# C PROGRAMMING LANGUAGE
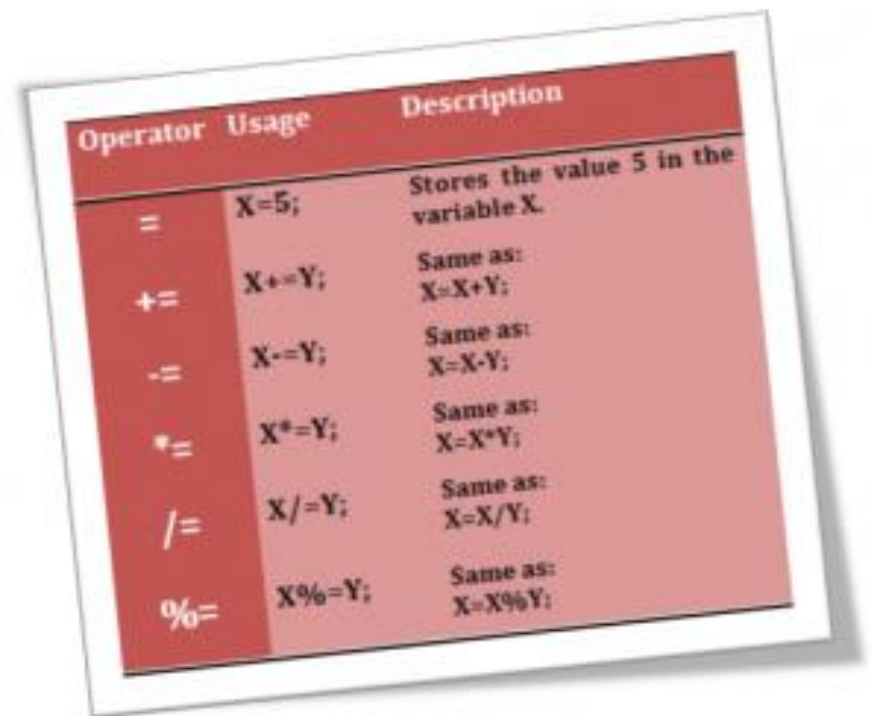
Mansoura

Open Source

# Operators

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Ternary Operator

| Operator | Usage | Description |
|---|---|---|
| = | X=5; | Stores the value 5 in the variable X. |
| += | X+=Y; | Same as: X=X+Y; |
| -= | X-=Y; | Same as: X=X-Y; |
| *= | X*=Y; | Same as: X=X*Y; |
| /= | X/=Y; | Same as: X=X/Y; |
| %= | X%=Y; | Same as: X=X%Y; |

# Arithmetic Operators

- **Unary Operators ++ , --.**
  - ++ add one to it's operand and – subtract one from it's operand
    - int d=10;
    - d++; /* value of d will be 11 */
  - Postfix : x++ ; uses the original value of x in the equation, and then increments it afterword's.
  - Prefix: ++x ; increments x first, and then uses the new value of x  inside the equation.
  - Example: x=10;  y = x++ ;  res: y=10 and x=11 as y=x;  then x=x+1;
  - Example: x=10; y = ++x ; res y=11 and x=11 as x=x+1; then y=x;
- **Binary Operators + , - , * , / , % (remainder).**

**Operates on two operands as sum =num1 + num2; the two operands are num1 and num 2**
  - If the two operands of / are int the result is int so the fraction part will be eliminated.
  - % modulus operator only operate with int operands and returns the remainder of division.
  - If the two operands  are int the result is int.
  - If the two operands are  float the result is float.
  - If  one operand is  int and the other is float the result is float.
  - + , - have less precedence than *,/,%.
  - +,-,*,/,% are left associativity
  - ++ ,-- have higher precedence than +,-,*,/,%.

# Operators

- **Assignment Operators. =  +=  -=  *=  /=  %=  &=  |=  ^=  <<=  >>=**
  - x=y; means assign the value of y to x, so the value of x and y will be the same.
  - lvalue=rvalue assigns the right value ( can be constant or variable ) to the left value ( must be variable ),
  - x+=y means x=x+y, x*=y+1 means x=x*(y+1); and so on.
- **Relational Operators.**  == != < > <= >=
  - x==y means ask if x equal to y or not , the result is true or false.
  - False in c means 0 other values treated as true so  values such as  1, 10 , -2 , -10 will be treated as true.
- **Logical Operators.**  !  &&  ||
  - Compare to relational expressions such as (a>b) && (b>c)
  - && the result will be true only if the two operands are true otherwise the result is false.
  - || the result is false only if the two operands are false otherwise the result is true.
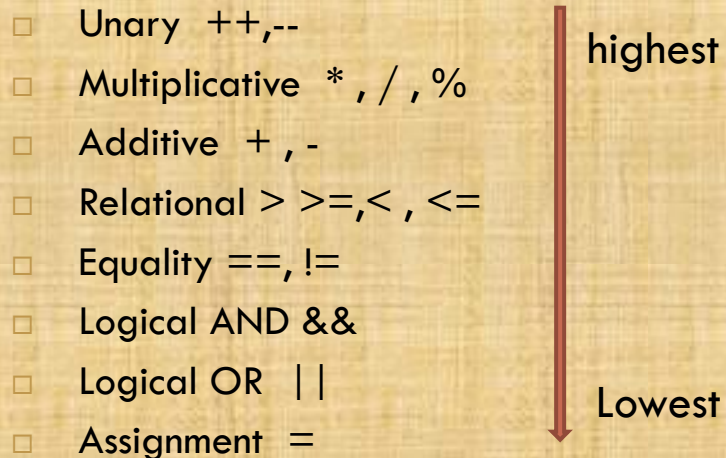- **Bitwise Operators.**
  - & , | , ^
  - << Left Shift
  - >> Right Shift
  - ~ Complement.
- **Ternary Operator (Conditional Operator).**
  - Variable = condition **?** value when true **:** value when false ;

# Operator precedence

The precedence of operators determines the order in which operations are performed in an expression. Operators with higher precedence are executed first. If two operators in an expression have the same precedence, associativity determines the direction in which the expression will be evaluated.

- Unary  ++,--
- Multiplicative  * , / , %
- Additive  + , -
- Relational > >=,< , <=
- Equality ==, !=
- Logical AND &&
- Logical OR  ||
- Assignment  =

highest

Lowest

# Control Statements

## Conditional Statements

- Switch Case statement
- If Else

## Loop Statements

- For loop
- While loop
- Do While loop

# Conditional Statements

These statements are used to have a program execute different statements depending on certain conditions. It makes a program "smarter" by allowing different choices to be made by the program.

In C, there are three conditional statements.

- **if** execute a statement or not

- **if-else** choose to execute one of two/more statements

- **Switch** choose to execute one of a number of statements

# If Statement

The if statement allows branching (decision making) depending upon a condition. **Program code is executed or skipped**. The basic syntax is
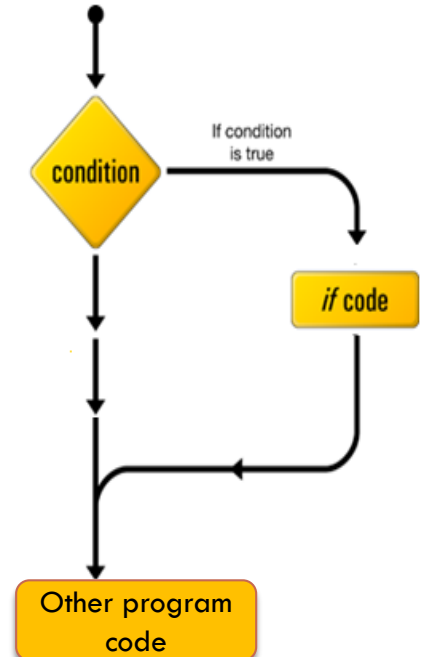
if ( control expression ){
         program statement;
}

If the control expression is TRUE, the body of the if is executed. If it is FALSE, the body of the if is skipped.

To avoid dividing by zero
if( y!=0 )
         z=x/y;
The statement z=x/y is executed only if y does not equal zero. If y=0 the z=x/y statement will not be executed.

```
if (condition)
{
    /*Some Logic If Code*/
}
/* other program code*/
```

# If-else Statement

**Used to decide between two action.** The syntax of the **if-else** statement is

**if (*expression*)**

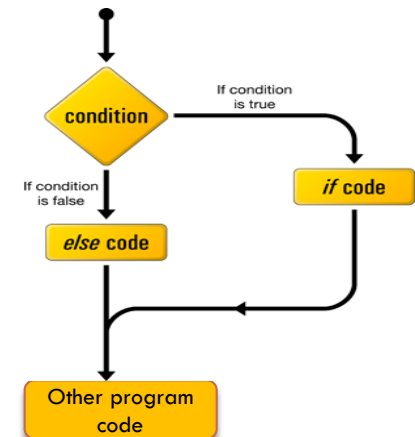        *statement1;*

**else**

        *statement2;*

**The statement 1 is executed if expression is true otherwise statement2 is executed**

To decide if a student succeeded or fail

if( degree>=60)

        printf("succeeded");

else

        printf("fails");

The succeeded word will be printed only if the degree of student is greater than or equal to 60. if the student degree less than 60 the fail word will be printed

```
if (condition)
{
    /*Some Logic If Code*/
}
else{
  /* else code */
}
/* other program code*/
```

# If-else if  Statement

If we want to display the grade of student in a subject. The grade of student will be one of the following grade ( EX. , VG., G, A, F) according to his degree. In this situation we can use if-else if.

For example
if(i>0)
        printf("positive number");
else if(i<0)
        printf("negative number");
else
        printf("zero number");

```
if (condition 1)
{
    /*this code will be
executed only if condition 1
true*/
}
else if(condition 2)
{
    /*this code will be executed
only if condition 1 false and
condition 2 true */
 }
else
{
    /*if condition 1 and 2 are false
}
```

# Switch-case statement

The **switch** statement is a alternative way of writing a program which employs an **if-else if** in condition that all condition are equality condition and data type to check is int or char.

The syntax for the **switch** statement is as follows:

**switch (***integer expression***) {**

        **case** *constant1***:**

                *statement1;*

                **break;**

        **case** *constant2***:**

                *statement2;*

                **break;**

        **...**

        **default:**

                *statement;*

        **}**

```c
char x;
switch (x) //switch expression may be
int or char- float is not allowed
{
    case 'a':
        //Some Logic
    break;
    case 'b':
        //Some Logic
    break;
    case 'c':
        //Some Logic
    break;
    default:
        //Some Logic
    break;
}
```

# Switch-Case statement

The **switch** statement works as follows
1 Integer control expression is evaluated.
2- A match is looked for between this expression value and the case *constants*. If a match is found, execute the statements for that case. If a match is not found, execute the default statement (if present).
3- Terminate switch when a break statement is reached.

- Case constants must be unique (How to decide otherwise?)
– case constants must be simple constant.
– switch statement only tests for equality
–The *control expression* can be of type character since characters are internally treated as integers

# Switch-Case statement

The statement **break** should be included at the end of each case statement. In general, whenever a statement is encountered in C, it interrupts the normal flow of control. In the **switch** statement, it causes an exit from the switch. If the break statement does not exists a fall through to the next case happens. The **default** clause is optional. The right brace at the end marks the end of **switch** statement

```c
char x;
switch (x)
{
    case 'a':
        /* no break. It
        will fall through
        to the next case b*/
    case 'b':
        //Some Logic
    break;
}
```

```c
char x;
switch (x)
{
// case grouping
    case 'a':
    case 'A':
        //Some Logic
    case 'b'
        //Some Logic
    break;
}
```
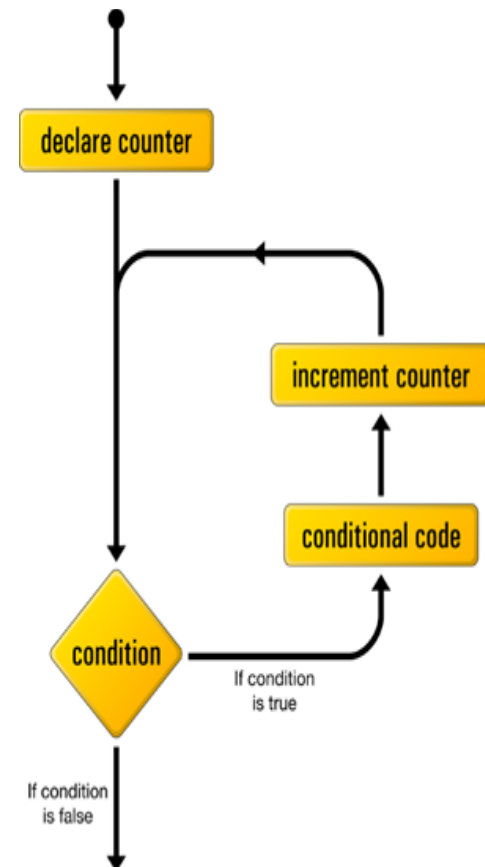
# For Loop

The for loop is used to execute statements number of times according to a counter.

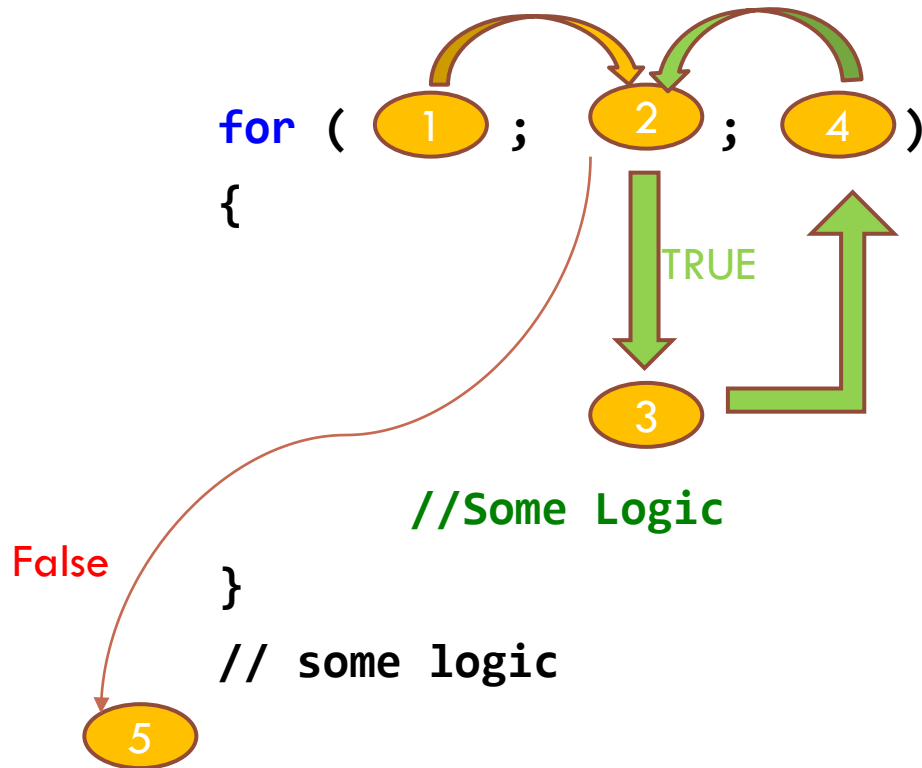for example the following code will print hello world 5 times.

```
for( counter=0; counter<5; counter++)
{
        printf("hello world");
}
```

The operation of the for loop is as follows

**1- The initialization expression (declaration) is evaluated.**

**2- The test condition is evaluated. If it is TRUE, the body of the loop is executed. If it is FALSE, exit the for loop.**

**3- Assume test condition is TRUE. Execute the program statements making up the body of the loop.**

**4- Evaluate the increment expression and return to step 2.**

**5- When test condition is FALSE, exit loop and move on to next line of code.**

declare counter

increment counter

conditional code

condition

If condition is true

If condition is false

# For Loop



```
for ( 1 ; 2 ; 4 )
{

                    TRUE


        //Some Logic


}
// some logic
```
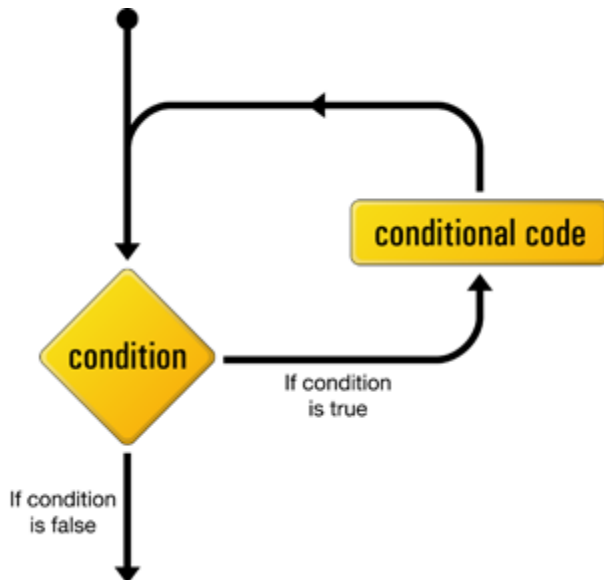
3

False

5

The for loop is used to execute statements number of times according to a counter for example the following code will print hello world 5 times.
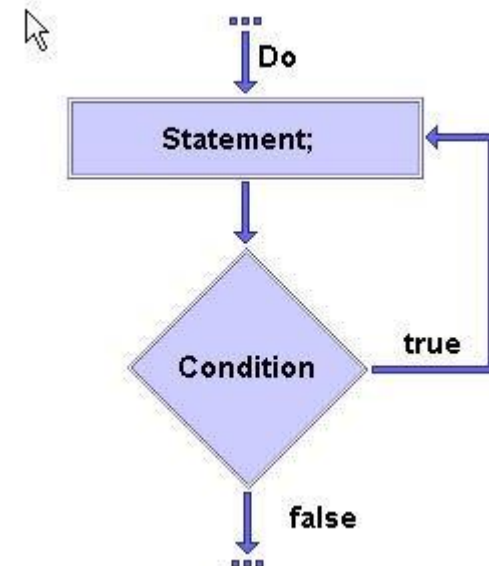for( counter=0; counter<5; counter++)
{
        printf("hello world");
}

# While Loop & Do While Loop

```
while (condition)
{
    //Some Logic
}
```



```
do
{
    //Some Logic
} while (condition);
```

# Lab.

1- simple menu with three choices ( new , display , exit ).

Inputs: n،d or e Characters

Output:

- if user enters **n** PRINT **New** and **continue** accepting new character,

- if he enters **d** PRINT **Display** and **continue** accepting new character,

- if he enters **e** print **EXIT** and **exit** from program

2-print grade of a student.

Input: degree between (0 & 100)

Output:

- if degree between 90 and 100 print Excellent.

- if degree between 80 and 90 print verygood.

- if degree between 70 and 80 print good.

- if degree between 60 and 70 print Acceptable.

- if degree between 0 and 60 print Fail.

# Lab.

3- receive numbers from the user and exit when sum exceeds 100.

Inputs: continue inserting numbers until there sum exceeds 100

Output: sum

4- magic box ( to be explained )

# Home.

1- prime number

Input : number n

Output: print if it is prime or not

Example:

i/p: 7 – o/p => prime.

i/p: 8 – o/p => not prime.

2- print prime numbers until specific number

Input : number n

Output: print all prime numbers until n.

Example:

i/p : 7 – o/p  => 2, 3 , 5 , 7

i/p : 20 – o/p => 2, 3 , 5 , 7 , 11 , 13 , 17 , 19