Algorithm and Programming 2

CHAPTER1

Arithmetic operator

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
div	Divides numerator by denominator	B div A will give 2
mod	Modulus Operator AND remainder after an integer division	B mod A will give 0

Example

```
program calculator;
var
a,b,c : integer;
d: real;
begin
   a:=21;
   b:=10;
   c := a + b;
   writeln(' Line 1 - Value of c is ', c );
   c := a - b;
   writeln('Line 2 - Value of c is ', c );
   c := a * b;
   writeln('Line 3 - Value of c is ', c );
   d := a / b;
   writeln('Line 4 - Value of d is ', d:3:2 );
   c := a mod b;
   writeln('Line 5 - Value of c is ' , c );
   c := a div b;
   writeln('Line 6 - Value of c is ', c );
end.
```

Relational operator

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes, then condition becomes true.	(A = B) is not true.
<>	Checks if the values of two operands are equal or not, if values are not equal, then condition becomes true.	(A <> B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes, then condition becomes true.	(A <= B) is true.

Example

```
program showRelations;
var
a, b: integer;
begin
   a := 21;
   b := 10;
   if a = b then
      writeln('Line 1 - a is equal to b' )
   else
      writeln('Line 1 - a is not equal to b' );
   if  a < b then
      writeln('Line 2 - a is less than b' )
   else
      writeln('Line 2 - a is not less than b' );
   if  a > b then
      writeln('Line 3 - a is greater than b' )
   else
      writeln('Line 3 - a is greater than b' );

   (* Lets change value of a and b *)
   a := 5;
   b := 20;
```

Boolean operator

Operator	Description	Example
and	Called Boolean AND operator. If both the operands are true, then condition becomes true.	A and B) is false.
and then	It is similar to the AND operator, however, it guarantees the order in which the compiler evaluates the logical expression. Left to right and the right operands are evaluated only when necessary.	(A and then B) is false.
or	Called Boolean OR Operator. If any of the two operands is true, then condition becomes true.	(A or B) is true.
or else	It is similar to Boolean OR, however, it guarantees the order in which the compiler evaluates the logical expression. Left to right and the right operands are evaluated only when necessary.	(A or else B) is true.
<=	Called Boolean NOT Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	not (A and B) is true.

Example

```
program beLogical;
var
a, b: boolean;
begin
   a := true;
   b := false;

   if (a and b) then
      writeln('Line 1 - Condition is true' )
   else
      writeln('Line 1 - Condition is not true');
   if  (a or b) then
      writeln('Line 2 - Condition is true' );

   (* lets change the value of a and b *)
   a := false;
   b := true;
   if  (a and b) then
      writeln('Line 3 - Condition is true' )
   else
      writeln('Line 3 - Condition is not true' );
   if not (a and b) then
   writeln('Line 4 - Condition is true' );
end.
```

Operator presedence

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7. Here the precedences table :

Operators	Precedence
~, not,	Highest
*, /, div, mod, and, &	
|, !, +, -, or,	
=, <>, <, <=, >, >=, in	
or else, and then	Lowest

Decision making/branching

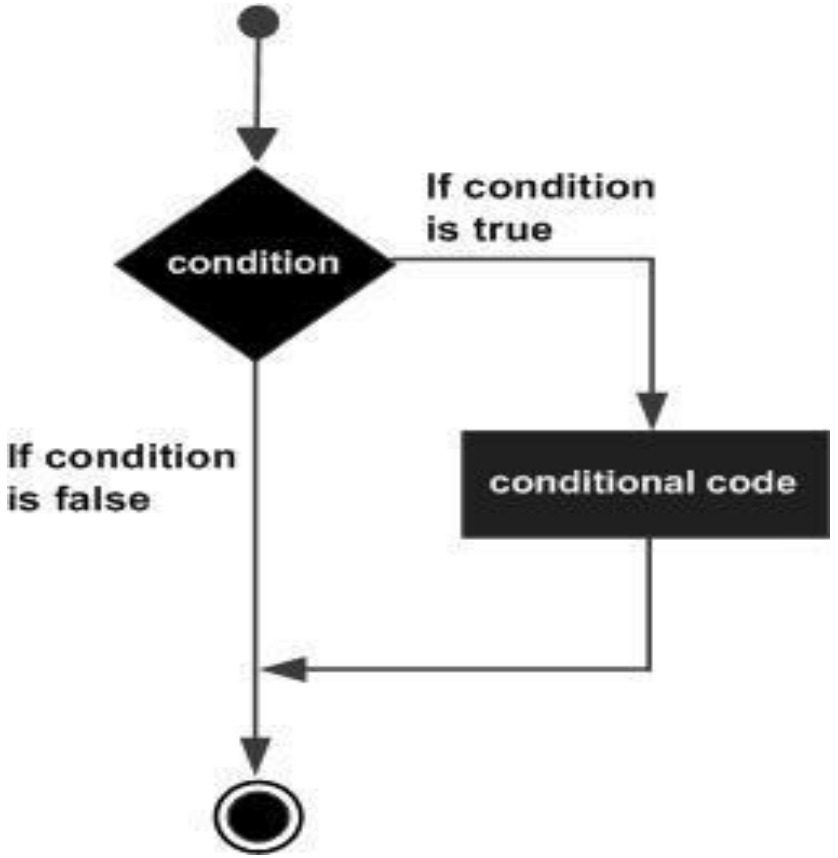
Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Here is following types of decision making statements details :

Statement	Description
if - then statement	An **if - then statement** consists of a boolean expression followed by one or more statements.
If-then-else statement	An **if - then statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
case statement	A **case** statement allows a variable to be tested for equality against a list of values.
case - else statement	It is similar to the **if-then-else** statement. Here, an **else** term follows the **case statement**.
nested case statements	You can use one **case** statement inside another **case** statement(s).

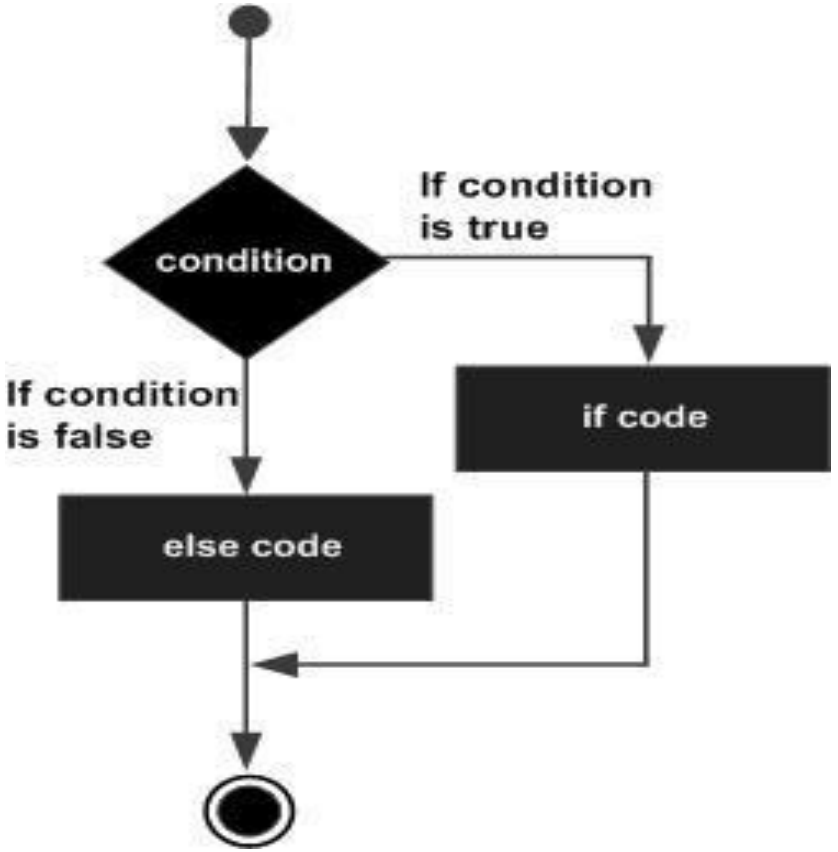
If - then

if condition then S



If – then - else

if condition then S1 else S2;



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.

```
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; (*when the none of the above condition is true*)
```

Nested if – then statement

It is always legal in Pascal programming to nest **if-else** statements, which means you can use one **if** or **else if** statement inside another **if** or **else if** statement(s). Pascal allows nesting to any level.

```
if( boolean_expression 1) then
   if(boolean_expression 2)then S1
else
   S2;
```

You can nest else if-then-else in the similar way as you have nested if-then statement. Please note that, the nested **if-then-else** constructs gives rise to some ambiguity as to which else statement pairs with which if statement.

The rule is that the else keyword matches the first if keyword (searching backwards) not already matched by an else keyword.

Equivalen with :

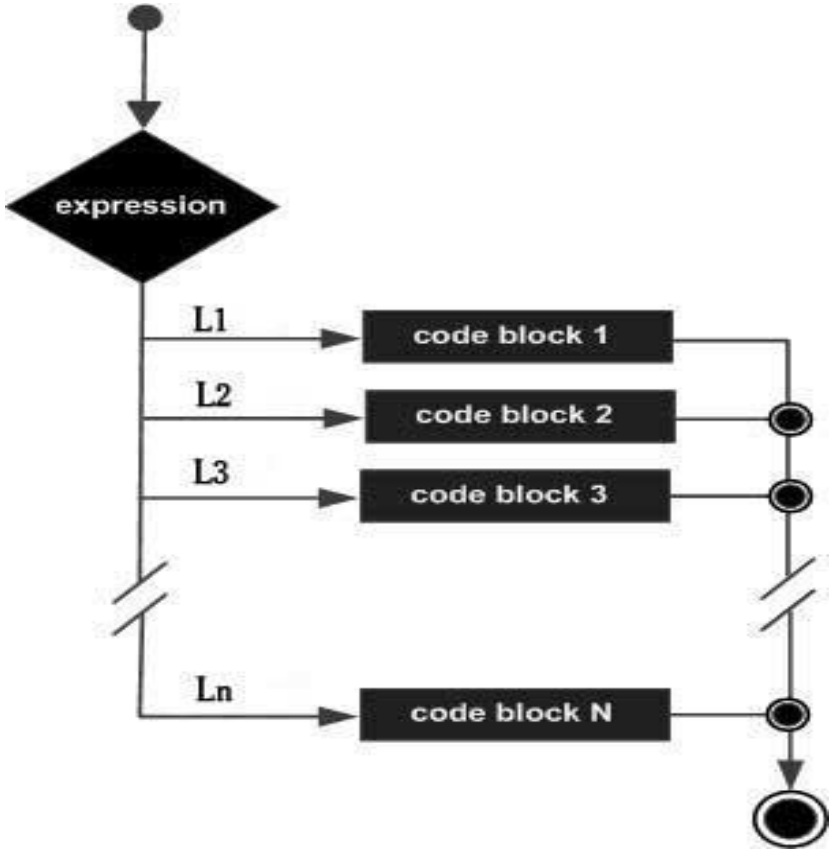
```
if( boolean_expression 1) then
   begin
   if(boolean_expression 2)then
      S1
else
      S2;
end;
```

Not equivalent with :

```
if ( boolean_expression 1) then
   begin
   if exp2 then
      S1
   end;
else
      S2;
```

Case statement

```
case (expression) of
   L1 : S1;
   L2: S2;
   ...
   ...
   Ln: Sn;
end;
```



Where, **L1**, **L2**... are case labels or input values, which could be integers, characters, boolean or enumerated data items. **S1**, **S2**, ... are Pascal statements, each of these statements may have one or more than one case label associated with it. The expression is called the **case selector** or the **case index**. The case index may assume values that correspond to the case labels.

The case statement must always have an **end** statement associated with it.

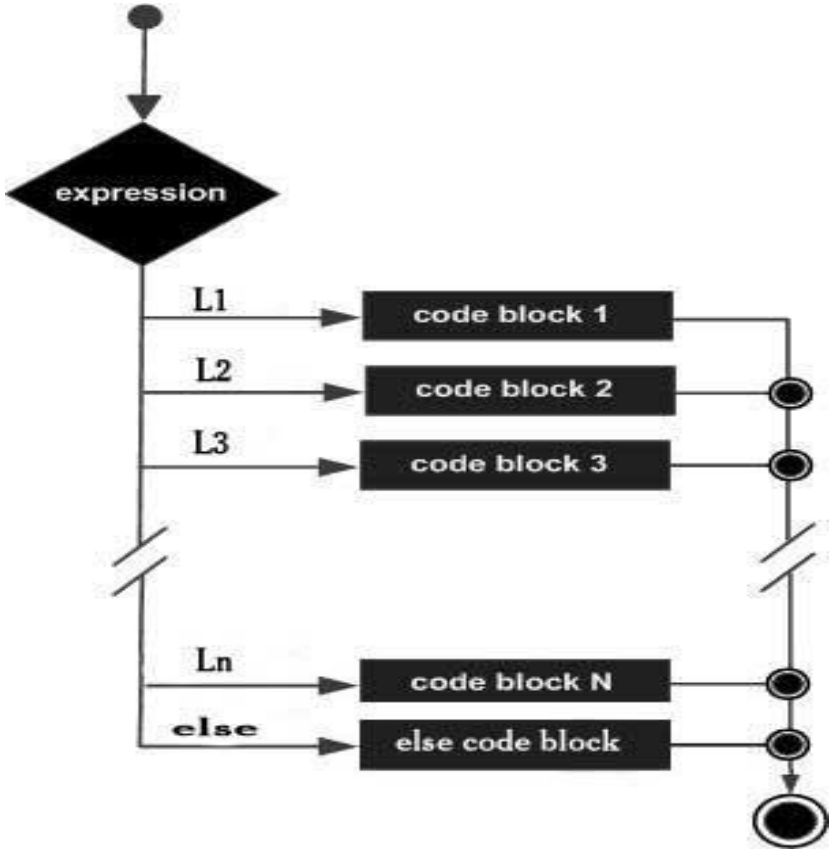
Rules

- You can have any number of case statements within a case. Each case is followed by the value to be compared to and a colon.
- The case label for a case must be the same data type as the expression in the case statement, and it must be a constant or a literal.
- The compiler will evaluate the case expression. If one of the case label's value matches the value of the expression, the statement that follows this label is executed. After that, the program continues after the final end.

- If none of the case label matches the expression value, the statement list after the else or otherwise keyword is executed. This can be an empty statement list. If no else part is present and no case constant matches the expression value, program flow continues after the final end.
- The case statements can be compound statements (i.e., a Begin ... End block).

Case – else statement

```
case (expression) of
   L1 : S1;
   L2 : S2;
   ...
   ...
   Ln: Sn;
else
   Sm;
end;
```



Thanks …