

## UNIT-III

### CONDITIONAL STATEMENTS AND LOOPS

**Topics:** Conditional statements (if statement, if-else statement, ternary statement or ternary operator, nested if-else statement, switch statement), Difference between performance of if else and switch, Advantages of if else and switch over each other.

**Loops** — 'for' loops, 'while' loops, 'do while' loops, entry control and exit control, break and continue, nested loops. **8 Hrs**

#### Branching Statement:

Statement in C is any instruction written as code that is executable and is terminated by a semicolon.

Branching statement is one that alters the flow of the program based upon the outcome of the condition evaluated. There are various types of branching statements.

- Decision making
- Looping
- Unconditional branching

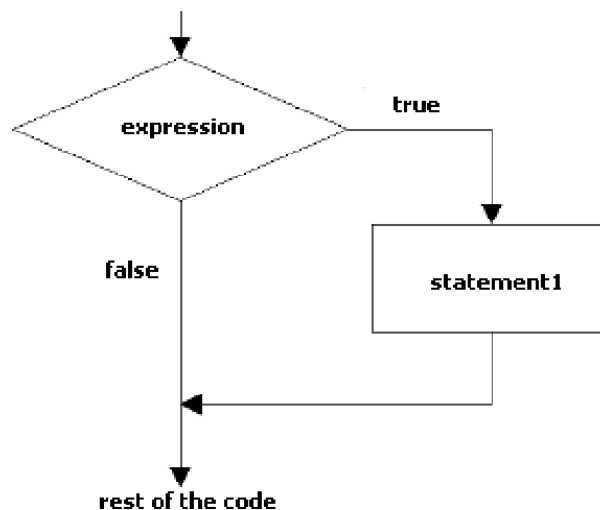
#### Decision making:

There are about five different types of decision-making statements:

- **if** —statement —also called **simple if**
- **if-else** statement
- **nested if**-statement
- **else-if** ladder
- **switch** statement

The flow diagram is shown below:

#### 1. **if** —statement:



The conditional expression/expression is evaluated to true or false. If the expression is evaluated to true, then statement1 is executed. If the expression is evaluated to false, then statement1 is skipped.

GeneralSyntax:

```
if( condition)
{    //Execute this block if condition is True Executable
    Statement(s);
}
Statement Below if;
```

**Conditional expression:** any expression that involves the operands and operators such as >,<,>=,<=,!= etc.

**Statement-1:** is any executable statement of C program.

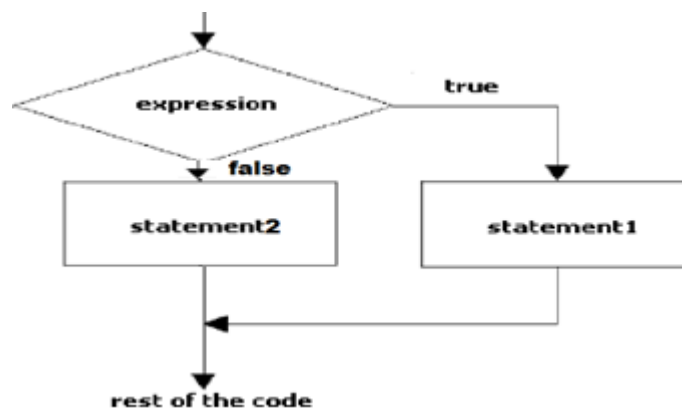
**Statement-x:** is any executable statement of C program that is always executed and is in the flow of the program.

Ex.

```
int x=10,y=-7, z=0;
if( x > y )
    z=x+y +100;
printf("The value of z is %d",z);
```

In the above program the conditional expression is  $x > y$ , when it's evaluates to a true value it executes the statement  $z = x + y + 100$ ; i.e the value of  $z$  becomes 103. However, the `printf` statement is always executed irrespective of the statements above it.

## 2. if —else statement:



The conditional expression/condition is evaluated to true or false. If the condition is evaluated to true, then statement1 is executed. If the condition is evaluated to false, then statement2 is executed.

General Syntax:

```
if( condition)
{
    //Execute this block if condition is True Executable
    Statement(s);
}
else
{
    //Execute this block if condition is False Executable
    Statement(s);
}
Statement Below if;
```

Ex.

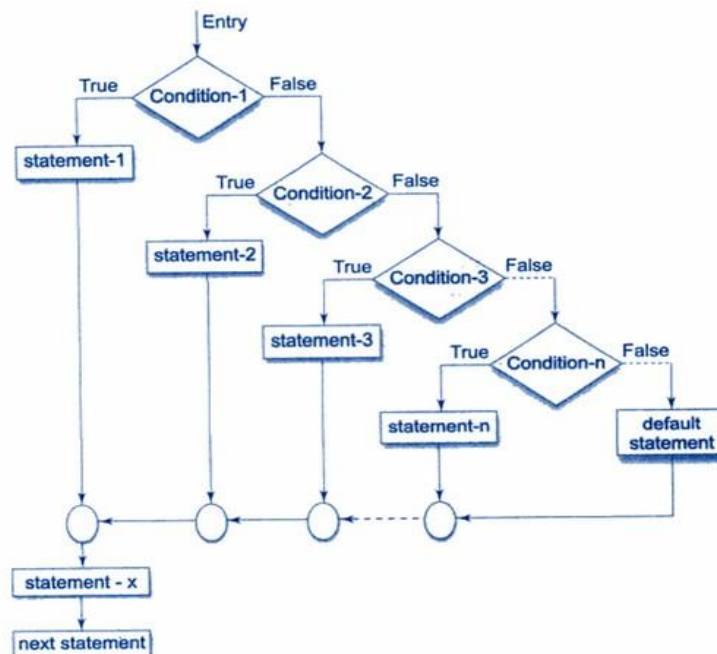
```
int x=10;
if( x%2 !=0)
    printf("%d is odd number", x);
else
    printf("%d is even number", x);
```

In the above program the conditional expression  $x \% 2 \neq 0$  is evaluated by dividing the value of  $x$  by 2 and then remainder is checked, if the remainder is not a zero value it results into true value, then it executes the printf statement by display the message that “10 is odd number” otherwise it displays the message “10 is a even number”.

### 3. else-if ladder:

else-if ladder is a set of cascaded if—else statements written one following another.

#### Flow Diagram:



General Syntax:

```

    if( condition-1)
        Statement(s);
    else if( condition-2)
        Statement(s);
    else if( condition-3)
        Statement(s);
        :
        :
    else if( condition-n)
        Statement(s)-n;
    else
        default-Statement(s);

    Statement Below if;

```

**Example:**

Program to illustrate the use of else if ladder statement.

```

#include <stdio.h>
void main ()
{
    int n;
    printf("Enter any integer:");
    scanf("%d", &n)
    if(n>0)
        printf ("Number is Positive");
    else
        if(n< 0)
            printf ("Number isNegative");
        else
            if(n== 0)
                printf ("Number isZero");
            else
                printf ("Invalid input");
}

```

4. **Nested if- statements:** if -statement within another if-statement is known as nested if-statements.

```
if(condition-1)
    if(condition-2)
        .....
        if(condition-n)
            statement-n;
```

**Example: To find biggest of three numbers.**

```
if(x>y)
    if(x>z)
        printf("x is biggest");
    else
        printf("z is biggest");
else
    if(y>z)
        printf("y is biggest");
    else
        printf("z is biggest");
```

## 5. THE switch STATEMENT

- This is basically a “multi-way” decision statement.
- This is used when we must choose among many alternatives.

**General Form:**

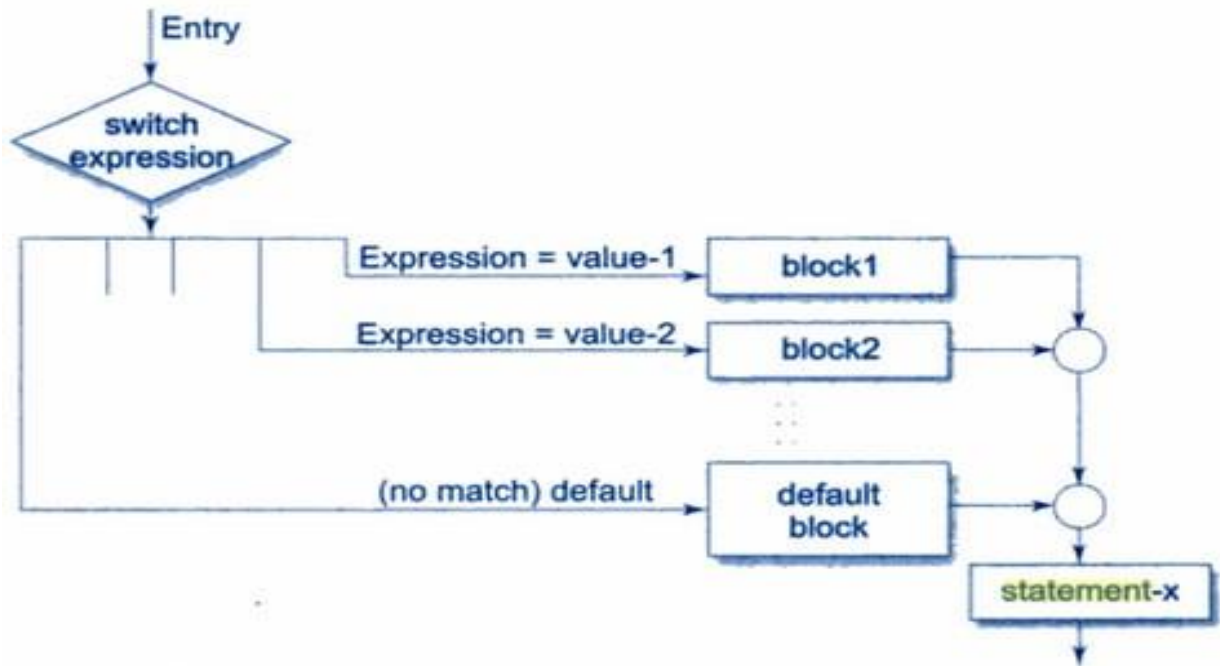
```
switch(Expression)
{
    case value-1: Statement-1;
        .
        .
        Statement-M; break;
    case value-2: Statement-1;
        .
        .
        Statement-M; break;
    .
    .
    .
    case value-N: statement-1;
        .
        .
        Statement-M; break;
```

```

    default: }
statement-1;
.
.
Statement-M;

```

The syntax & flow diagram is shown below:



- Here, choice can be either any integer value or a character.
- Based on this integer value, the control is transferred to a particular case-value where necessary statements are executed.
- During execution, if break statement is encountered, then the control comes out of the switch block.
- If the value of the choice does not match with any of the case values (i.e. value1, value2, value3,...valueN) then control goes to default label.
- All case-values must be different.
- Example: Program to illustrate the use of switch statement.

/\* Program to check the typed character is vowel or not\*/

```

void main()
{
    char letter; // local variable definition
    printf("Type any capital letter between A to Z: \n");
    scanf("%c",&letter);
}

```

```
switch(grade)
{
case'A':printf("Youtyped avowel.\n");
        break;
case'E':printf("Youtyped avowel.\n");
        break;
case'I':printf("Youtyped avowel.\n");
        break;
case'O':printf("Youtyped avowel.\n");
        break;
case'U':printf("Youtyped avowel.\n");
        break;
default:
        printf("Youtypedaconsonant.\n");
        break;
}
return 0;
}
```

## goto and labels

```
#include<stdio.h>

void main()
{
    int x;
    char resp;
next: printf("\nEnter a Number:");
    scanf("%d", &x);
    if(x <0)
    {
        printf("\nEnter only positive numbers.\n");
        goto next;
    }
    printf("The square root of %d is %f.\n",x,sqrt(x));
    printf("Would you like to continue[Y/N]:");
    scanf("%c",&resp);
    if(resp=='Y' )
        goto next; printf("\nExiting the program.");
}
```

Output:

Enter a Number:-7

Enteronlypositivenumbers. Enter a Number:225

The square root of 225 is15.000000.

Would you like to continue[Y/N]:N

Exiting theprogram.



## BASIC CONCEPT OF LOOP

- Let's consider a situation when you want to write a message “Welcome to C language” several times.

Here is a simple C program to print a statement five times:

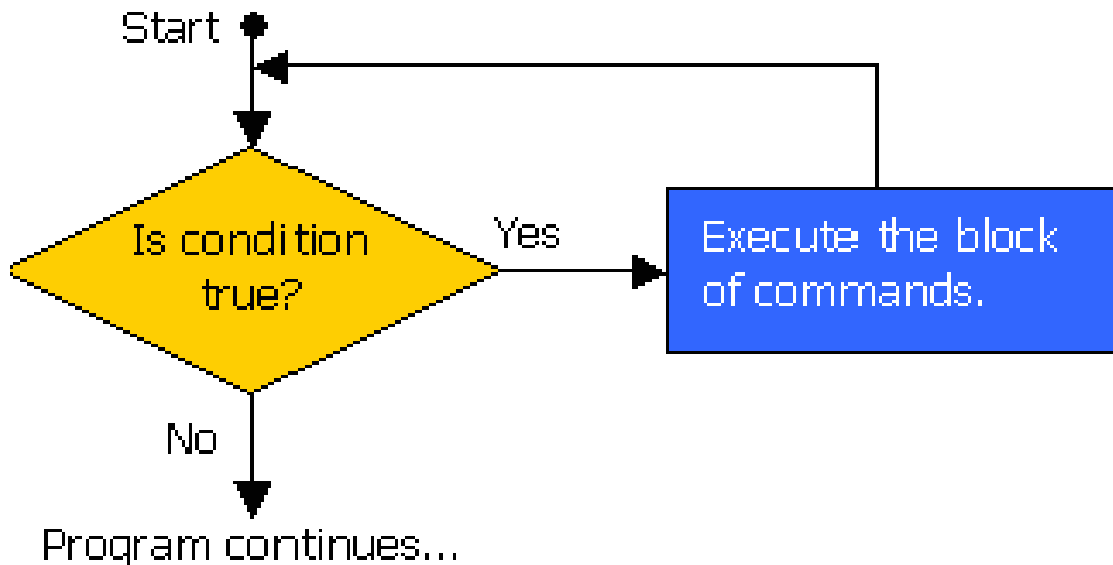
```
#include<stdio.h>

void main()
{
    printf( " Welcome to C language \n");
    printf( " Welcome to C language \n");
    printf( " Welcome to C language \n");
    printf( " Welcome to C language \n");
    printf( " Welcome to C language \n");
}
```

### Output:

Welcome to C language  
Welcome to C language  
Welcome to C language  
Welcome to C language  
Welcome to C language

- When the above program is executed, it produces the above result, but let's consider another situation when the same message needs to be printed thousand times.
- It is certainly not feasible to write printf() statement thousand times.
- C language provides a concept called loop, which helps in executing one or more statements up to desired number of times.
- Loops are used to execute one or more statements repeatedly.



There are three kinds of loops in C programming:

- 1) for loop
- 2) while loop
- 3) do while loop

### THE for LOOP

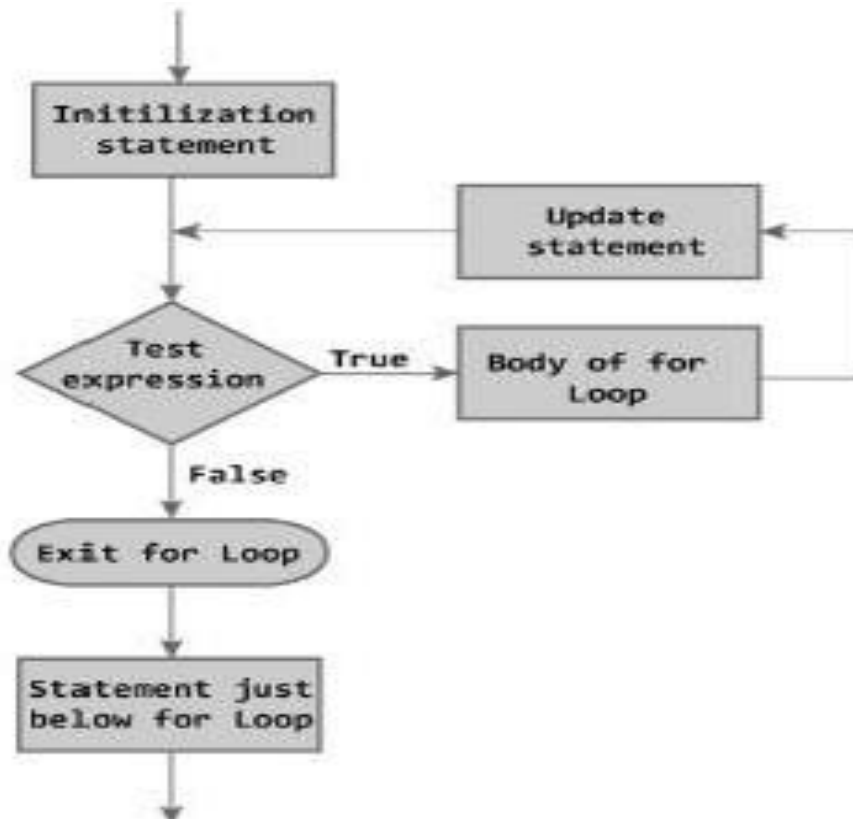
- A for loop statement can be used to execute a set of statements repeatedly as long as a given condition is true.

- The syntax is shown below:

```
for( initialization; condition; modification )  
{  
    Block of statement(s);  
}
```

- Firstly, initialization is done by assigning the first value to a variable & is executed only once.
- Then, condition is evaluated to true or false.
- If condition evaluates to false, the control comes out of the loop without executing the body of the loop.
- If condition evaluates to true, the body of the loop (i.e. Block of statement/statements) are executed.
- After executing the body of the loop, modification part is evaluated.
- Then again condition is checked for true or false. This cycle continues until expression condition becomes false.

- The flow diagram is shown below:



Example: Program to display a message five times using for statement.

```
#include<stdio.h>

void main()
{
    int i;
    for(i=1;i<=5;i++)
    {
        printf("Welcome to C language \n");
    }
}
```

Output:

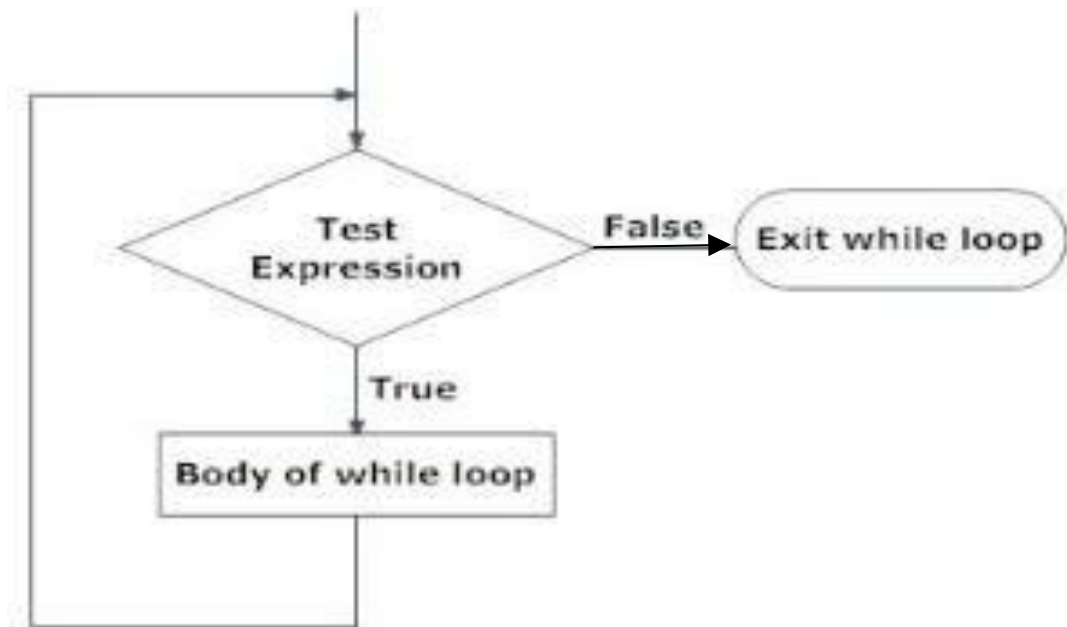
```
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
```

## **while LOOP**

- A while loop statement can be used to execute a set of statements repeatedly as long as a given condition is true.
- The syntax is shown below:

```
while(conditional expression)
{
    /* Block-statement(s); */
    /* Modify the loop control variable; */
}
```

- Firstly, the conditional expression is evaluated to true or false.
- If the conditional expression evaluates to false, the control comes out of the loop without executing the body of the loop.
- If the expression evaluates to true, the body of the loop (i.e. block- statement(s)) are executed.
- After executing the body of the loop, control goes back to the beginning of the while statement and condition is checked again for true or false. This cycle continues until expression becomes false.
- The flow diagram is shown below:



Example: Program to display a message five times using while statement.

```
#include<stdio.h> void main()
{
    int i=1;

    while(i<=5)
    {
        printf("Welcome to C language \n");
        i=i+1;
    }
}
```

Output:

```
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
```

### THE do while STATEMENT

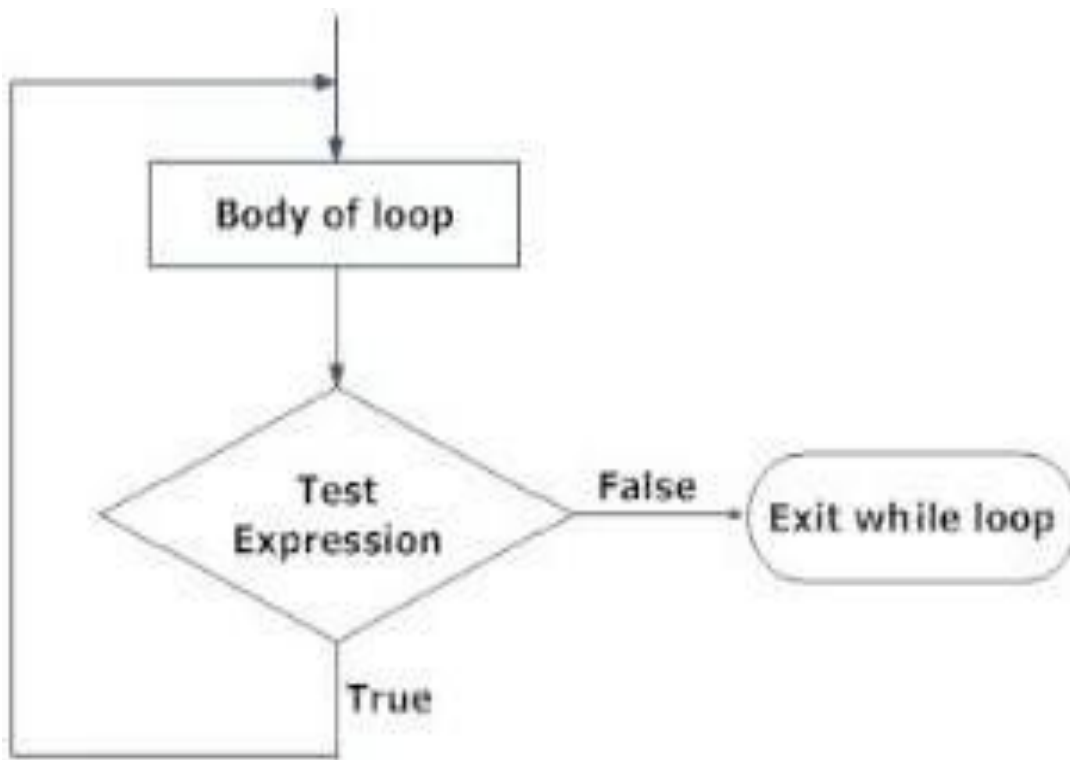
- do-while statement is used when we do not know exactly how many times a set of statements have to be executed repeatedly.
- The syntax is shown below:

```

do
{
    /* Block-statement(s); */
    /* Modify the loop control variable; */
}while(conditional expression);

```

- Firstly, the body of the loop is executed. i.e. the body of the loop is executed at least once.
- Then, the conditional expression is evaluated to true or false.
- If the expression evaluates to true, the body of the loop (i.e. block of statement/ statements) are executed.
- After executing the body of the loop, the expression is again evaluated to true or false. This cycle continues until conditional expression becomes false.
- The flow diagram is shown below:



Example: Program to display a message five times using do while statement.

```

#include<stdio.h>
void main()
{
    inti=1;
    do
    {
        printf("Welcome to C language \n");
        i=i+1;
    } while(i<=5);
}

```

Output:

Welcome to C language

Welcome to C language

Welcome to C language

Welcome to C language

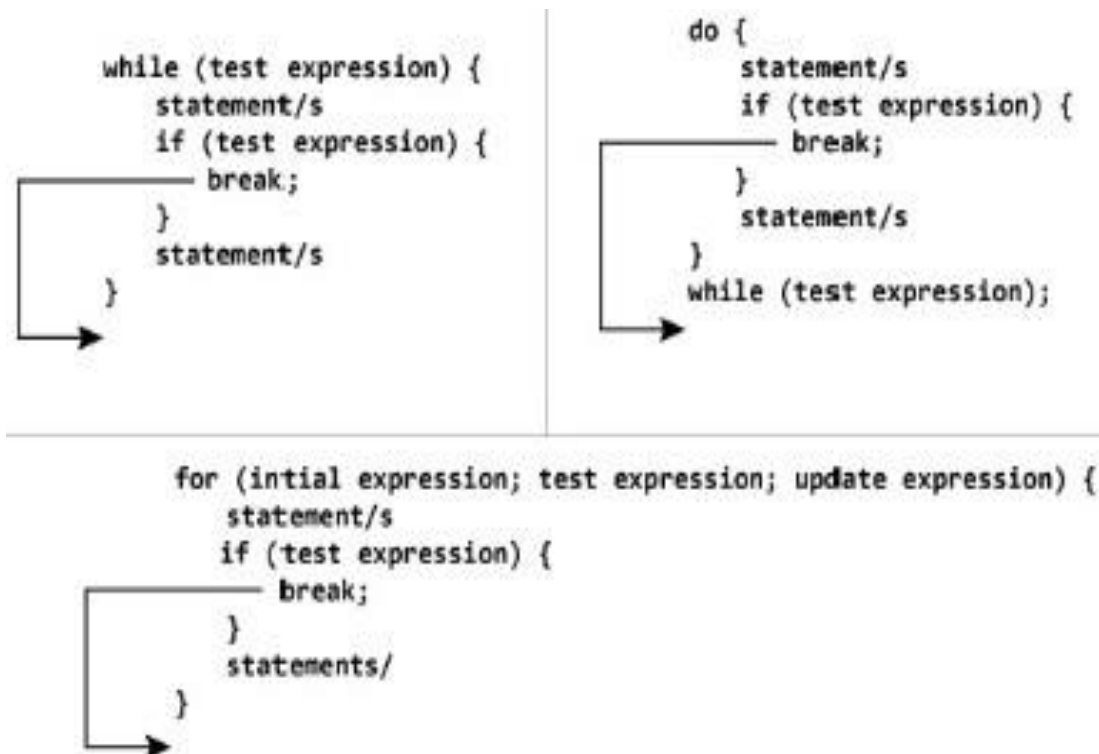
Welcome to C language

### Differences between while and do-while loop

While	do-while
Entry controlled loop	Exit controlled loop
Body of the loop is executed zero or more times	Body of the loop is executed at least one time
Condition is first checked and then body of the loop is executed	first body of the loop is executed and checks the condition
No semicolon at the end of the while construct	Semicolon at the end of the while construct

## THE break STATEMENT

- The break statement is jump statement which can be used in switch statement and loops.
- The break statement works as shown below:
  - 1) If break is executed in a switch block, the control comes out of the switch block and the statement following the switch block will be executed.
  - 2) If break is executed in a loop, the control comes out of the loop and the statement following the loop will be executed.
- The syntax is shown below:



Example-1: Program to find whether integer accepted from the user is prime or not.

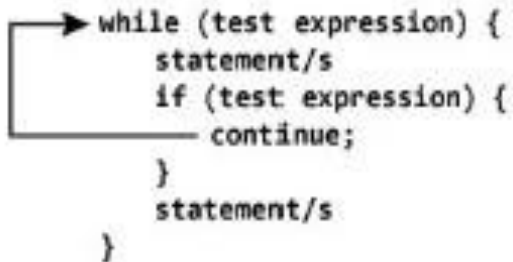
```
#include<stdio.h>  
void main()  
{  
    int i=1,num;  
    printf("Type an integer:");  
    scanf("%d",&num);  
    for(i=2; i <10; i ++)  
    {  
        if(num%i==0)  
            break;  
    }  
    if(i>=n)  
        printf("%d is a prime number",num);  
}
```

## THE continue STATEMENT

- During execution of a loop, it may be necessary to skip a part of the loop based on some condition.

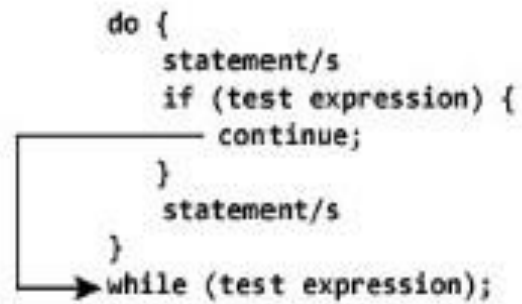
In such cases, we use continue statement.

- The continue statement is used only in the loop to terminate the current iteration.
- The syntax is shown below:



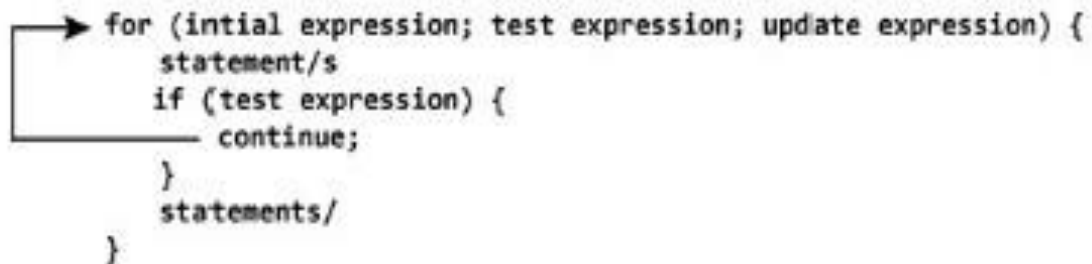
```
while (test expression) {  
    statement/s  
    if (test expression) {  
        continue;  
    }  
    statement/s  
}
```

The diagram shows a 'while' loop. An arrow points to the 'while' keyword. Inside the loop, there is an 'if' statement. An arrow points from the 'continue;' statement back to the 'while' condition, indicating that the current iteration is skipped and the loop starts over.



```
do {  
    statement/s  
    if (test expression) {  
        continue;  
    }  
    statement/s  
} while (test expression);
```

The diagram shows a 'do-while' loop. An arrow points to the 'do' keyword. Inside the 'do' block, there is an 'if' statement. An arrow points from the 'continue;' statement back to the 'while' condition, indicating that the current iteration is skipped and the loop starts over.



```
for (initial expression; test expression; update expression) {  
    statement/s  
    if (test expression) {  
        continue;  
    }  
    statements/  
}
```

The diagram shows a 'for' loop. An arrow points to the 'for' keyword. Inside the loop, there is an 'if' statement. An arrow points from the 'continue;' statement back to the 'for' condition, indicating that the current iteration is skipped and the loop starts over.



Example: Program to read and add only positive numbers using **continue** statement.

```
#include<stdio.h>
void main()
{
    int i=1, num, sum=0;

    for(i=0; i <5; i ++ )
    {
        printf(" Enter aninteger:");
        scanf( "%d", &num);
        if(num <0)
        {
            printf("you have entered a negative number \n");
            continue ; // skip the remaining part of loop
        }
        sum=sum+num;
    }
    printf("The sum of the Positive Integers Entered = %d", sum);
}
```

Output:

Enter an integer: 5

Enter an integer:11

Enter an integer: -33

You have entered a negative number

Enter an integer:100

Enter an integer:-99

You have entered a negative number

The sum of the positive integers entered = 116

## **Nested Loops**

- Loops can be placed inside other loops
- The break/continue statement applies to the innermost enclosing while or for statement

### **Syntax:**

The syntax for a nested for loop statement in C is as follows –

```
for (initialization; condition; increment ) {

    for (initialization; condition; increment ) {
        statement(s);
    }
    statement(s);
}
```

```
*
* *
* * *
* * * *
* * * * *
```

```
#include <stdio.h>
int main() {
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    for (i = 1; i <= rows; ++i) {
        for (j = 1; j <= i; ++j) {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
#include <stdio.h>
int main() {
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    for (i = 1; i <= rows; ++i) {
        for (j = 1; j <= i; ++j) {
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

**PROGRAM 1: Print Numbers from 1 to N**

```
// Use of for loop
#include <stdio.h>

int main()
{
    int i = 1;
    int N;

    printf("Enter the value of the Last Term \n") ;
    scanf("%d", &N);

    for ( i = 1 ; i <= N ; i++) // Show difference of < and <=
        printf ("%d \n", i);

    return 0;
}
```

**PROGRAM 2: Sum of Natural Numbers from 1 to Numbers**

```
// Use of for loop
#include <stdio.h>

int main()
{
    int i = 1 ;
    int      ;
    int sum = 0 ; // Important to Initialise to ZERO

    printf("Enter the value of the Last Term \n") ;
    scanf("%d", &N);

    for ( i = 1 ; i <= N ; i++) // Show difference of < and <=
        sum = sum + i ;

    printf("Sum of series from 1 to %d is %d \n",N, sum);

    return 0;
}
```

### **PROGRAM 3: Sum of First N Odd Number Terms FOR LOOP example**

```
#include <stdio.h>
```

```
int main()
{
    int i = 1 ;
    int sum = 0 ;
    int N ;
    int count = 0 ;

    printf("Enter the number of ODD terms you wish to find Sum for\n");
    scanf("%d", &N);

    for ( i = 1 ; count < N ; count++) // Why count is less than N not <= ?
    {
        sum = sum + i ;
        i = i + 2 ;
    } // For clarity written here

    printf ("Sum of the first %d Odd Number Series is %d\n", N, sum);

    return 0;
}
```

### **PROGRAM 4: Factorial of a Number WHILE LOOP**

```
#include <stdio.h>
```

```
int main ()
{
    int num ;
    long fact = 1;

    printf("Enter a Number \n");
    scanf("%d", &num);

    while ( num > 1 )
    {
        fact = fact * num ;
        num = num - 1 ;
    }

    printf("Factorial is %ld \n", fact);
}
```

**PROGRAM 5: Sum of Series while Term greater than 0.001 WHILE LOOP**

**// Sum of Series  $S = 1 + 1/3 + 1/5 + 1/7... 1/N$**

```
#include <stdio.h>
int main ()
{
    int i = 1 ;
    float sum = 0.0 ;
    float term = 1.0 ;

    while (term > 0.001)
    {
        sum = sum + term ;
        i = i + 2 ;
        term = 1.0/i ;
    }

    printf("Sum of Series is %6.2f \n", sum);
}
```

**PROGRAM 6: Sum of the Series  $1 + 1/2 + 1/3 + 1/4 + 1/5...1/N$**

```
#include <stdio.h>
int main()
{
    int i = 1;
    float sum = 0;
    int N;
    printf("Enter the value of the Series last Term\n");
    scanf("%d", &N);

    for ( i = 1 ; i <= N ; i++)
        sum = sum + 1.0 / i ;    // Why 1.0 not 1 ??
    printf ("Sum of the Series is %6.2f\n", sum);

    return 0;
}
```

**PROGRAM 7: Printing Count of Positive and Negative**

**// On entering Zero Do While Loop**

```
#include <stdio.h>
int main ()
{
    int num ;
    int PosCount = 0 ;
    int NegCount = 0 ;
```

```
printf("Enter a Series of Numbers and Zero to Quit \n");
do
{
    scanf("%d", &num );
    if ( num > 0 )
        PosCount += 1 ;
    else if ( num < 0 )
        NegCount += 1 ;
}while ( num != 0 );

printf("Pos Count =%d and Neg Count = %d\n", PosCount, NegCount);
}
```