

UNIT-3

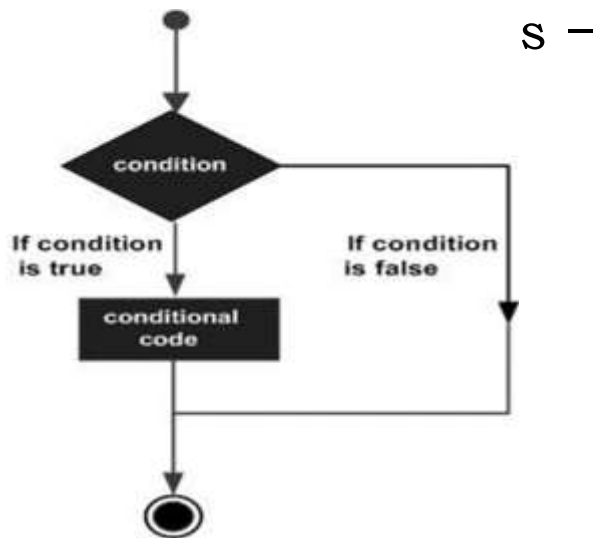
CONTROL STRUCTURES

Decision Making :

Decision making structures require that the programmer specifies 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.

Decision-making statements are the statements that are used to verify a given condition and decide whether a block of statements gets executed or not based on the condition result.

Show below is the general form of a typical decision making structure found in most of the p:



C programming language assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value.

C programming language provides the following types of decision making statements.

if statement in c

In c, if statement is used to make decisions based on a condition. The if statement verifies the given condition and decides whether a block of statements are executed or not based on the condition result. In c, if statement is classified into four types as follows...

1.Simple if statement

2.if-else statement

3.Nested if statement

4.if-else-if statement (if-else ladder)

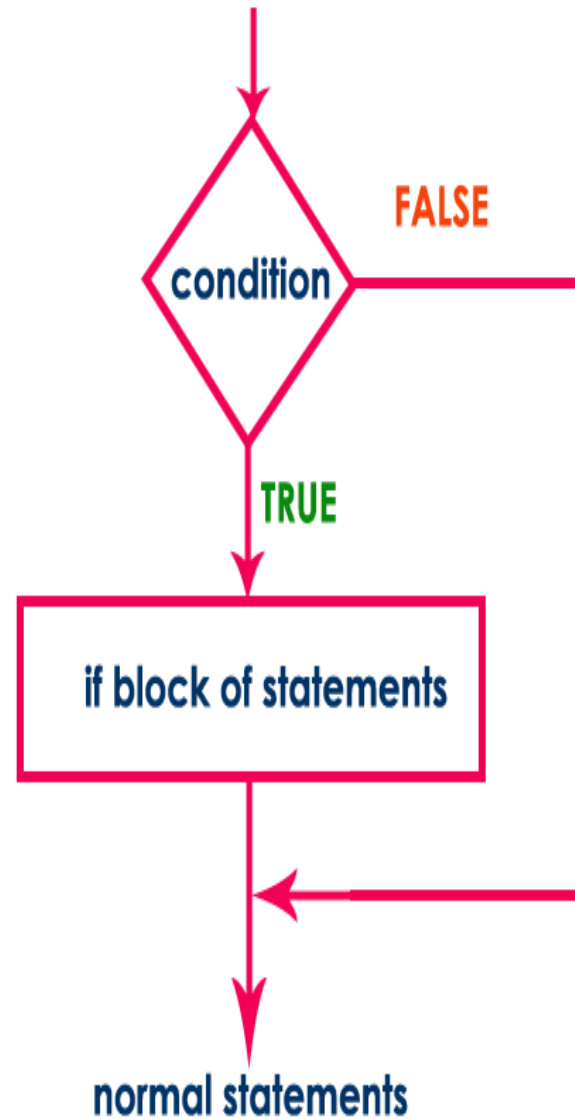
Simple if statement

Simple if statement is used to verify the given condition and executes the block of statements based on the condition result. The simple if statement evaluates specified condition. If it is TRUE, it executes the next statement or block of statements. If the condition is FALSE, it skips the execution of the next statement or block of statements. The general syntax and execution flow of the simple if statement is as follows.

Syntax

```
if ( condition )  
{  
    ...  
    block of statements;  
    ...  
}
```

Execution flow diagram



Simple if statement is used when we have only one option that is executed or skipped based on a condition.

| Test whether given number is divisible by 5.

Example Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n ;
    clrscr() ;
    printf("Enter any integer number: ") ;
    scanf("%d", &n);
    if ( n%5 == 0 )
    {
        printf("Given number is divisible by 5\n") ;
    }
    printf("statement does not belong to if!!!") ;
}
```

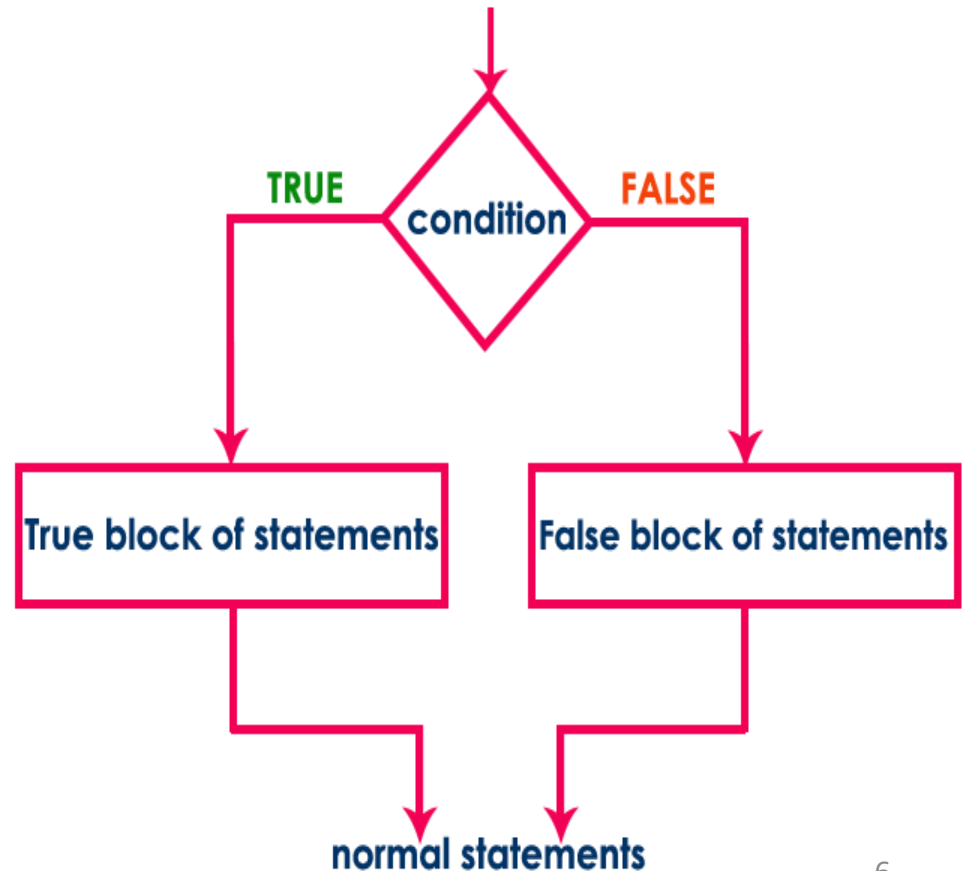
if-else statement

The if-else statement is used to verify the given condition and executes only one out of the two blocks of statements based on the condition result. The if-else statement evaluates the specified condition. If it is TRUE, it executes a block of statements (True block). If the condition is FALSE, it executes another block of statements (False block). The general syntax and execution flow of the if-else statement is as follows.

Syntax

```
if ( condition )  
{  
    ...  
    True block of statements;  
    ...  
}  
else  
{  
    ...  
    False block of statements;  
    ...  
}
```

Execution flow diagram



The if-else statement is used when we have two options and only one option has to be executed based on a condition result (TRUE or FALSE).

| Test whether given number is even or odd.

Example Program

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

void main(){
    int n ;
    clrscr() ;
    printf("Enter any integer number: ") ;
    scanf("%d", &n) ;
    if ( n%2 == 0 )
        printf("Given number is EVEN\n") ;
    else
        printf("Given number is ODD\n") ;
}
```

Nested if statement

Writing a if statement inside another if statement is called nested if statement. The general syntax of the nested if statement is as follows...

Syntax

```
if ( condition1 )  
{  
    if ( condition2 )  
    {  
        ....  
        True block of statements 1;  
    }  
    ....  
}  
else  
{  
    False block of condition1;  
}
```

The nested if statement can be defined using any combination of simple if & if-else statements.

Example Program

| Test whether given number is even or odd if it is below 100.

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

```
void main(){
    int n ;
    clrscr() ;
    printf("Enter any integer number: ") ;
    scanf("%d", &n) ;
    if ( n < 100 )
    {
        printf("Given number is below 100\n") ;
        if( n%2 == 0)
            printf("And it is EVEN") ;
        else
            printf("And it is ODD") ;
    }
    else
        printf("Given number is not below 100") ;
}
```

if-else-if statement (if-else ladder)

Writing a if statement inside else of an if statement is called if-else-if statement. The general syntax of the if-else-if statement is as follows...

Syntax

```
if ( condition1 )  
{  
    ....  
    True block of statements1;  
    ....  
}  
else if ( condition2 )  
{  
    False block of condition1;  
    &  
    True block of condition2  
}
```

The if-else-if statement can be defined using any combination of simple if & if-else statements.

Example Program

| Find the largest of three numbers.

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

void main(){
    int a, b, c ;
    clrscr() ;

    printf("Enter any three integer numbers: ") ;
    scanf("%d%d%d", &a, &b, &c) ;

    if( a>=b && a>=c)
        printf("%d is the largest number", a) ;

    else if (b>=a && b>=c)
        printf("%d is the largest number", b) ;

    else
        printf("%d is the largest number", c) ;
}
```

When we use a conditional control statement like if statement, the condition might be an expression evaluated to a numerical value, a variable or a direct numerical value.

If the expression value or direct value is zero the condition becomes FALSE otherwise becomes TRUE.

To understand more consider the following statements.

- **if(10)** - is TRUE
- **if(x)** - is FALSE if x value is zero otherwise TRUE
- **if(a+b)** - is FALSE if a+b value is zero otherwise TRUE
- **if(a = 99)** - is TRUE because a value is non-zero
- **if(10, 5, 0)** - is FALSE because it considers last value
- **if(0)** - is FALSE
- **if(a=10, b=15, c=0)** - is FALSE because last value is zero

'switch' statement in C

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

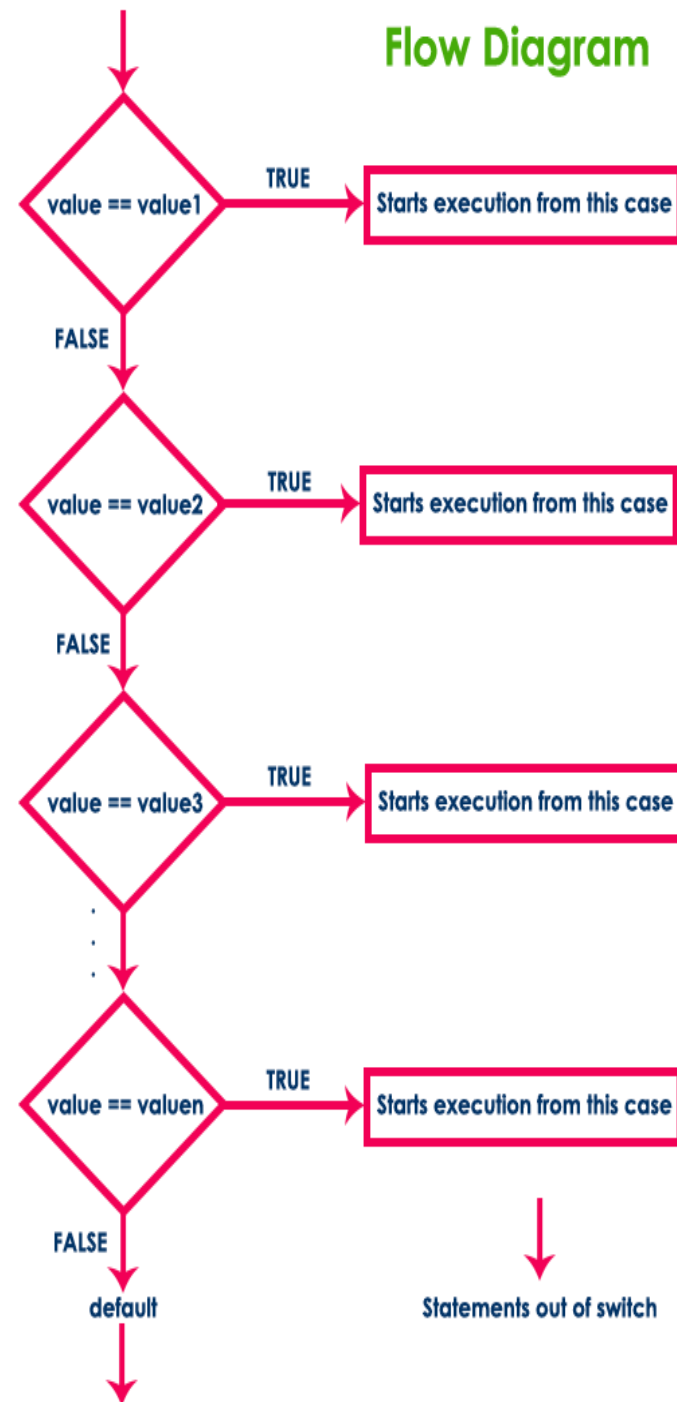
Consider a situation in which we have many options out of which we need to select only one option that is to be executed. Such kind of problems can be solved using **nested if** statement. But as the number of options increases, the complexity of the program also gets increased. This type of problem can be solved very easily using a **switch** statement. Using the switch statement, one can select only one option from more number of options very easily. In the switch statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches the value associated with an option, the execution starts from that option. In the switch statement, every option is defined as a **case**.

The switch statement has the following syntax and execution flow diagram.

Syntax

```
switch ( expression or value )  
{  
    case value1: set of statements;  
        ....  
    case value2: set of statements;  
        ....  
    case value3: set of statements;  
        ....  
    case value4: set of statements;  
        ....  
    case value5: set of statements;  
        ....  
    .  
    .  
    default: set of statements;  
}
```

Flow Diagram



The switch statement contains one or more cases and each case has a value associated with it. At first switch statement compares the first case value with the switchValue, if it gets matched the execution starts from the first case. If it doesn't match the switch statement compares the second case value with the switchValue and if it is matched the execution starts from the second case. This process continues until it finds a match. If no case value matches with the switchValue specified in the switch statement, then a special case called **default** is executed.

When a case value matches with the switchValue, the execution starts from that particular case. This execution flow continues with the next case statements also. To avoid this, we use the "**break**" statement at the end of each case. That means the **break** statement is used to terminate the switch statement. However, it is optional.

```

#include <stdio.h>
int main()
{
    char ch='b';
    switch (ch)
    {
        case 'd':
            printf("Case D ");
            break;
        case 'b':
            printf("Case B");
            break;
        case 'c':
            printf("Case C");
            break;
        case 'z':
            printf("Case Z ");
            break;
        default:
            printf("Default ");
    }
    return 0;
}

```

The following rules apply to a **switch** statement –

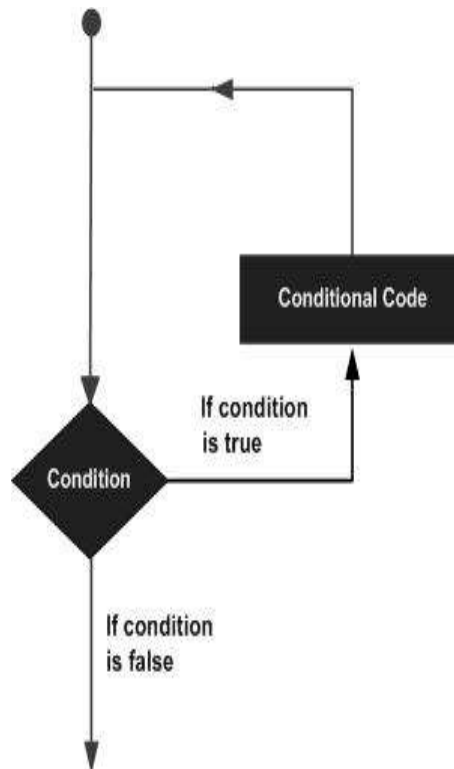
- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

C – Loops

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages –



C programming language provides the following types of loops to handle looping requirements.

Sr.No.	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop It is more like a while statement, except that it tests the condition at the end of the loop body.
4	nested loops You can use one or more loops inside any other while, for, or do..while loop.

while Loop

A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a while loop in C programming language is –

```
while(condition) {  
    statement(s);  
}
```

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

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

Example

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

```
int main ()  
{
```

```
    /* local variable definition */
```

```
    int a = 10;
```

```
    /* while loop execution */
```

```
    while( a < 20 )
```

```
    {
```

```
        printf("value of a: %d\n", a);
```

```
        a++;
```

```
    }
```

```
    return 0;
```

```
}
```

for loop in C

A for 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 of a for loop in C programming language is –

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

Here is the flow of control in a 'for' loop –

- The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- 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 the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- 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' loop terminates.

Example

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

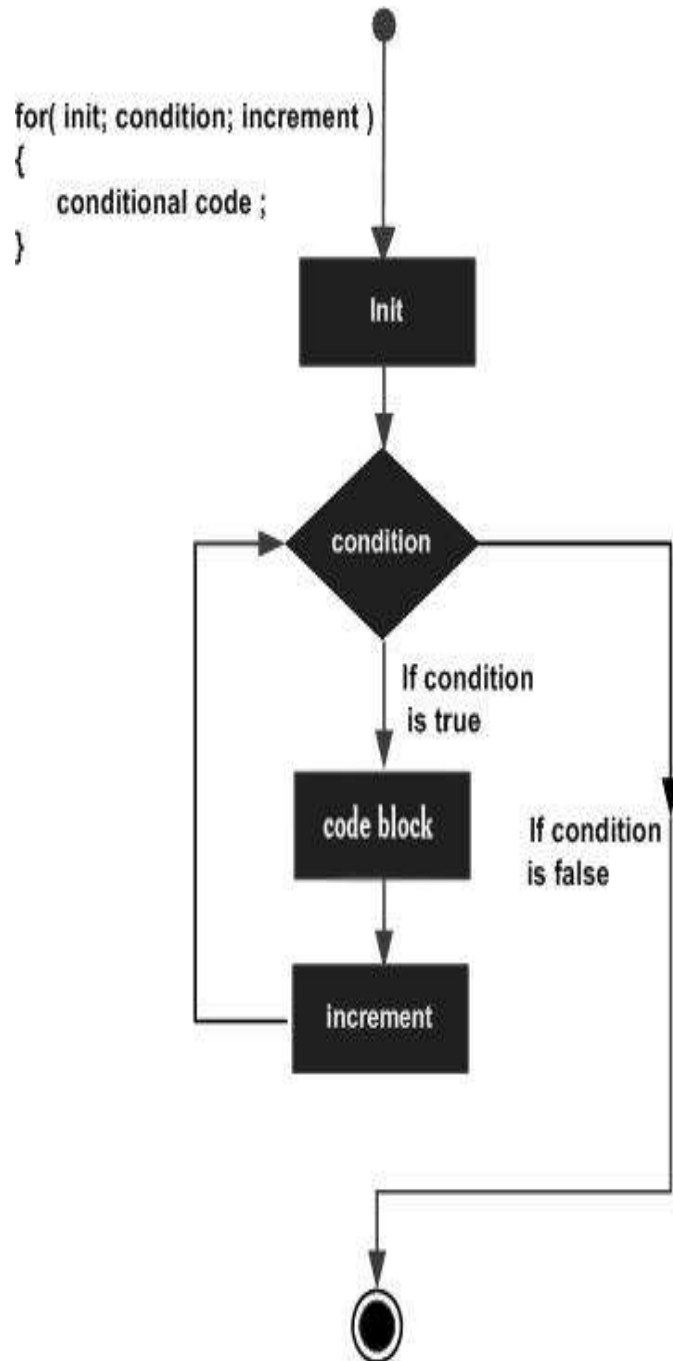
```
int main ()  
{
```

```
    int a;
```

```
    /* for loop execution */  
    for( a = 10; a < 20; a = a + 1 )  
    {  
        printf("value of a: %d\n", a);  
    }
```

```
    return 0;
```

```
}
```



do...while loop in C

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the do...while loop in C programming checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

The syntax of a do...while loop in C programming language is –

```
do {  
    statement(s);  
} while( condition );
```

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

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

Example

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

```
int main ()  
{
```

```
/* local variable definition */
```

```
int a = 10;
```

```
/* do loop execution */
```

```
do
```

```
{
```

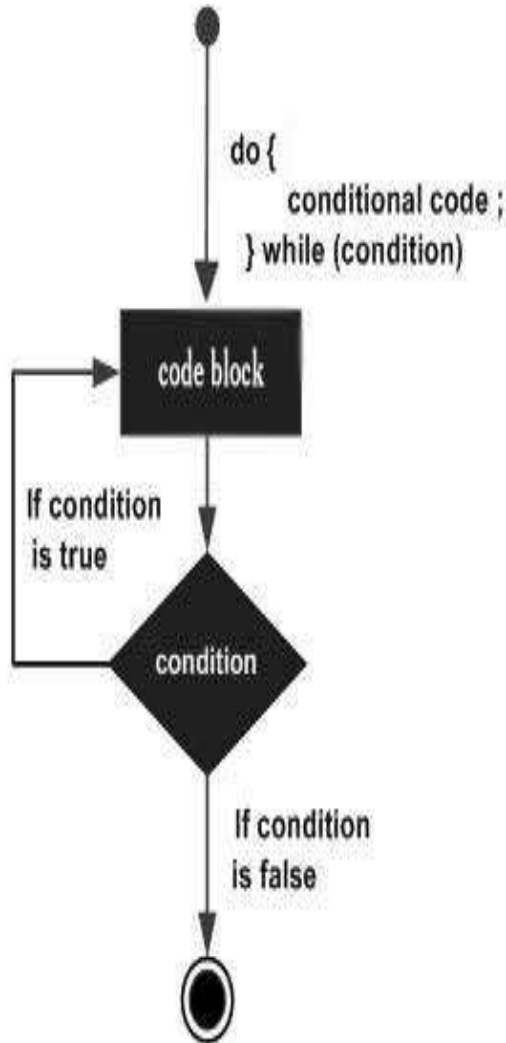
```
    printf("value of a: %d\n", a);
```

```
    a = a + 1;
```

```
}while( a < 20 );
```

```
return 0;
```

```
}
```



nested loops in C

C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept.

Syntax

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

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

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

```
while(condition) {  
  
    while(condition) {  
        statement(s);  
    }  
    statement(s);  
}
```

The syntax for a nested do...while loop statement in C programming language is as follows –

```
do {  
    statement(s);  
  
    do {  
        statement(s);  
    }while( condition );  
}  
}while( condition );
```

A final note on loop nesting is that you can put any type of loop inside any other type of loop. For example, a 'for' loop can be inside a 'while' loop or vice versa.

Example

The following program uses a nested for loop to find the prime numbers from 2 to 100 –

```
#include <stdio.h>
int main ()
{
    /* local variable definition */
    int i, j;
    for(i = 2; i<100; i++)
    {
        for(j = 2; j <= (i/j); j++)
        if(!(i%j))
            break;    // if factor found, not prime
        if(j > (i/j))
            printf("%d is prime\n", i);
    }
    return 0;
}
```

The Infinite Loop

A loop becomes an infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

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

```
int main () {
```

```
    for( ; ; ) {  
        printf("This loop will run forever.\n");  
    }
```

```
    return 0;  
}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the for(;;) construct to signify an infinite loop.

NOTE – You can terminate an infinite loop by pressing Ctrl + C keys.

UNIT – 3

COMPLETE