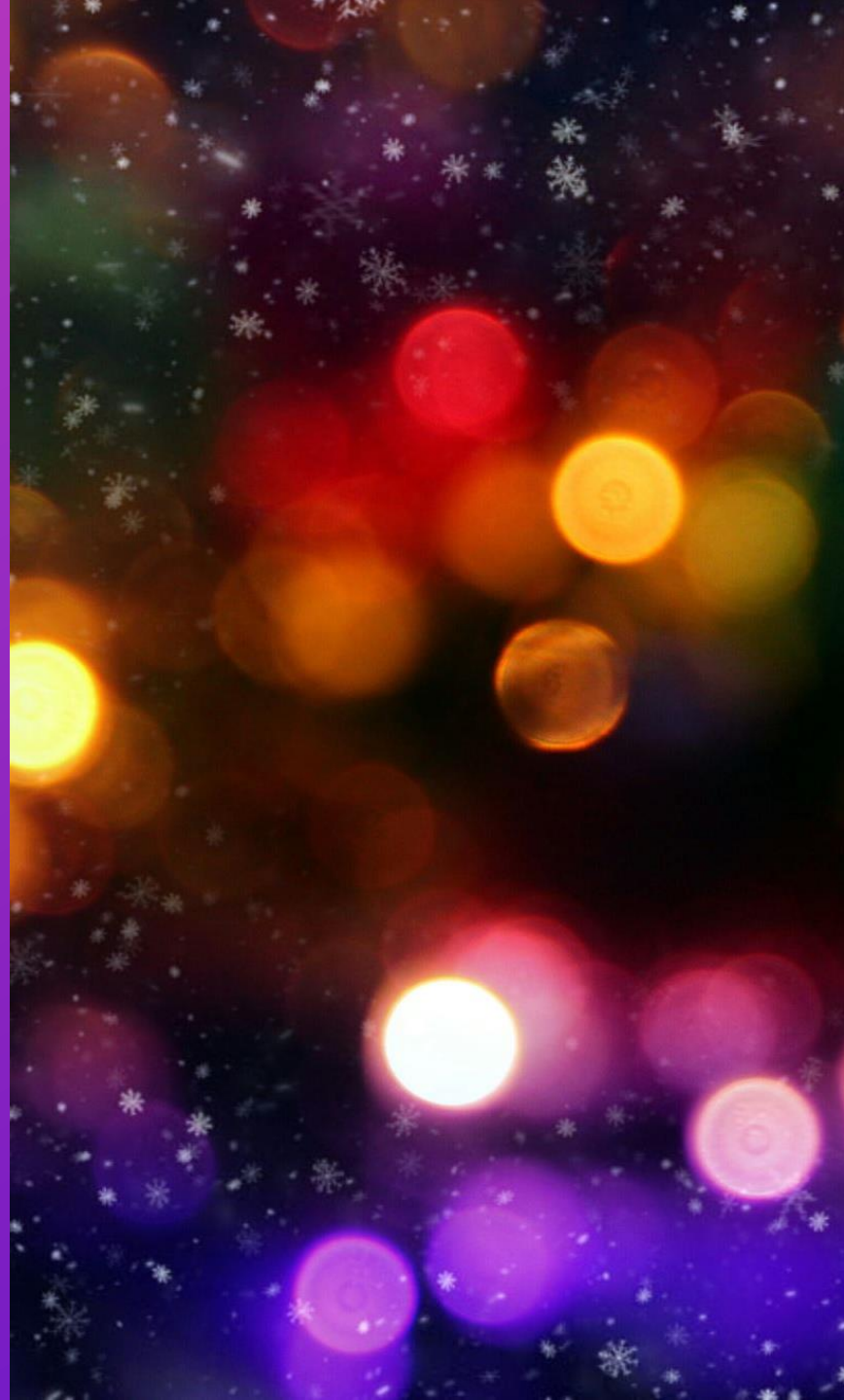
MNNIT COMPUTER CODING CLUB

CLASS-3

BASICS OF C



PRECEDENCE AND ASSOCIATIVITY

Operator	Description	Precedence level	Associativity
( ) [ ] → .	Function call Array subscript Arrow operator Dot operator	1	Left to Right
+ - ++ - - ! ~ * & (datatype) sizeof	Unary plus Unary minus Increment Decrement Logical NOT One's complement Indirection Address Type cast Size in bytes	2	Right to Left
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
= = !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
? :	Conditional operator	13	Right to Left
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right

ESCAPE SEQUENCES

```c
#include<stdio.h>

int main()
{
    printf("Hello World\n");
    printf("Hello\tWorld\n");

    printf("\"");
    printf("\n");

    printf("\'");
    printf("\n");

    printf("\\");
    printf("\n");

    printf("\141");
    printf("\n");

    printf("\x61");
    printf("\n");

    return 0;
}
```

Output:

```
Hello World
Hello   World
"
'
\
a
a
```

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

CONTROL STATEMENTS

The control flow statement in a language specify the order in which computations are performed.

As discussed in the previous class, expressions such as

- x = 0;
- i++;
- printf(...);

become a statement when followed by a semicolon.

Braces { and } are used to group declarations and statements together into a compound statement, or block, so that they are syntactically equivalent to a single statement.

C provides two styles of flow control:

- Branching
- Looping

  Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

IF STATEMENT

- The syntax of the `if` statement in C programming is:

```
if (test expression)
{
   // statements to be executed if the test expression is true
}
```

Working

The `if` statement evaluates the test expression inside the parenthesis `()`.

- If the test expression is evaluated to true, statements inside the body of `if` are executed.
- If the test expression is evaluated to false, statements inside the body of `if` are not executed.

IF ELSE STATEMENT

The `if` statement may have an optional `else` block. The syntax of the `if..else` statement is:

```
if (test expression) {
    // statements to be executed if the test expression is true
}
else {
    // statements to be executed if the test expression is false
}
```

Working

If the test expression is evaluated to true,

- statements inside the body of `if` are executed.
- statements inside the body of `else` are skipped from execution.

If the test expression is evaluated to false,

- statements inside the body of `else` are executed
- statements inside the body of `if` are skipped from execution.
- Conditional Operator

```c
#include <stdio.h>
int main()
{
    int num;
    printf( "Enter a number\n");
    scanf("%d",&num);
    if(num<0){
        printf("Number entered is negative");
    }else{
        printf("Number entered is non negative");
    }
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int num;
    printf( "Enter a number\n");
    scanf("%d",&num);
    if(num<0){
        printf ("Number entered is negative");
    }
    return 0;
}
```

IF ELSE PROGRAMS

IF ELSE LADDER

When a choice has to be made from more than 2 possibilities, the `if...else` ladder is used. The `if...else` statement executes a block of code depending upon whether the test expression is true or false.

The if...else ladder allows you to check between multiple test expressions and execute different statements.

```
if (test expression1) {
   // statement(s)
}
else if(test expression2) {
   // statement(s)
}
else if (test expression3) {
   // statement(s)
}
.
.
else {
   // statement(s)
}
```

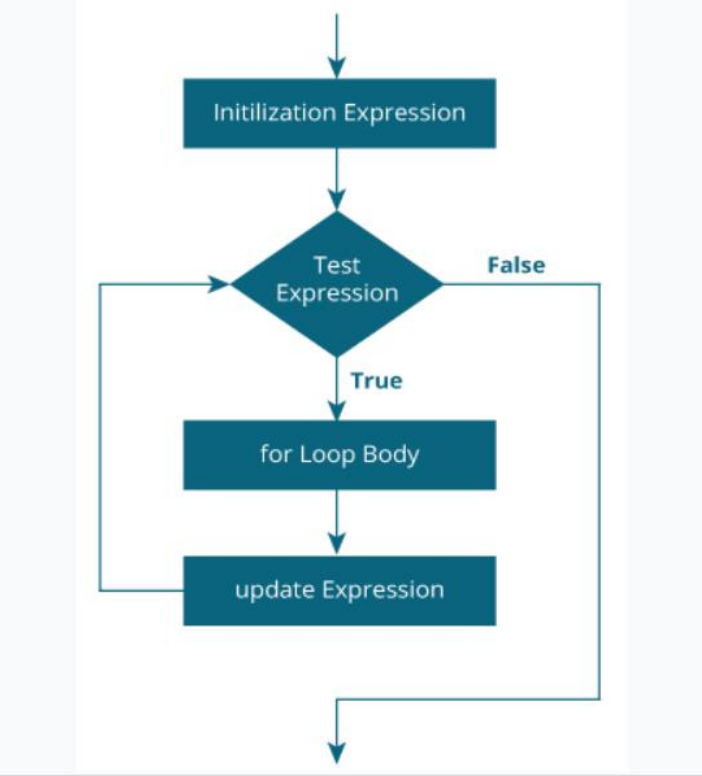
LOOPS IN C

In programming, a loop is used to repeat a block of code until the specified condition is met.

C programming has three types of loops:

- for loop
- while loop
- do...while loop

FOR LOOP



How for loop works?

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again, the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.

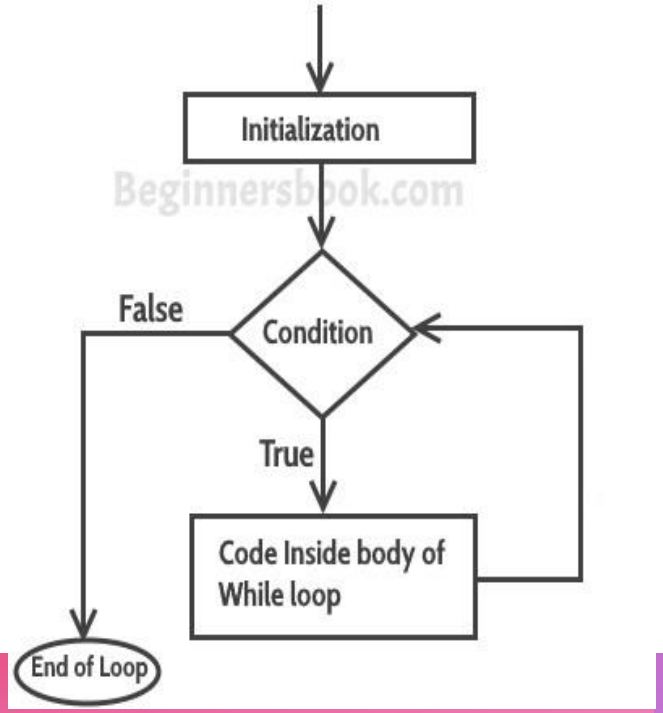
```
for (initializationStatement; testExpression; updateStatement)
{
    // statements inside the body of loop
}
```

```c
#include<stdio.h>
int main()
{
    int i;
    for(i=0; i<5; i++)
    {
        printf("Value of i: %d\n", i);
    }
}
```

WHILE LOOP

How while loop works?

- The initialization statement is kept out and executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the wile loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of while loop are executed. These also contains update statements if any.
- Again, the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.

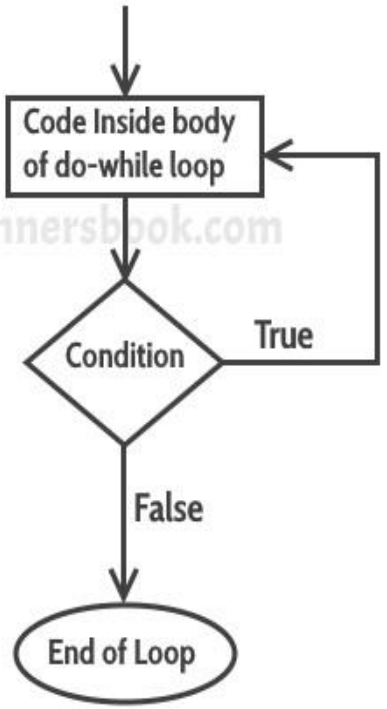



```c
#include<stdio.h>
int main()
{
    int i; //Initialisation
    while(i<5) //Test Condition
    {
        printf("Value of i: %d", i); //Body
        i++; //Updation
    }
}
```

DO-WHILE LOOP

How DO while loop works?

- The initialization statement is kept out and executed only once.
- Then, the body of the loop gets executed first including any updation specified inside body
- Then, the body exits, and test expression specified in while() is executed
- If the test expression is true, again the statements inside body are executed and testing is done untill condition becomes false
- If the test expression is false, then the statements after while(); are executed.




```c
#include<stdio.h>
int main()
{
    int i; //Initialisation
    do
    {
        printf("Value of i: %d", i); //Body
        i++; //Updation
    }
    while (i<5);
}
```

Difference between do while and while Loop

do-while	while
It is exit controlled loop	It is entry controlled loop
The loop executes the statement at least once	loop executes the statement only after testing condition
The condition is tested before execution.	The loop terminates if the condition becomes false.
There is semicolon at the end of while statement.	There is no semicolon at the end of while statement