **var** a:integer;  
 **begin**  
  **readln**(a);  
  **IF** (a>5) **THEN**  
    **writeln**('a is greater than 5')  
  **ELSE**  
    **IF** (a<5) **THEN**  
      **writeln**('a is lower than 5')  
    **ELSE**  
      **writeln**('a is equal to 5');  
  **readln**;  
 **end**.

In this example you can notice a Pascal syntax specialty that the semicolon is not written at the end of the line before ELSE.

## Complex Conditions, Operators AND and OR

If a more complex condition is needed, it can be put together from more conditions using the operators AND and OR.

### AND Operator

The operator AND is a parallel to the conjunction "and". If we put together several conditions using AND, then the resultant condition is valid **ONLY if EACH** condition is valid. In everyday life we use this operator in the following consideration: "When it's raining AND I am out THEN I open my umbrella." or "When it is raining on me AND I have my umbrella THEN I open it." (IF (it`s raining on me) AND (I have got my umbrella) THEN (I open it)).

**AN EXAMPLE USING THE AND OPERATOR**

**Program** Operator\_AND;  
 **var** a,b:integer;  
 **begin**  
  **readln**(a);  
  **readln**(b);  
  **IF** ((a<0) AND (b<0)) **THEN**  
    **writeln**('Numbers a and b are both negative')  
  **ELSE**  
    **writeln**('Numbers a and b are not both negative');  
 **end**.

Note that the whole condition ((a <0) AND (b <0)) is closed in the brackets - usually it is not necessary, but in some programming languages it is required to close any complex condition in the brackets, that is why it is good way to get used to this syntax.

### OR Operator

The operator OR is parallel to the conjunction "or". If we put together several conditions using OR, then the resultant condition is valid when **AT LEAST ONE** of the conditions is valid (and it is not valid only if none of the conditions is valid). In everyday life we use this operator in the following considerations, for example: "When I am hungry or I like tasty meals, I take something delicious." or "When I want to be a good student or I am bored, I learn something." (IF (I want to be a good student) OR (I`m bored) THEN (I learn)).

**AN EXAMPLE USING THE OR OPERATOR**

**Program** Operator\_OR;  
 **var** a,b:integer;  
 **begin**  
  **readln**(a);  
  **readln**(b);  
  **IF** ((a>0) OR (b>0)) **THEN**  
    **writeln**('At least one of the numbers a and b is greater than zero.')  
  **ELSE**  
    **writeln**('Both numbers a and b are smaller than or equal to zero.');  
 **end**.

You can see the results of AND and OR operators, 1 means true, 0 means false.

| **a** | **b** | **a AND b** | **a OR b** |
| --- | --- | --- | --- |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |

## Example of Advanced Mathematical Formula

In previous lessons we solved the calculation of simple formulas. Some of them already looked quite complicated. Try to write a program for calculating a simple quotient of two numbers a and b.

**Sample:** Calculation of the quotient of two arbitrarily given numbers **a** and **b**.

**A QUOTIENT OF TWO NUMBERS IN PASCAL**

**Program** Quotient\_A\_a\_B;  
 **var** a,b:integer;  
 **begin**  
 writeln('Enter number a:');   
 readln(a);  
 writeln('Enter number b:');   
 readln(b);  
 if (b=0) then  
  writeln('Division by zero is not defined, the quotient cannot be computed.')  
 else  
  writeln('The quotient of the two given numbers is ',a/b)  
 readln;  
 **end**.

As we know from Math, division by zero is not an allowed operation. If we forget this fact, the program crashes when it tries to divide by zero (which is an unacceptable status). We can see that without the condition IF-THEN it is not possible to handle a simple quotient of any two numbers correctly.

We can complicate the task slightly by the requirement that the previous programme should store the quotient of a / b in a variable c. The following solution is the first thing that comes to our minds:

**A QUOTIENT OF TWO NUMBERS IN PASCAL**

**Program** Podil\_A\_a\_B;  
 **var** a,b:integer;  
     c:real;  
 **begin**  
 writeln('Enter number a:');   
 readln(a);  
 writeln('Enter number b:');   
 readln(b);  
 if (b=0) then  
  writeln('Division by zero is not defined, the quotient cannot be computed.')  
 else  
  c:=a/b;  
  writeln('The quotient of the two given numbers is ',c)  
 readln;  
 **end**.

Does our program actually work reliably and well? To verify this try to enter zero as a value of b. What can we see? Surprisingly, the fact that the program provides us the result! Why? The reason is that in case that b is zero, the program prints that division by zero is not a defined operation, but at the same time it displays the text "Quotient ..." and prints a value.

Why does the program write "Quotient ..."? It is because of the fact that IF and ELSE statements are linked to only one following command and then the program continues evaluating the rest of source code.

If the IF is valid c: = a / b is not carried out, because it is an ELSE instruction. Nevertheless, the command (writeln ...) is always executed, because it is the second command after ELSE and thus it is not affected by the IF-THEN-ELSE command. If you want to carry out more than one command in IF or ELSE, you have to close them between "begin" and "end" commands so that the program knows which commands belong to IF and ELSE parts. The correct solution is displayed below this text:

**A QUOTIENT OF TWO NUMBERS IN PASCAL**

**Program** Podil\_A\_a\_B;  
 **var** a,b:integer;  
     c:real;  
 **begin**  
 writeln('Enter number a:');   
 readln(a);  
 writeln('Enter number b:');   
 readln(b);  
 if (b=0) then  
  writeln('Division by zero is not defined, the quotient cannot be computed.')  
 else  
  begin  
    c:=a/b;  
    writeln('The quotient of the two given numbers is ',c);  
  end;  
 readln;  
 **end**.

# Pascal - While-do Loops

A **while-do** loop statement in Pascal allows repetitive computations till some test condition is satisfied. In other words, it repeatedly executes a target statement as long as a given condition is true.

Syntax:

The syntax of a while-do loop is:

while (condition) do S;

Where, **condition** is a Boolean or relational expression whose value would be true or false and **S** is a simple statement or group of statements within BEGIN ... END block.

For example,

while number>0 do

begin

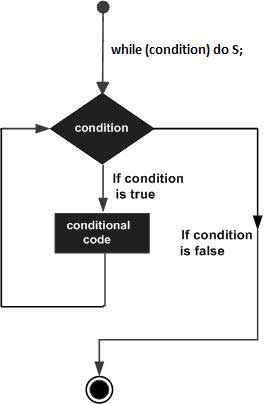
sum := sum + number;

number := number - 2;

end;

When the condition becomes false, program control passes to the line immediately following the loop.

Flow Diagram:



Here, key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example:

program whileLoop;

var

a: integer;

begin

a := 10;

while a < 20 do

begin

writeln('value of a: ', a);

a := a + 1;

end;

end.

When the above code is compiled and executed, it produces the following result:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

# Pascal - For-do Loop

A **for-do** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:

The syntax for the for-do loop in Pascal is as follows:

for < variable-name > := < initial\_value > to [down to] < final\_value > do

S;

Where, the *variable-name* specifies a variable of ordinal type, called control variable or index variable;*initial\_value* and *final\_value* values are values that the control variable can take; and S is the body of the for-do loop that could be a simple statement or a group of statements.

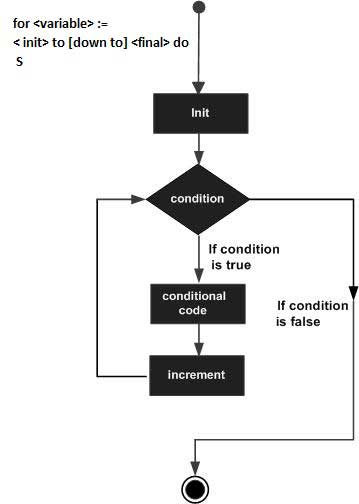
For example,

for i:= 1 to 10 do writeln(i);

Here is the flow of control in a for-do loop:

* The initial step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
* Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for-do loop.
* After the body of the for-do loop executes, the value of the variable is either increased or decreased.
* The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for-do loop terminates.

Flow Diagram



Example:

program forLoop;

var

a: integer;

begin

for a := 10 to 20 do

begin

writeln('value of a: ', a);

end;

end.

When the above code is compiled and executed, it produces the following result:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

value of a: 20

# Pascal - Repeat-Until Loop

Unlike for and while loops, which test the loop condition at the top of the loop, the **repeat ... until** loop in Pascal checks its condition at the bottom of the loop.

A repeat ... until loop is similar to a while loop, except that a repeat ... until loop is guaranteed to execute at least one time.

Syntax:

repeat

S1;

S2;

...

...

Sn;

until condition;

For example,

repeat

sum := sum + number;

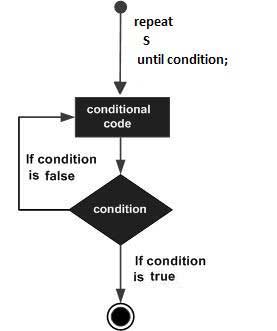
number := number - 2;

until number = 0;

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is false, the flow of control jumps back up to repeat, and the statement(s) in the loop execute again. This process repeats until the given condition becomes true.

Flow Diagram:



Example:

program repeatUntilLoop;

var

a: integer;

begin

a := 10;

(\* repeat until loop execution \*)

repeat

writeln('value of a: ', a);

a := a + 1

until a = 20;

end.

When the above code is compiled and executed, it produces the following result:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19