COMPUTER PROGRAMMING

Conditional and Iterative Statements

COMPUTER PROGRAMMING

Conditional Statements

Branching

Topics covered

- Conditional branching statements, iterative statements, nested loops, break and continue statements.

Branching

- C program may require that a logical test be carried out at some particular point within the program.

- One of several possible actions will then be carried out, depending on the output of the logical test.

This is known as *Branching*.

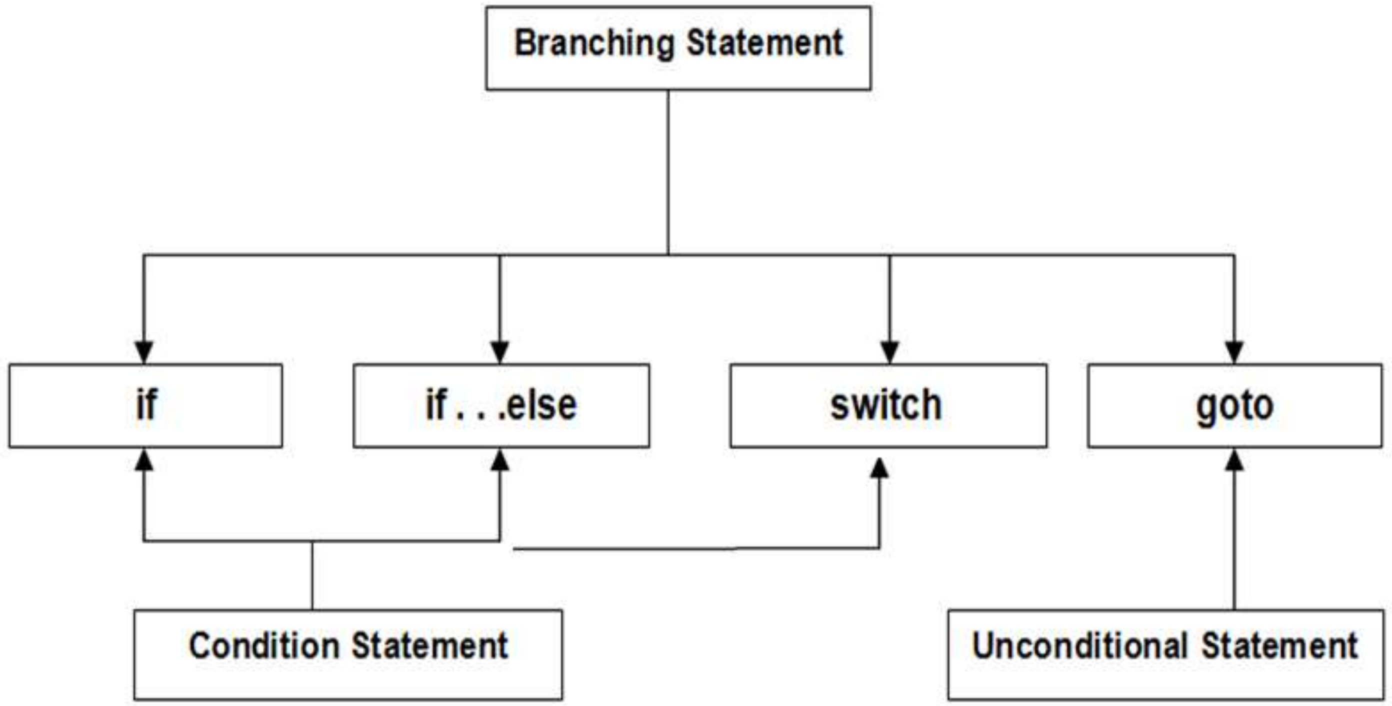
Branching

❖ There is also special kind of branching, called *Selection*, in which one group of statements is selected from several available groups.

DECISION MAKING

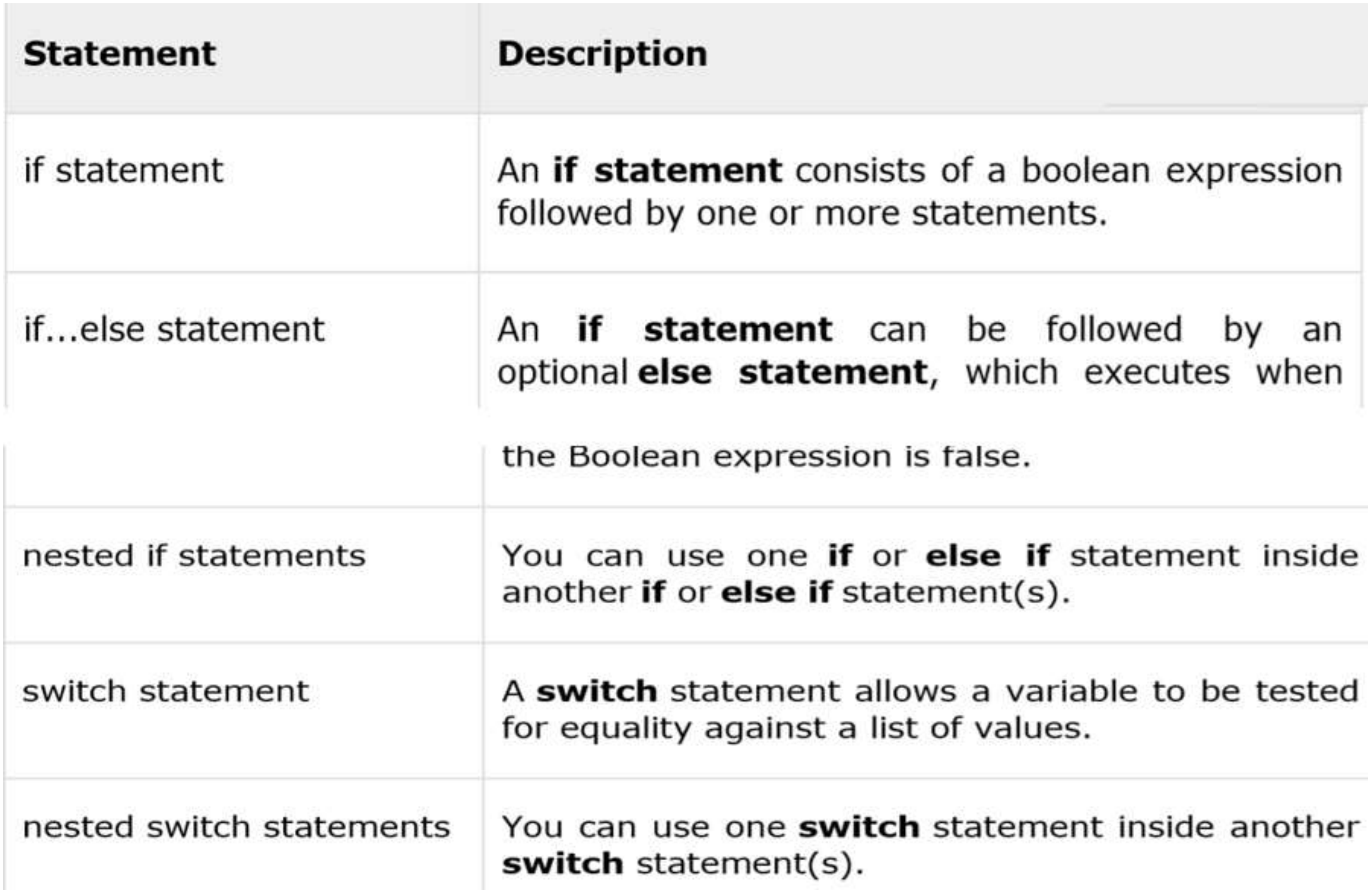
- Decision-making structures require that the programmer **specifies one or more conditions to be evaluated** or tested by the program,
- along with a statement or **statements to be executed if the condition is determined to be true**, and
- optionally, **other statements to be executed if the condition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

Branching Statements



All of these operations can be carried out using various control statements included in C

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



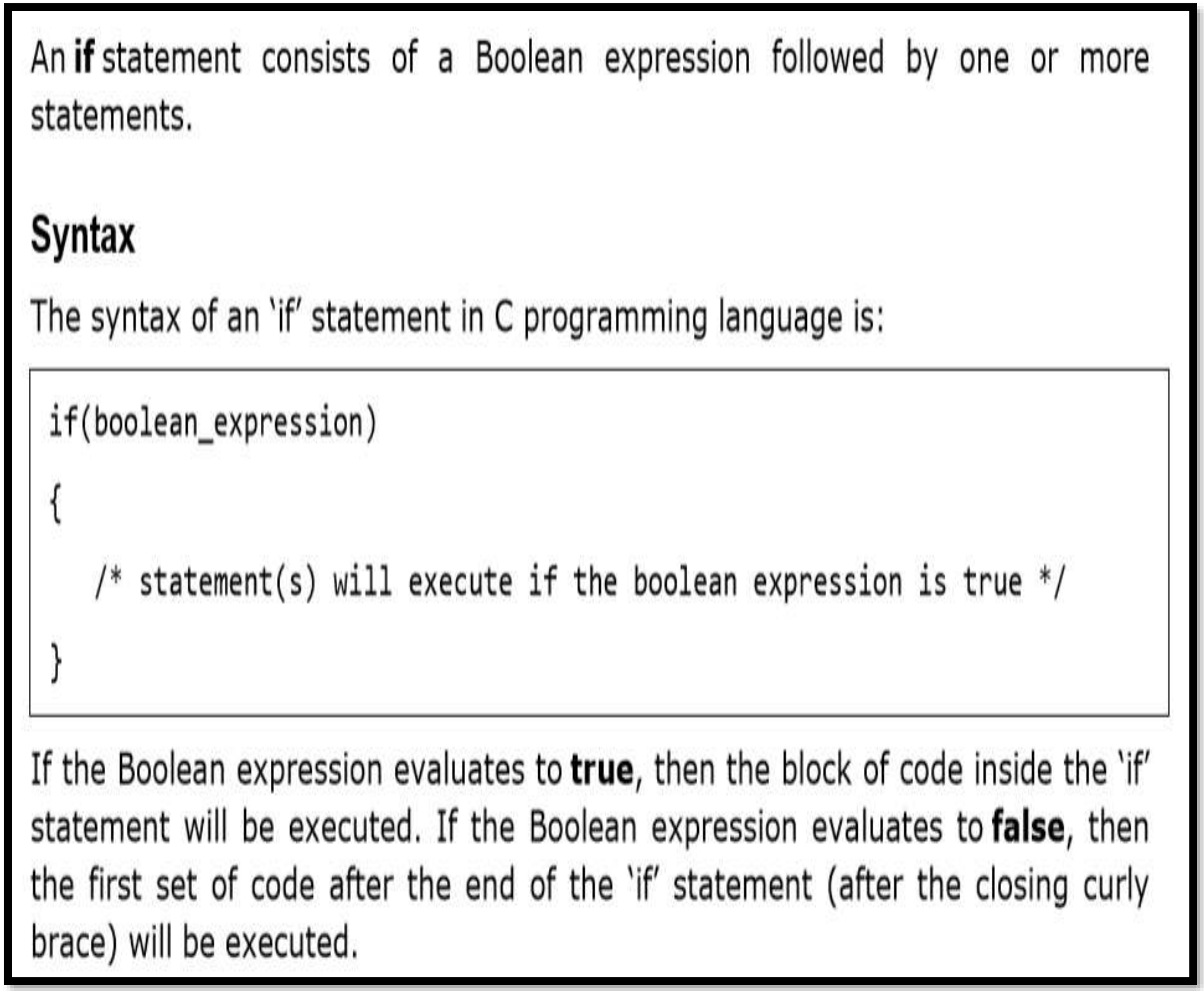
Statement	Description
if statement	An **if statement** consists of a boolean expression followed by one or more statements.
if...else statement	An **if statement** can be followed by an optional **else statement**, which executes when the Boolean expression is false.
nested if statements	You can use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

COMPUTER PROGRAMMING

Conditional Statements

if Statement

if Statements



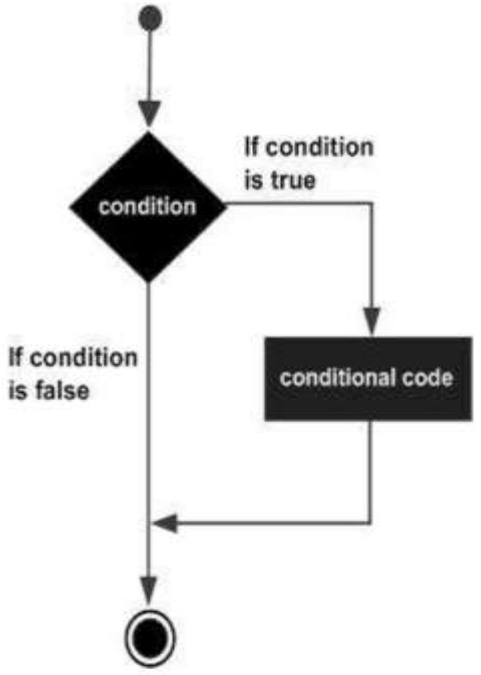
An **if** statement consists of a Boolean expression followed by one or more statements.

Syntax

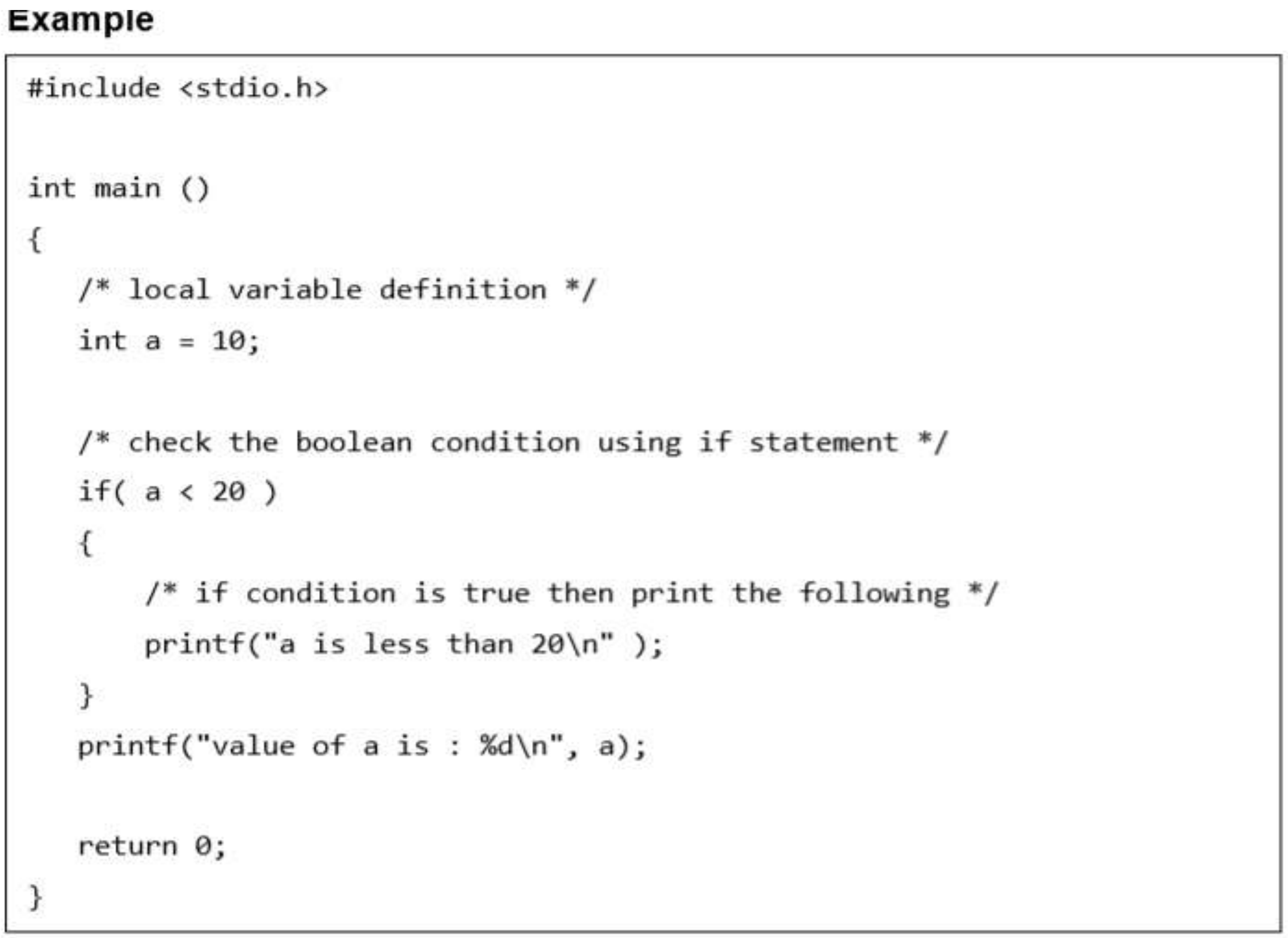
The syntax of an 'if' statement in C programming language is:

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
```

If the Boolean expression evaluates to **true**, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to **false**, then the first set of code after the end of the 'if' statement (after the closing curly brace) will be executed.



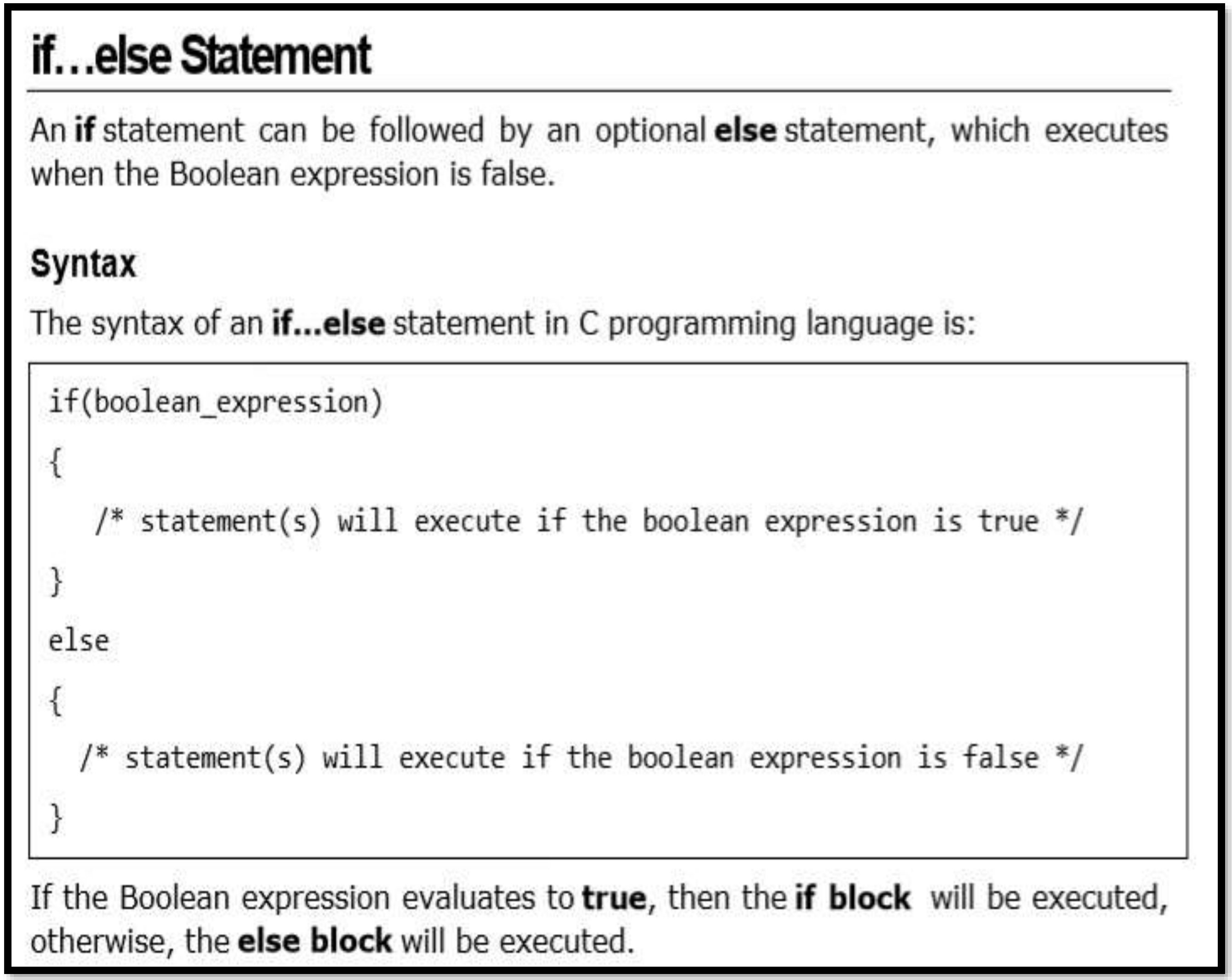
if Statements



Example

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* check the boolean condition using if statement */
   if( a < 20 )
   {
       /* if condition is true then print the following */
       printf("a is less than 20\n" );
   }
   printf("value of a is : %d\n", a);

   return 0;
}
```

If..else Statements



if...else Statement

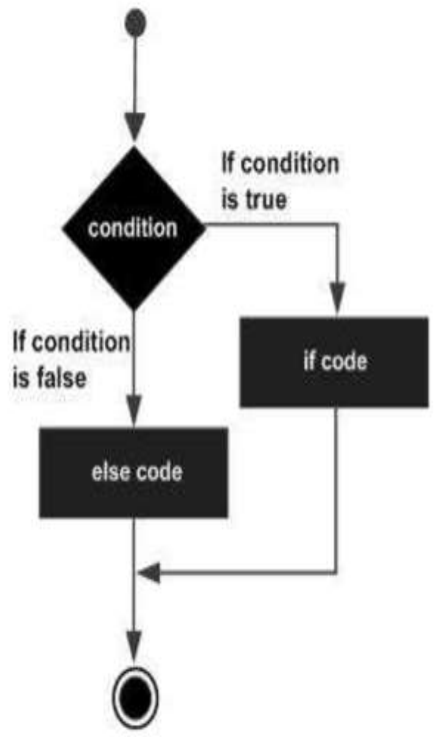
An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.

Syntax

The syntax of an **if...else** statement in C programming language is:

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
else
{
  /* statement(s) will execute if the boolean expression is false */
}
```

If the Boolean expression evaluates to **true**, then the **if block** will be executed, otherwise, the **else block** will be executed.



....example

```
                    // IF statements
main()
{
int a=10;
if(a<20)
  {
 printf("a<20\n");
  }
if(a>20)
  {
 printf("a>20\n");
  }
}
// {...} may be avoided for single statement under if-else
```

….example

// IF-ELSE statements

```
main()
{
int a=10;
if(a<20)
     {
 printf("a<20\n");
     }
else
      {
 printf("a>20\n");
     }
}
```

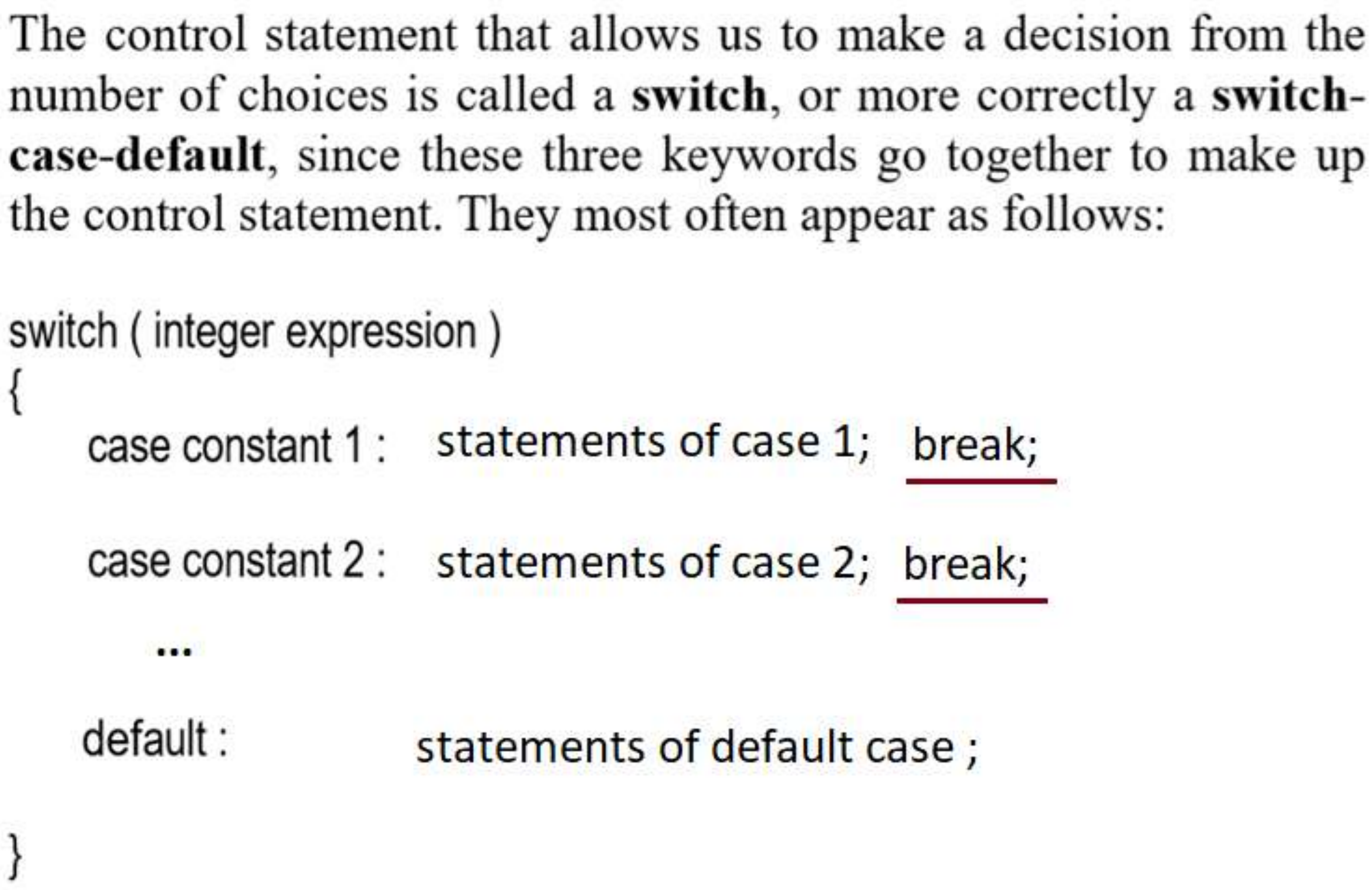
// {…} may be avoided for single statement under if-else

....example

```
        // Number is even or odd using if-else
#include <stdio.h>
int main(){
   int x;
   printf("enter the value of x:");
   scanf("%d", &x);
   if (x%2==0)
        printf("x is an even number\n");
   else
        printf("x is an odd number\n");
return 0;
}
```

Conditional Statements

switch Statement



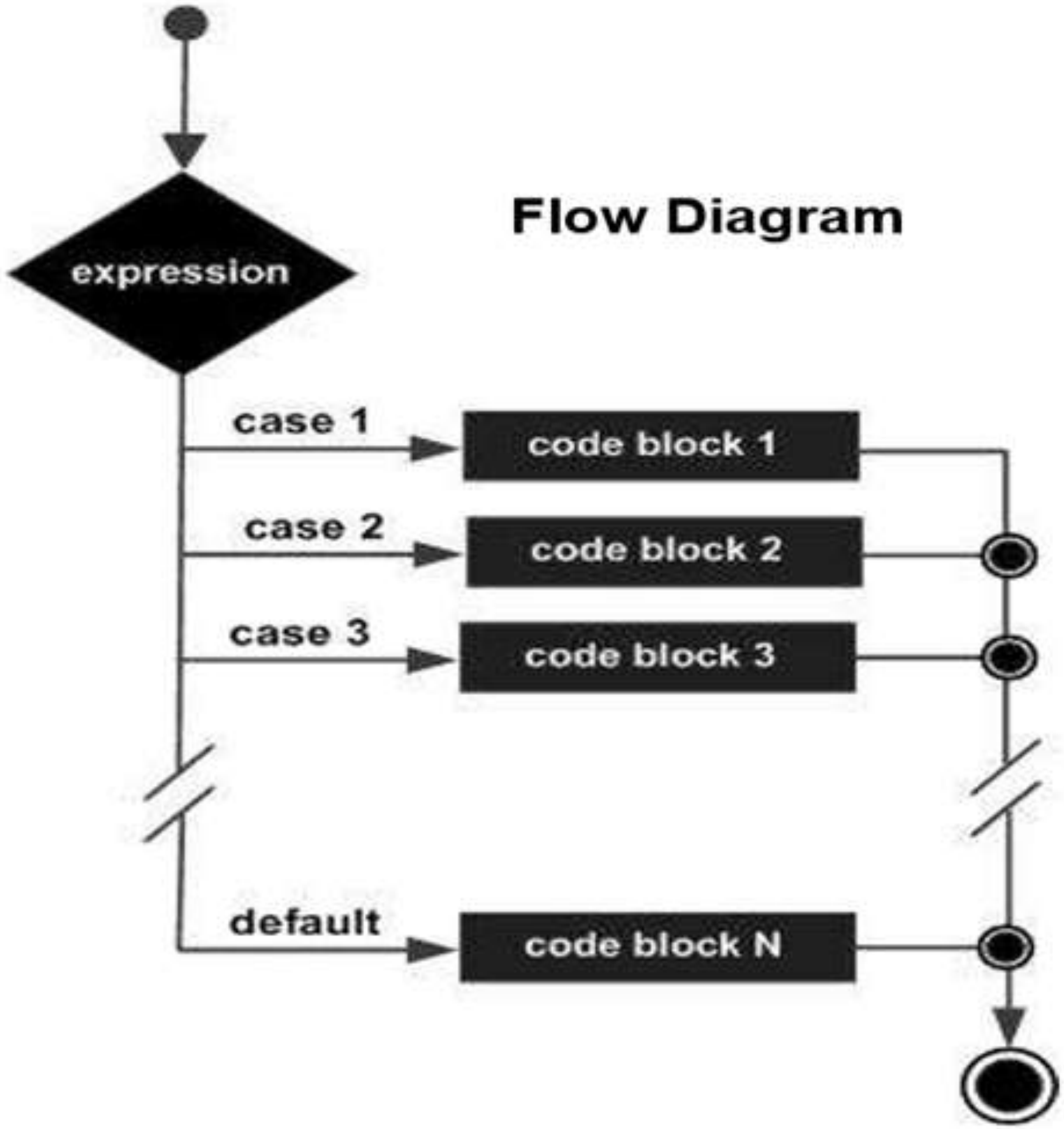
The control statement that allows us to make a decision from the number of choices is called a **switch**, or more correctly a **switch-case-default**, since these three keywords go together to make up the control statement. They most often appear as follows:

```
switch ( integer expression )
{
      case constant 1 :   statements of case 1;  break;

      case constant 2 :   statements of case 2;  break;

         ...

      default :           statements of default case ;

}
```

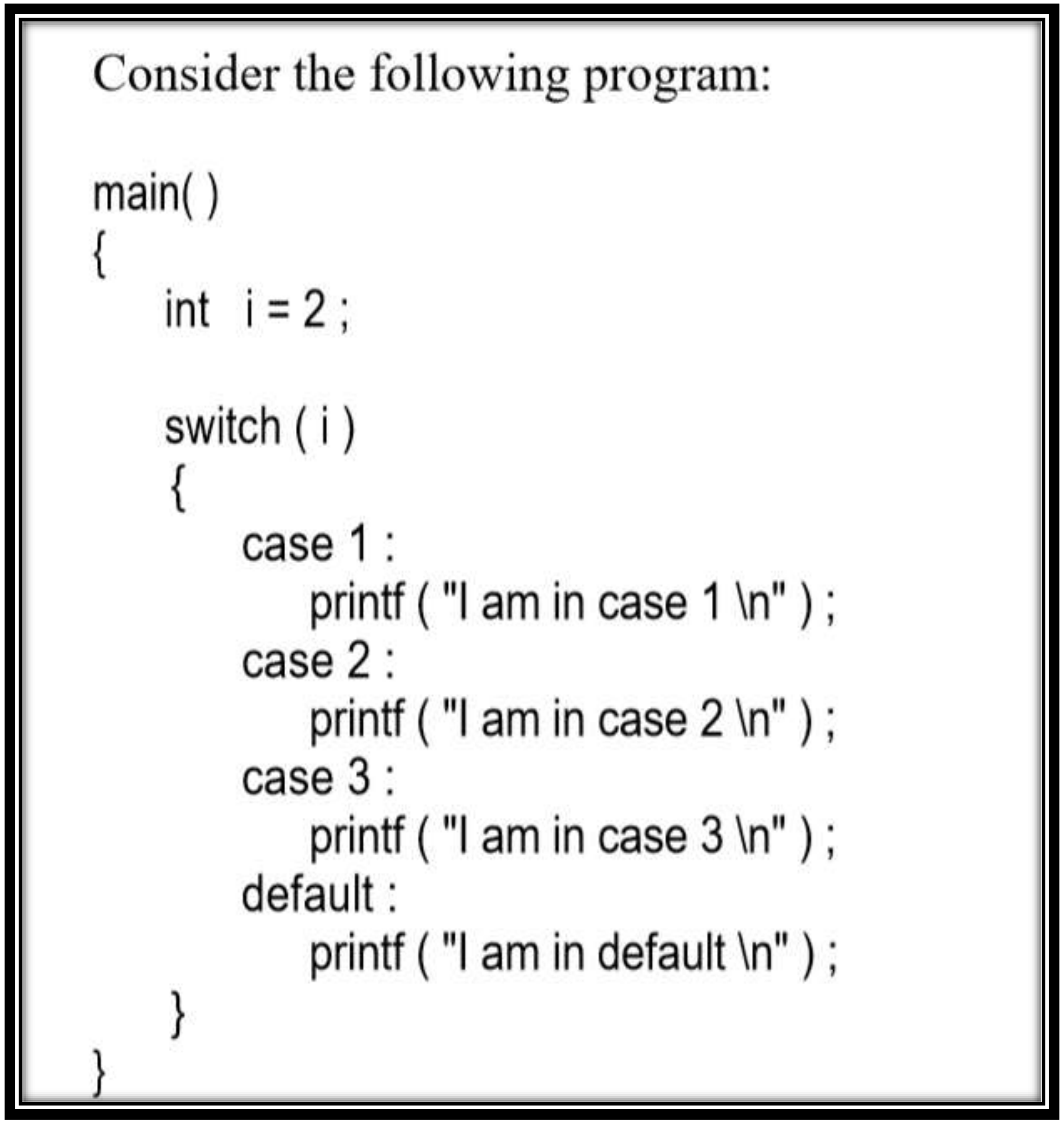
A switch allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

// The break; statement causes an immediate exit from the switch

Flow Diagram



switch Statement



Consider the following program:

```
main( )
{
    int  i = 2 ;

    switch ( i )
    {
        case 1 :
            printf ( "I am in case 1 \n" ) ;
        case 2 :
            printf ( "I am in case 2 \n" ) ;
        case 3 :
            printf ( "I am in case 3 \n" ) ;
        default :
            printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case 2

I am in case 3
I am in default
```

switch Statement

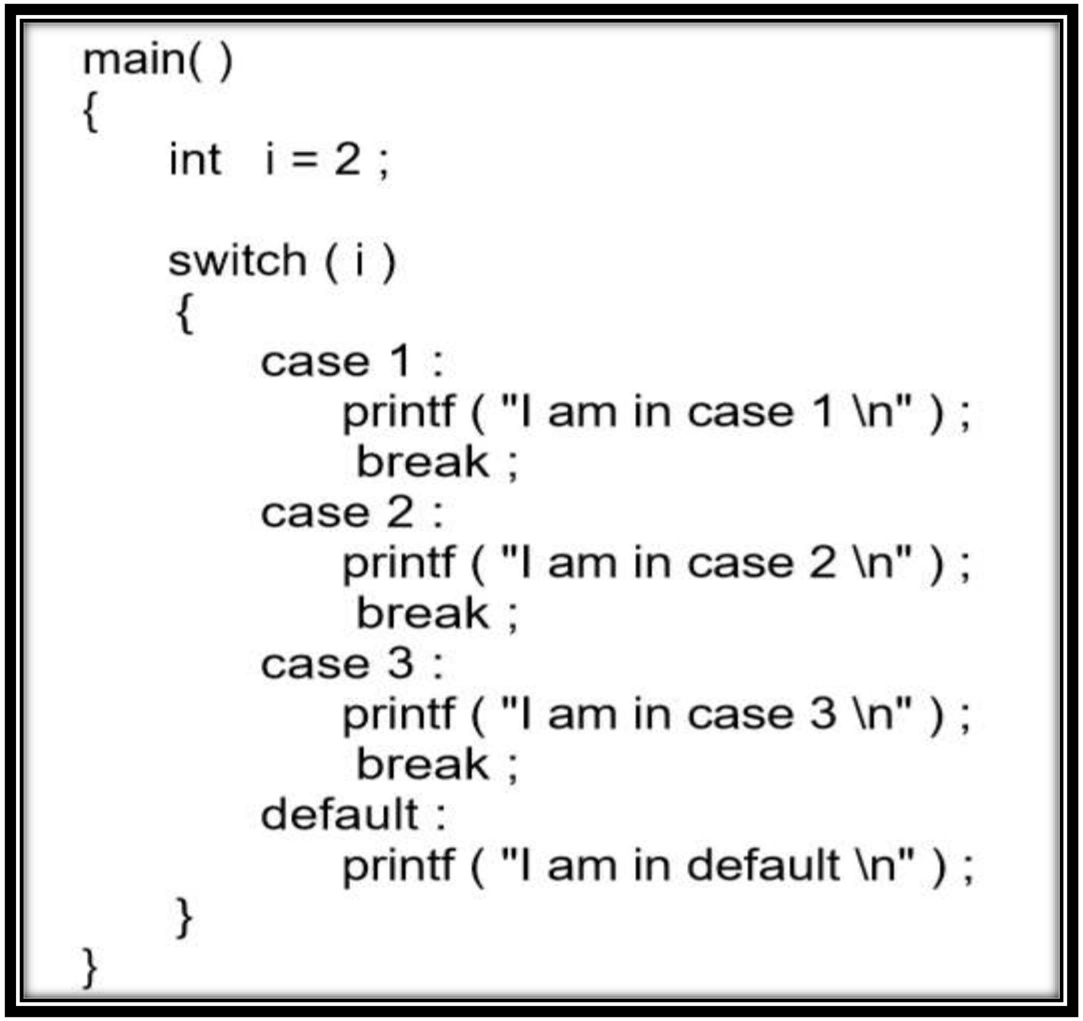
The output is definitely not what we expected! We didn’t expect the second and third line in the above output. The program prints case 2 and 3 and the default case. Well, yes. We said the **switch** executes the case where a match is found and all the subsequent **cases** and the **default** as well.

If you want that only case 2 should get executed, it is upto you to get out of the **switch** then and there by using a **break** statement. The following example shows how this is done. Note that there is no need for a **break** statement after the **default**, since the control comes out of the **switch** anyway.

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 may 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.

switch Statement



```
main( )
{
    int   i = 2 ;

    switch ( i )
    {
        case 1 :
            printf ( "I am in case 1 \n" ) ;
             break ;
        case 2 :
            printf ( "I am in case 2 \n" ) ;
             break ;
        case 3 :
            printf ( "I am in case 3 \n" ) ;
             break ;
        default :
            printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

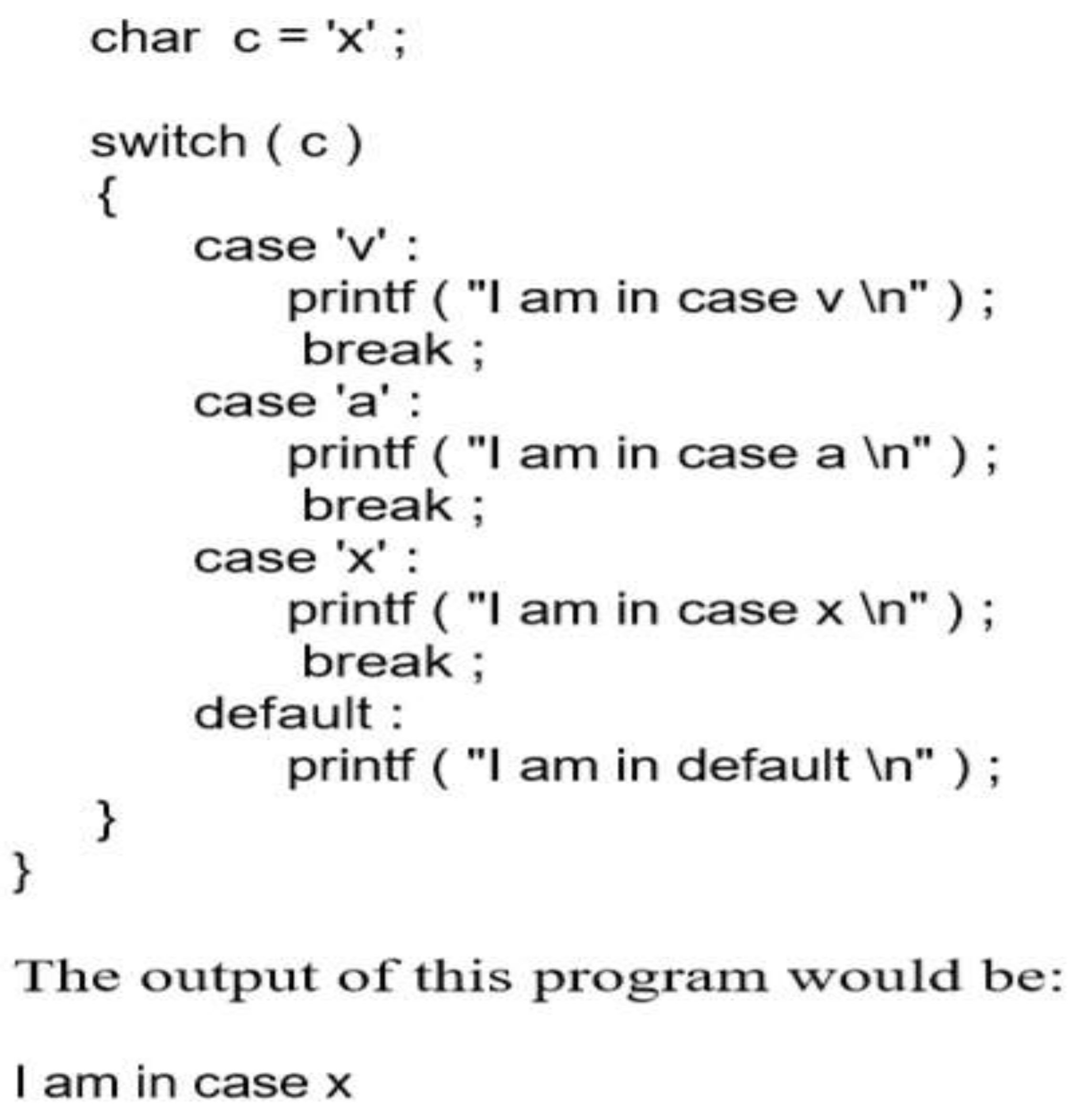
```
I am in case 2
```

switch Statement

```
main( )
{
    int  i = 22 ;

    switch ( i )
    {
         case 121 :
              printf ( "I am in case 121 \n" ) ;
               break ;
         case 7 :
              printf ( "I am in case 7 \n" ) ;
               break ;
         case 22 :
              printf ( "I am in case 22 \n" ) ;
               break ;
         default :
              printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

switch Statement

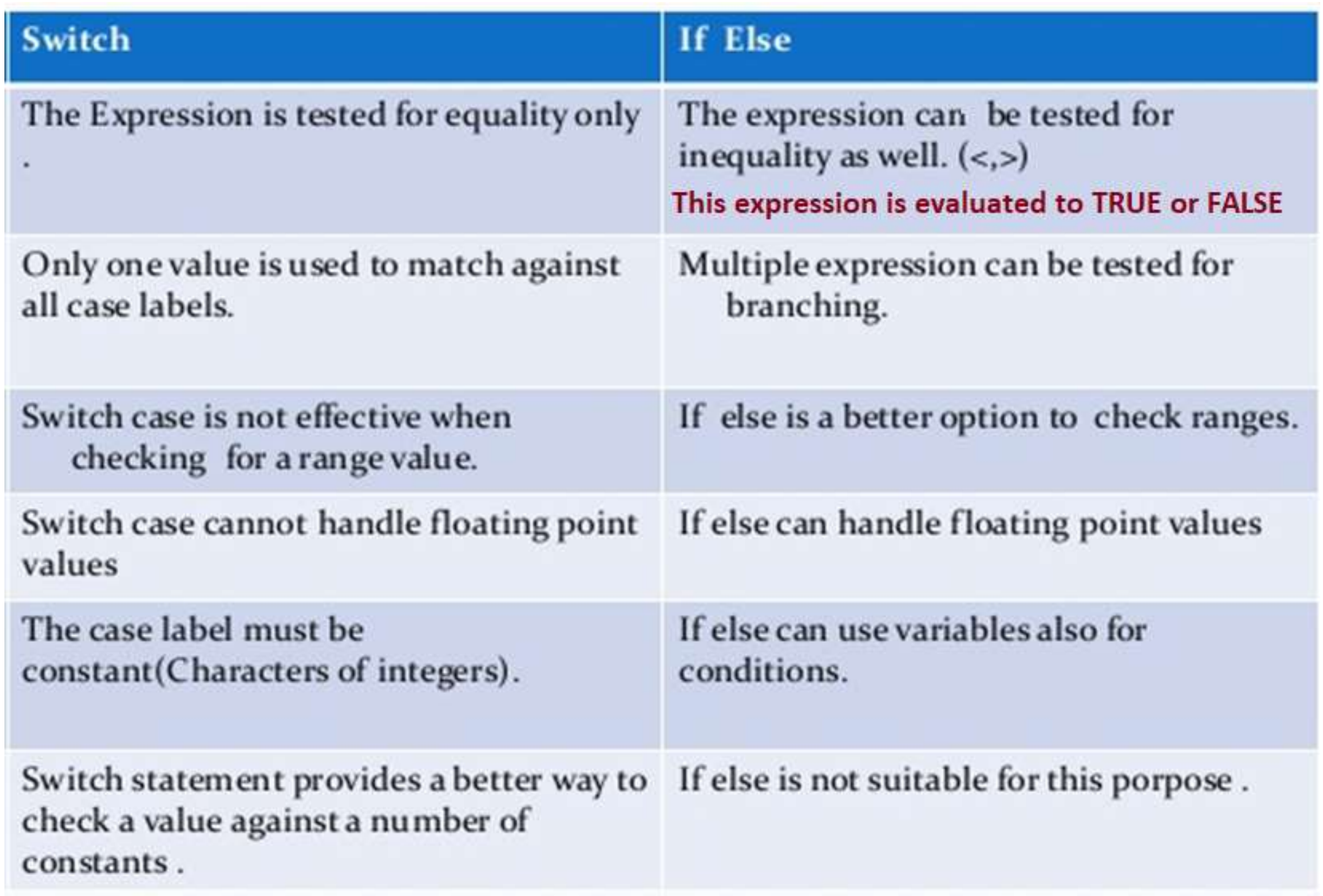


```
    char  c = 'x' ;

    switch ( c )
    {
         case 'v' :
              printf ( "I am in case v \n" ) ;
               break ;
         case 'a' :
              printf ( "I am in case a \n" ) ;
               break ;
         case 'x' :
              printf ( "I am in case x \n" ) ;
               break ;
         default :
              printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case x
```

switch vs if



Switch	If Else
The Expression is tested for equality only .	The expression can be tested for inequality as well. (<,>) This expression is evaluated to TRUE or FALSE
Only one value is used to match against all case labels.	Multiple expression can be tested for branching.
Switch case is not effective when checking for a range value.	If else is a better option to check ranges.
Switch case cannot handle floating point values	If else can handle floating point values
The case label must be constant(Characters of integers).	If else can use variables also for conditions.
Switch statement provides a better way to check a value against a number of constants .	If else is not suitable for this porpose .

COMPUTER PROGRAMMING

Iterative Statements

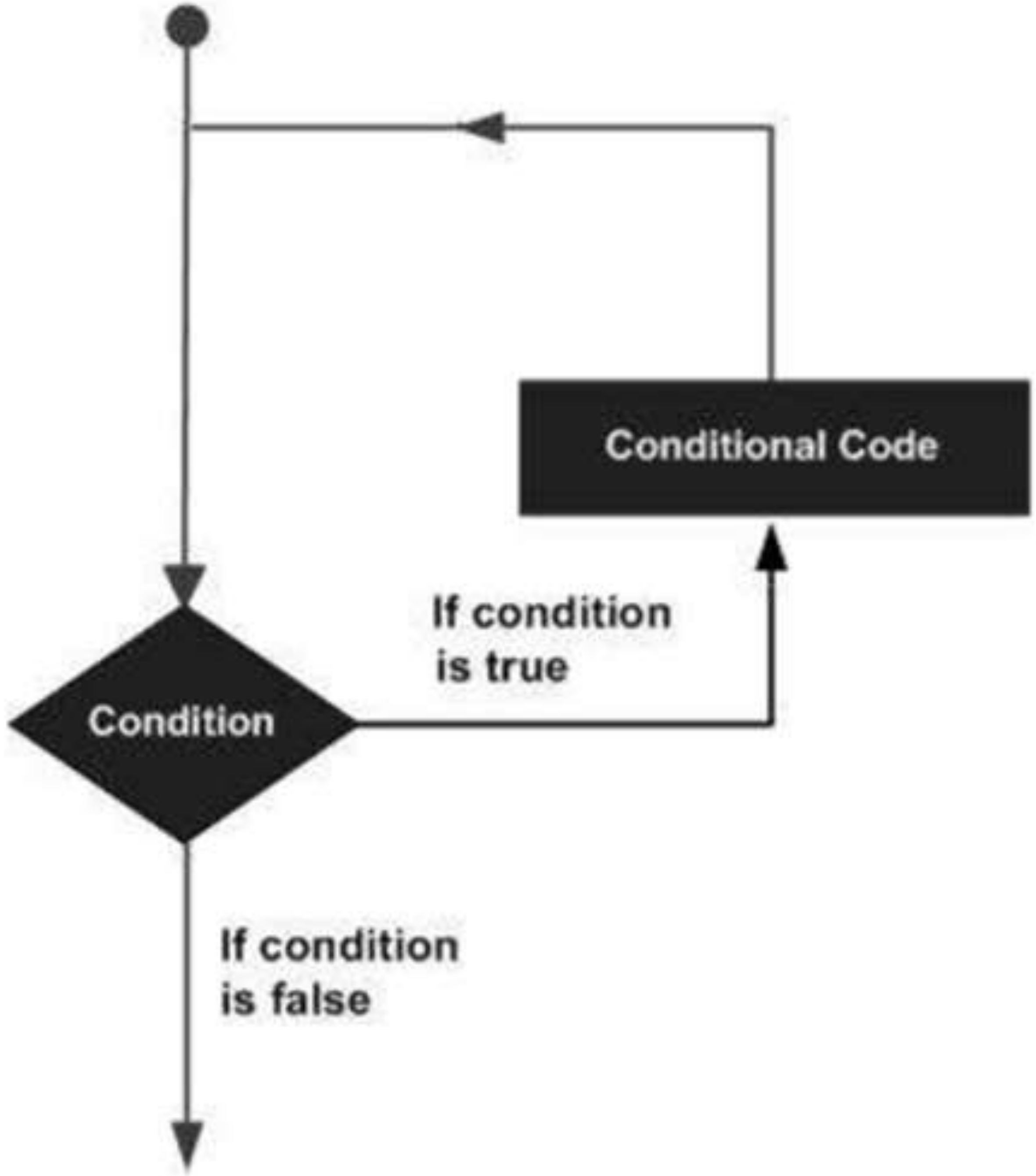
Looping

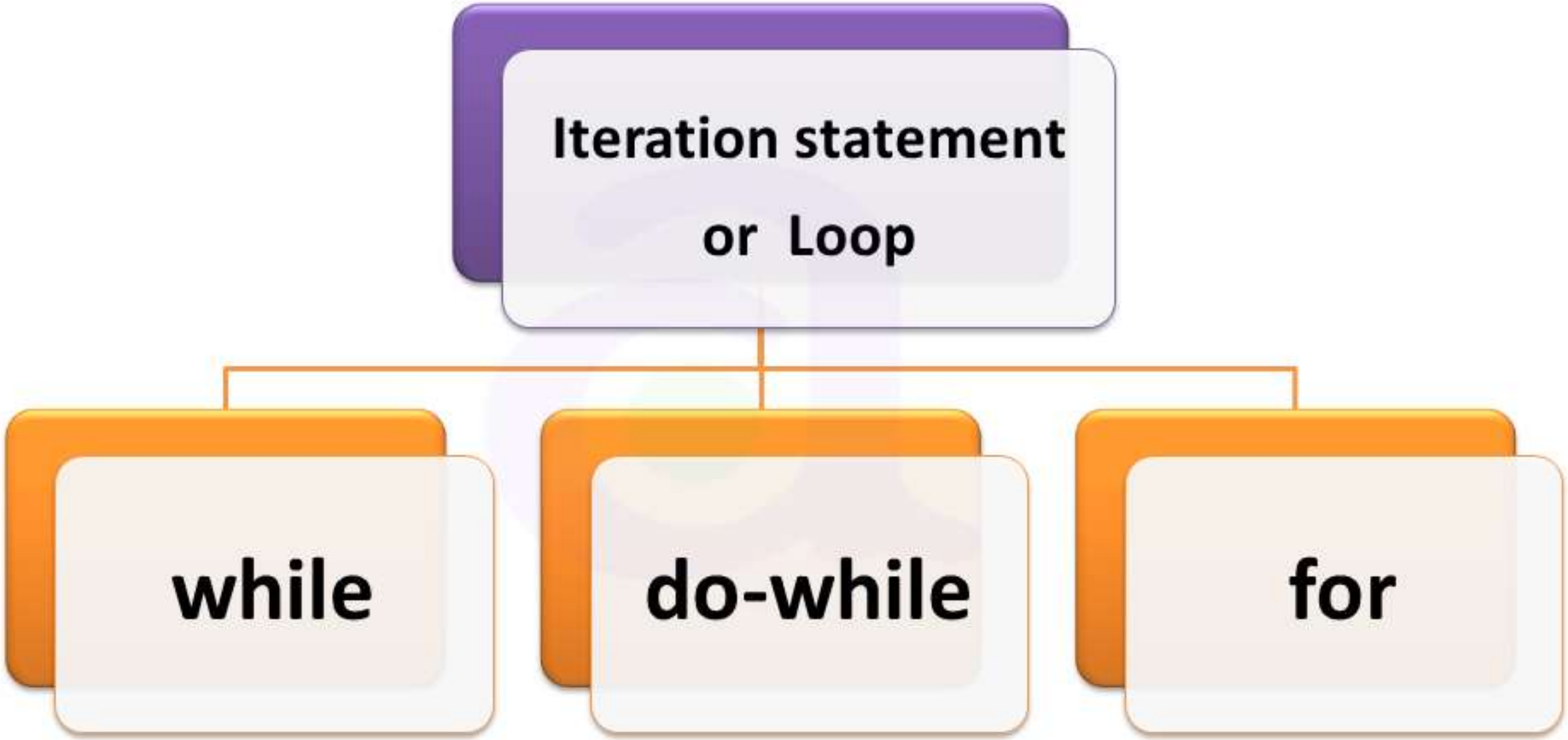
Looping

- ❖ A program may require that a group of instructions be executed repeatedly, till the logical condition is true.

This is known as *Looping.*

- ❖ *Sometimes the required number of repetitions is known in advance; and sometimes the computation continues indefinitely until the logical condition becomes false.*


Conditional Code
If condition
is true
Condition
If condition
is false


Iteration statement or Loop
while
do-while
for

Iterative Statements

while loop

While Loop

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

Syntax

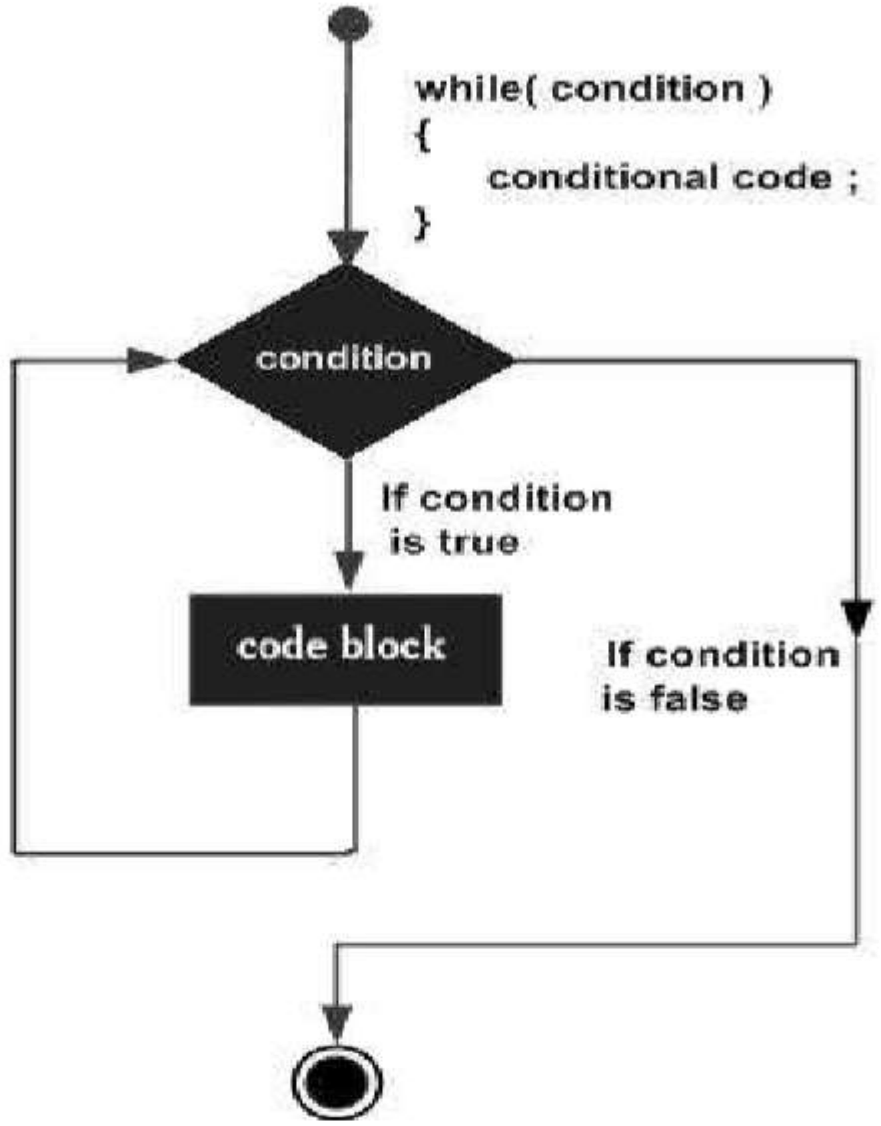
The syntax of a **while** loop in C programming language is:

```
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 if any nonzero value. The loop iterates while the condition is true.

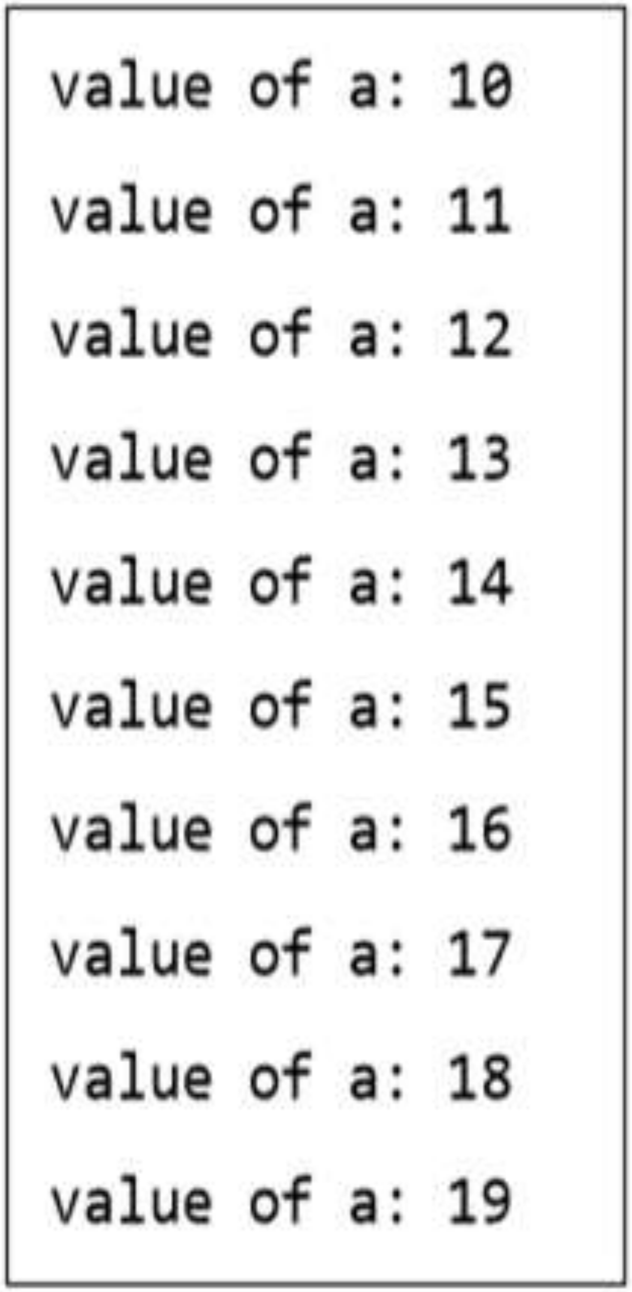
When the condition becomes false, the program control passes to the line immediately following the loop.

While Loop


while(condition)
{
conditional code ;
}
condition
If condition is true
code block
If condition is false

```c
#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;
}
```

While Loop



```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

While Loop

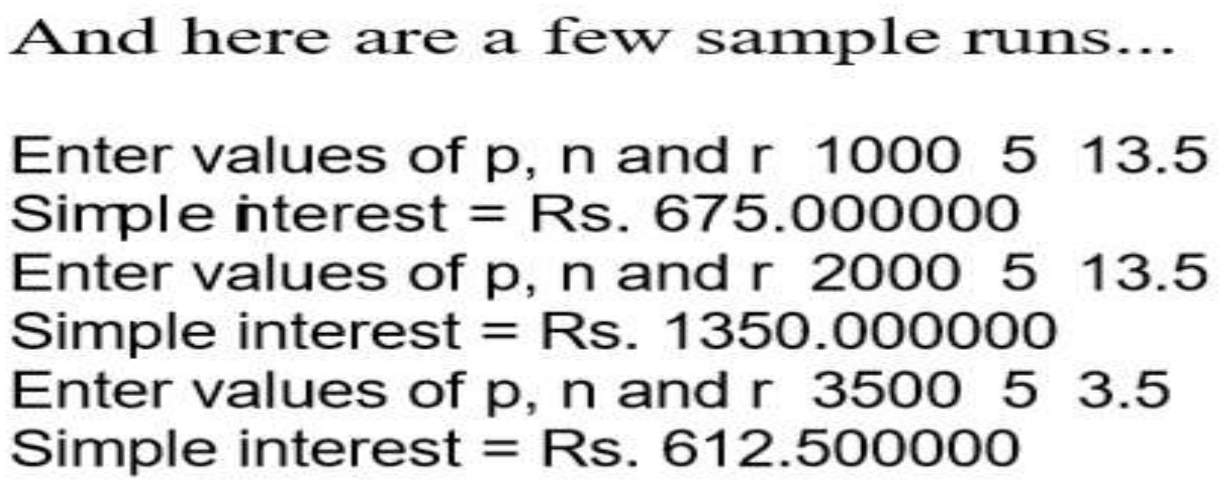
The condition being tested may use relational or logical operators as shown in the following examples:

```
while ( i <= 10 )
while ( i >= 10 && j <= 15 )
while ( j > 10 && ( b < 15 || c < 20 ) )
```

While Loop

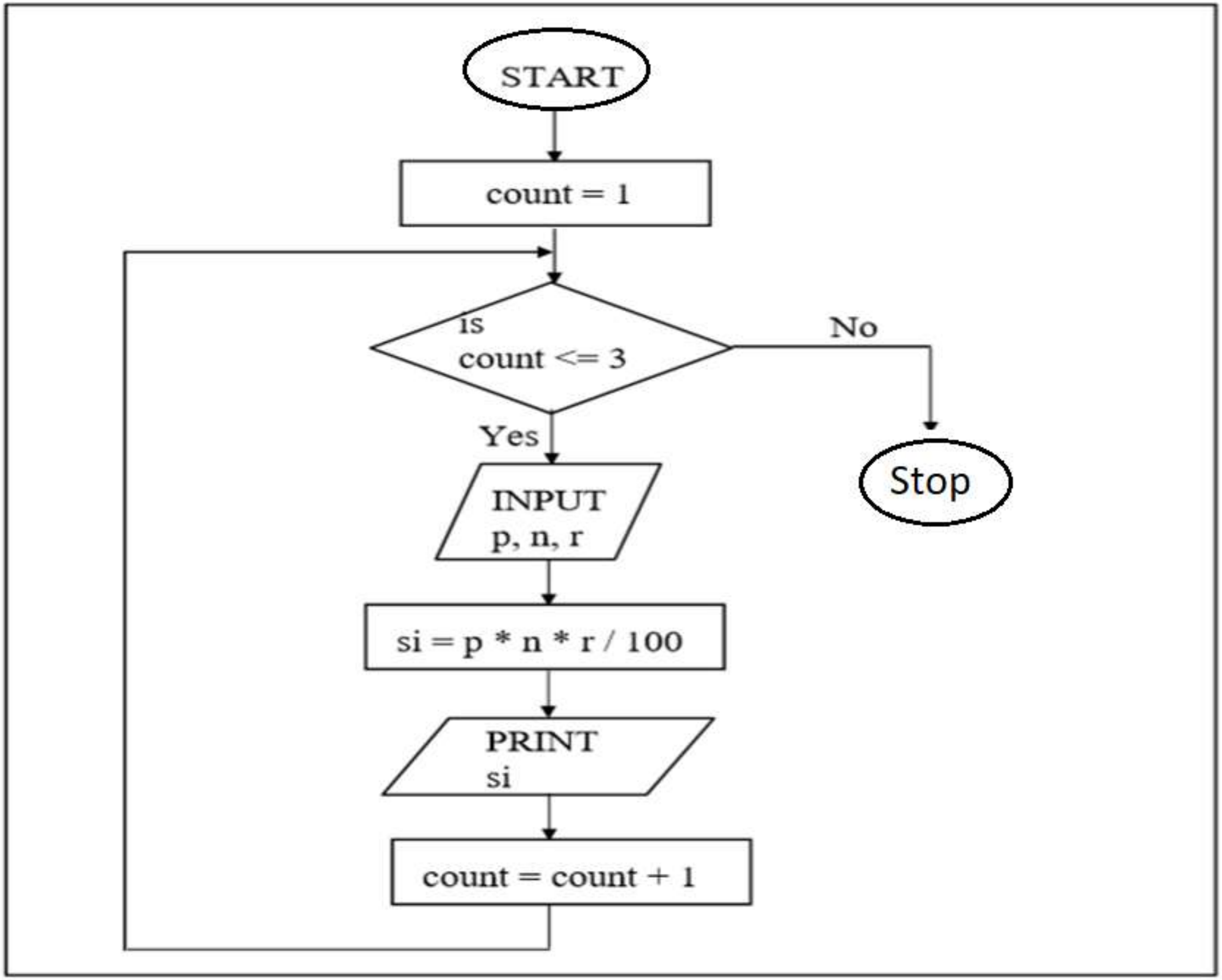
```
/* Calculation of simple interest for 3 sets of p, n and r */
main( )
{
    int   p, n, count ;
    float   r, si ;

    count = 1 ;


   while ( count <= 3 )
   {
        printf ( "\nEnter values of p, n and r " ) ;
        scanf ( "%d %d %f", &p, &n, &r ) ;
        si = p * n * r / 100 ;
        printf ( "Simple interest = Rs. %f", si ) ;

        count = count + 1 ;
   }
}
```

While Loop



And here are a few sample runs...

```
Enter values of p, n and r  1000  5  13.5
Simple interest = Rs. 675.000000
Enter values of p, n and r  2000  5  13.5
Simple interest = Rs. 1350.000000
Enter values of p, n and r  3500  5  3.5
Simple interest = Rs. 612.500000
```


START
count = 1
is
count <= 3
No
Yes
Stop
INPUT
p, n, r
si = p * n * r / 100
PRINT
si
count = count + 1

While Loop

```
#include<stdio.h>
int main() {
int i=1,n,f=1;
printf("\n enter value of n=");
scanf("%d", &n);
while(i<=n){
    f = f*i;
    i++;
}
    printf("Factorial of %d = %d",n, f);
}
```

Iterative Statements

for loop

for Loop

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:

1. 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.
2. 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.

for Loop

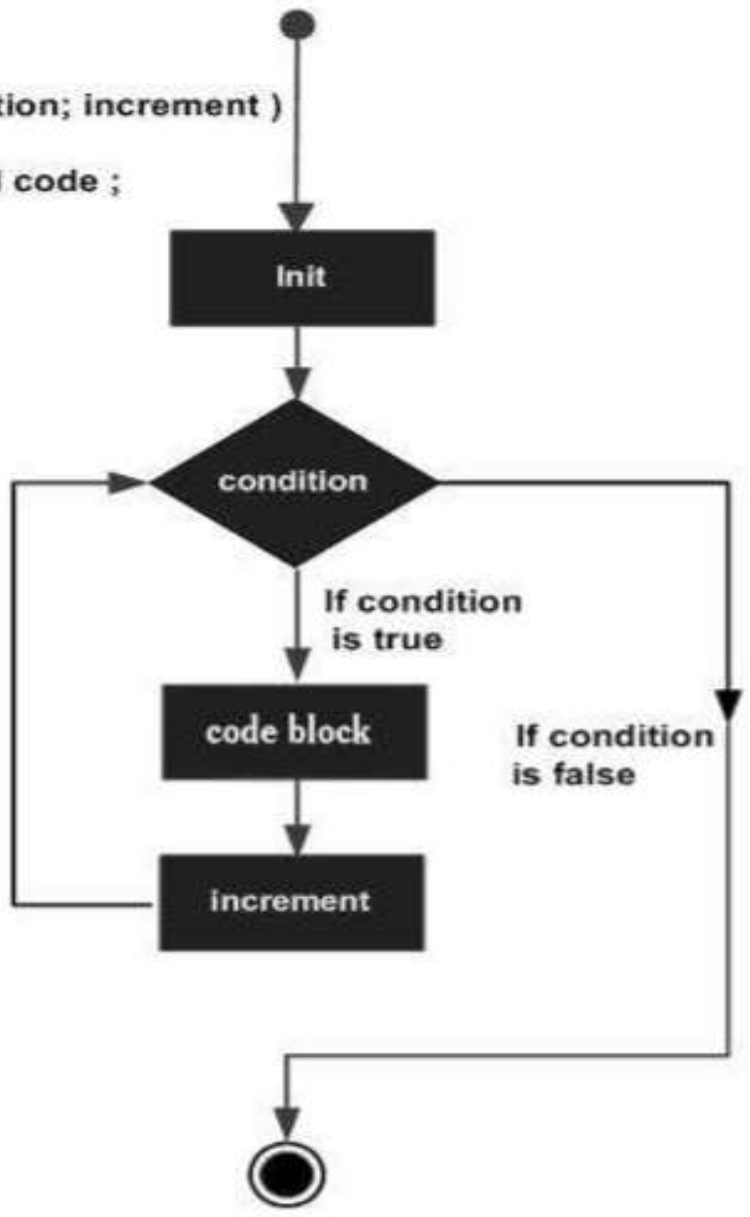
3. After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

4. 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.

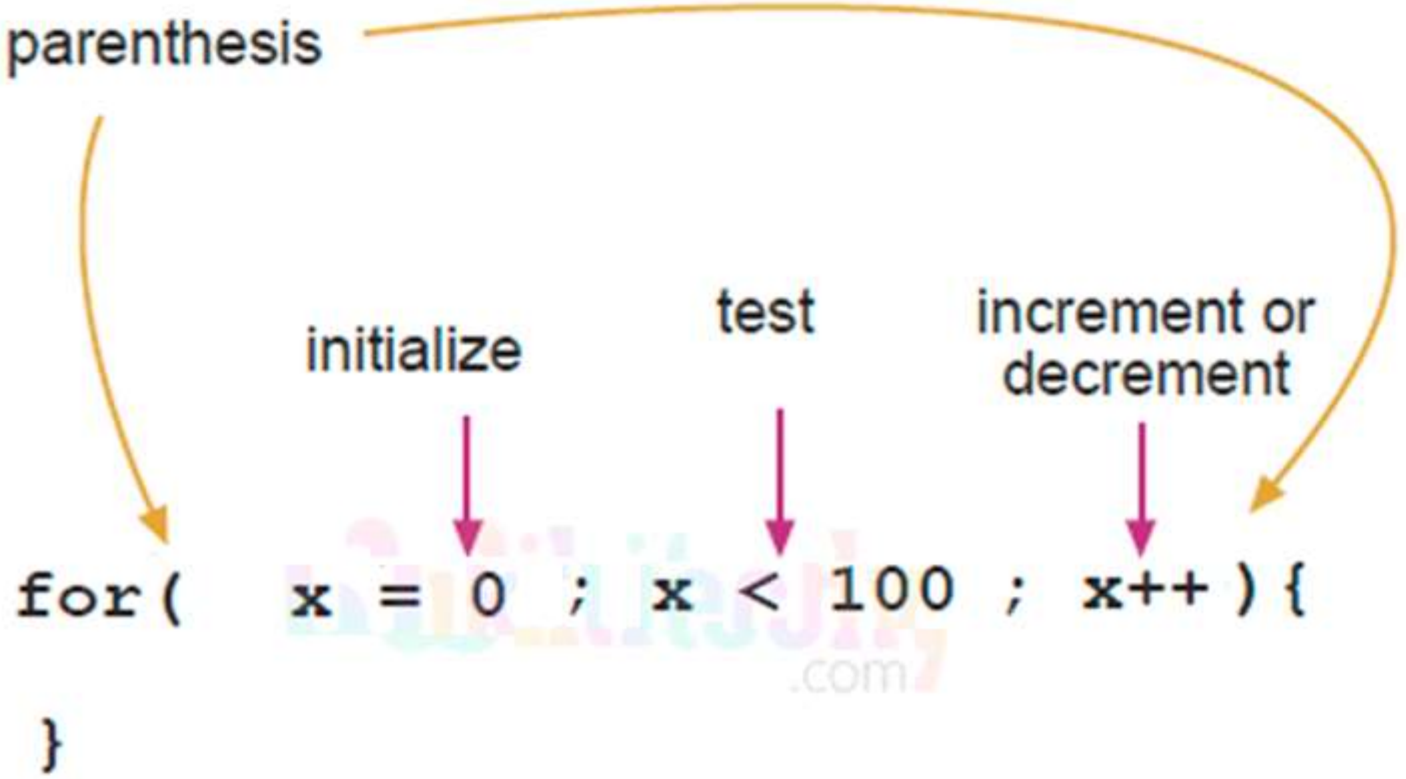
for Loop

Flow Diagram

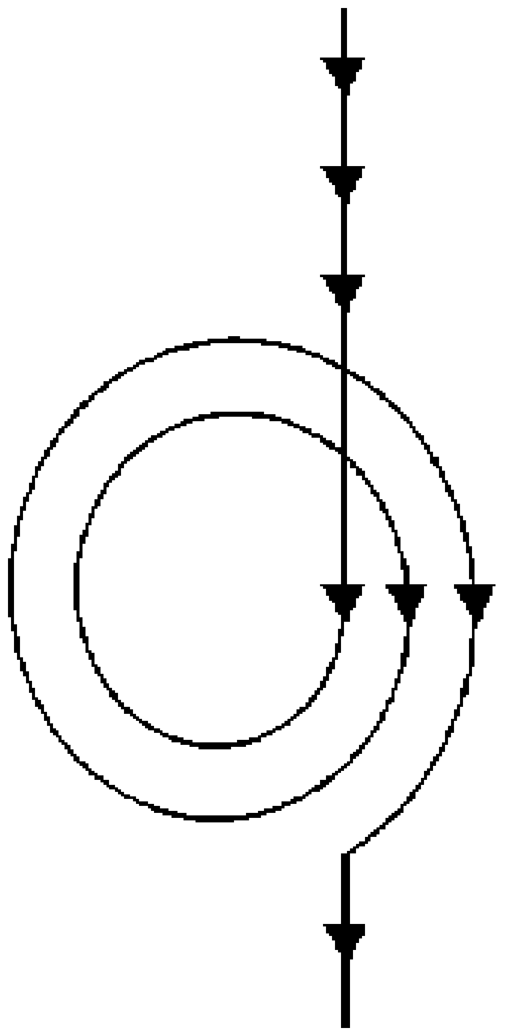
```
for( init; condition; increment )
{
    conditional code ;
}
```



for Loop

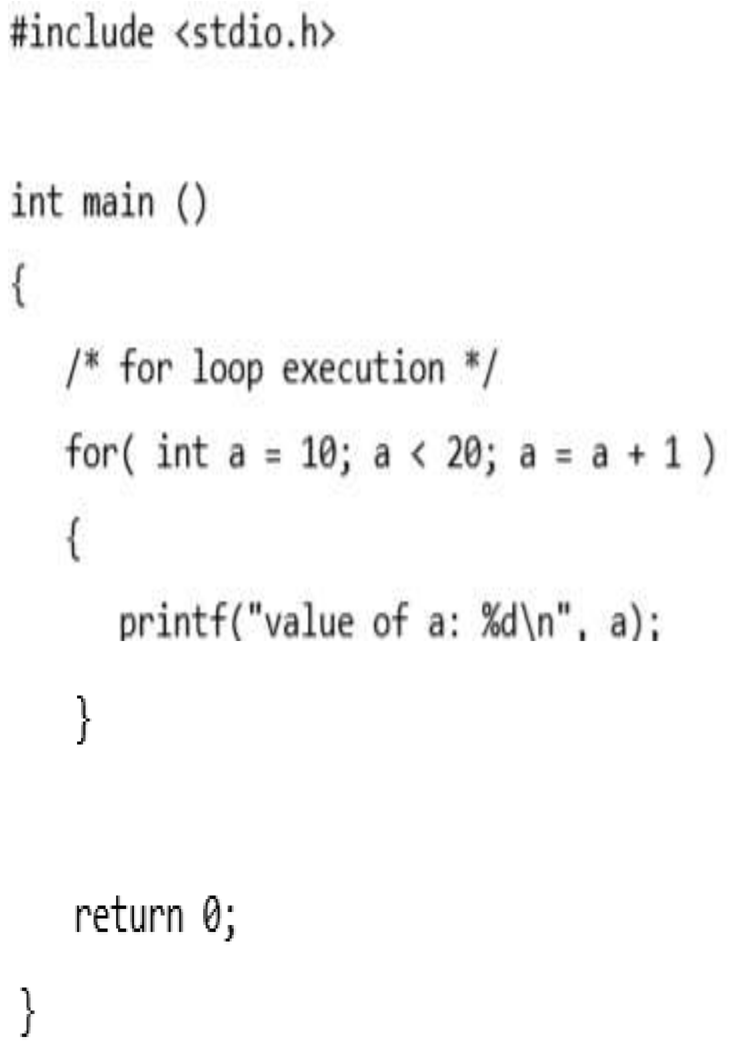

parenthesis
initialize
test
increment or decrement
for(x = 0 ; x < 100 ; x++){
}

for Loop

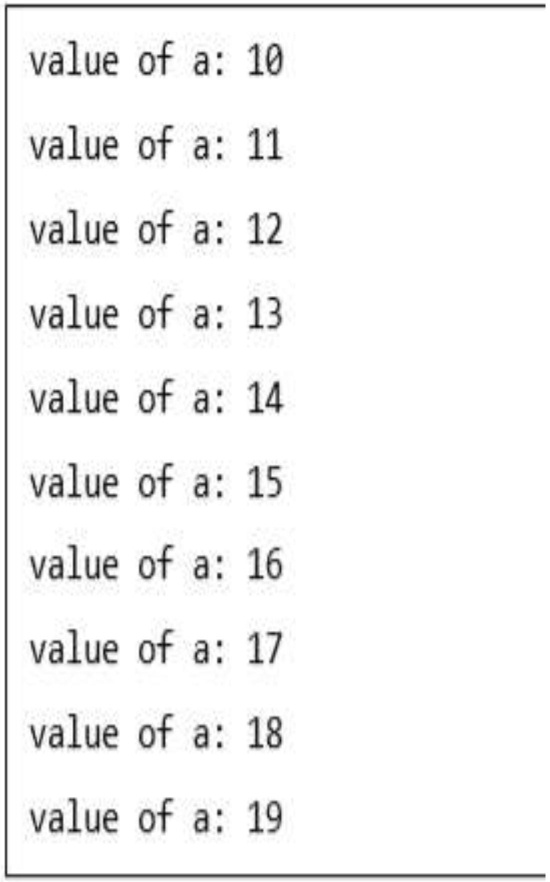


```
int data1 = 100;

int i;

data1 =  data1 + 10;


for (i = 0; i < 3; i++)
{

    data1 = data1 + 20;

}


data1 = data1 + 30;
```

for Loop



```
#include <stdio.h>


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


    return 0;
 }
```



```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

for Loop

Factorial using for loop

```
main()
{
int i,n=5,f=1;
for(i=1;i<=n;i++)
{
   f = f*i;
}
   printf(“Factorial is=%d", f);
}
```

COMPUTER PROGRAMMING

Iterative Statements

do-while looping

do-while Loop

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:

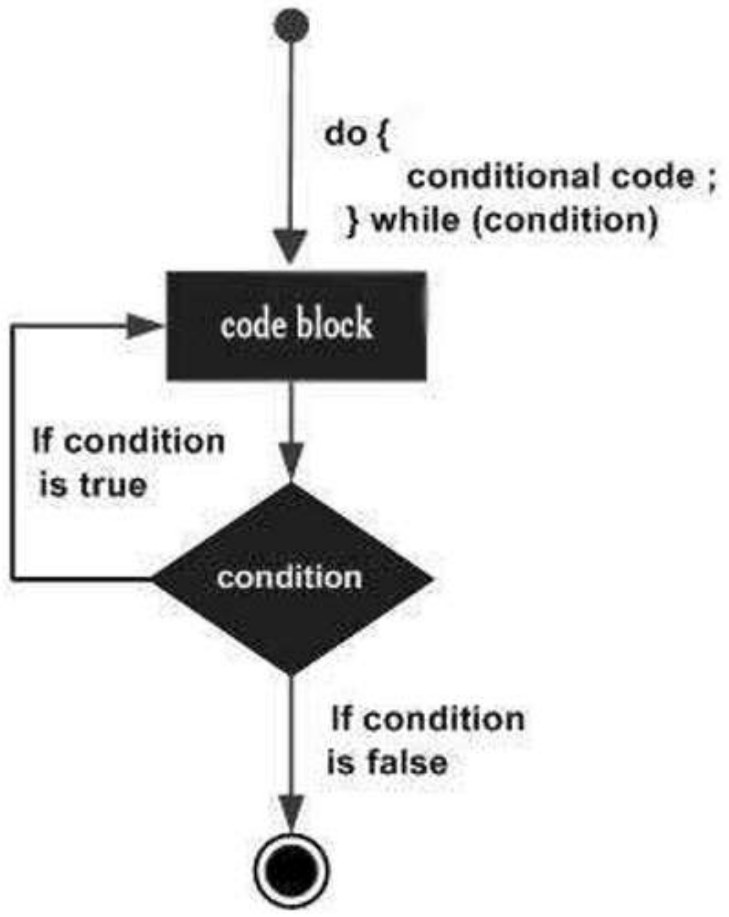
```
do

{

   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.

do-while Loop

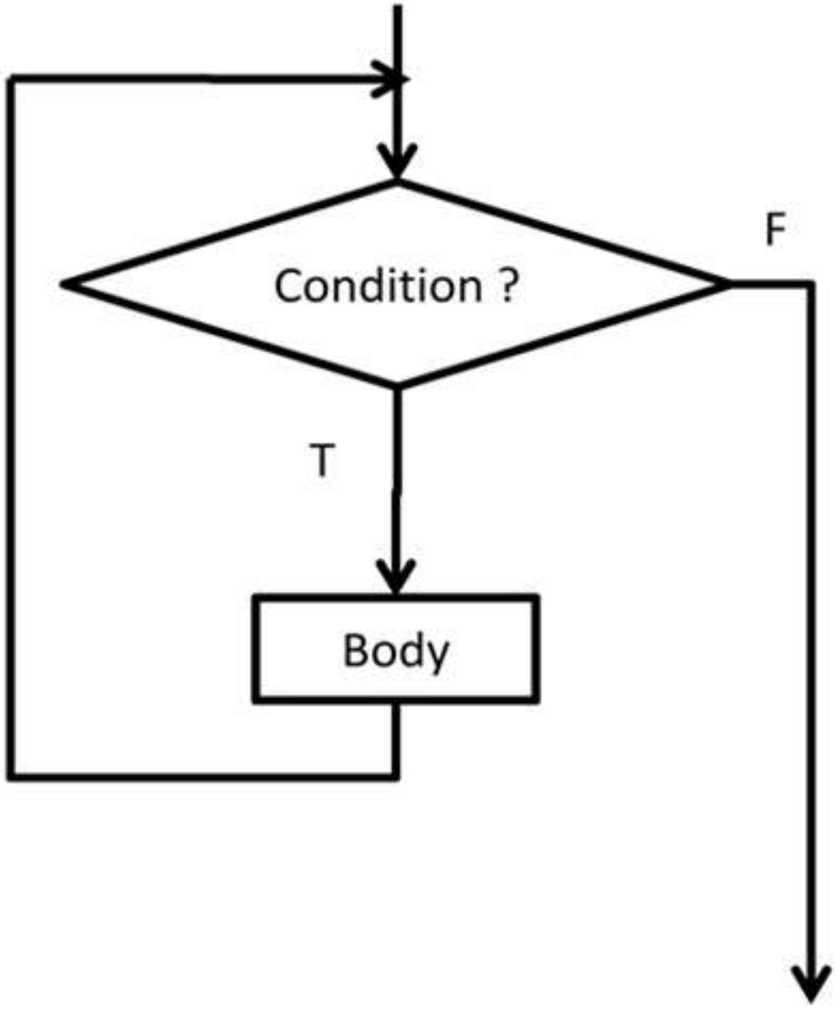
If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Flow Diagram

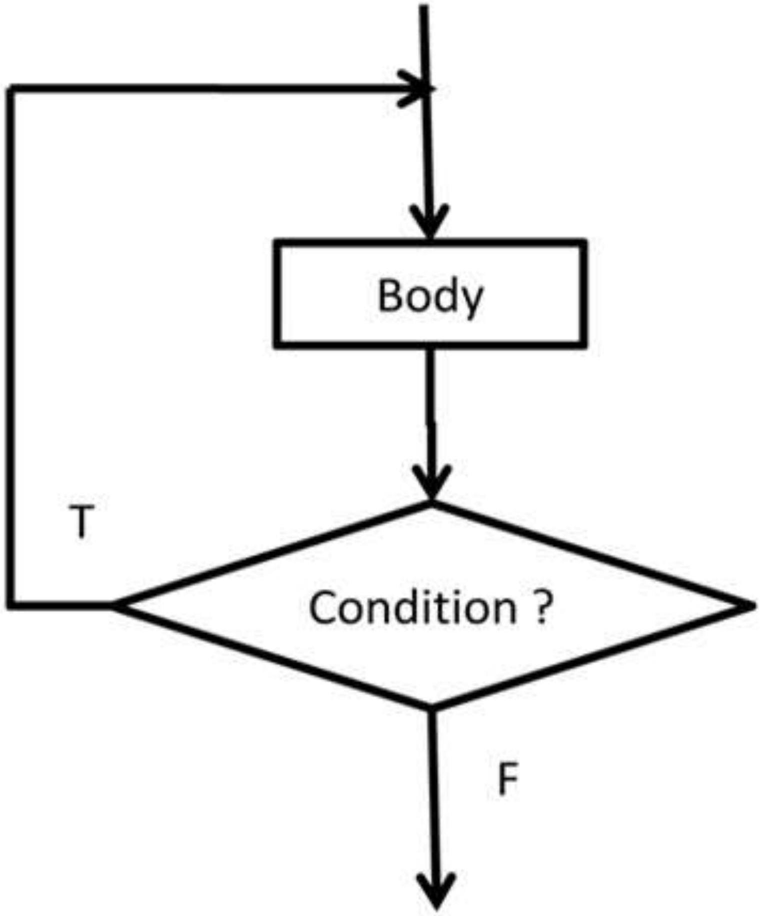


While vs do-while

```
while( condition )
        body;
```



```
do {
        body;
} while( condition );
```



do-while Loop

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;
```

do-while Loop

When the above code is compiled and executed, it produces the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

do-while Loop

Factorial using do-while

```
main()
{
int i=1,n=5,f=1;
do{
    f = f*i;
    i++;
} while(i<=n);          // semi colon is must
    printf("Factorial is =%d",f);
}
```

COMPUTER PROGRAMMING

Iterative Statements

Nested Loops

Nested Loops

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);

}
```

Nested Loops

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

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

Nested Loops

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

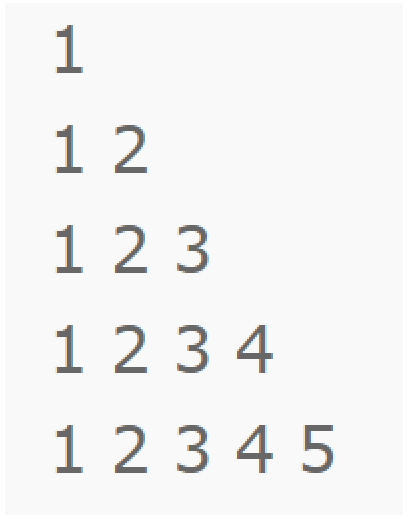
```
do
{
   statement(s);
   do
   {
      statement(s);
   }while( condition );

}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.

Nested Loops

Example

```
#include <stdio.h>
int main()
{
    int i=1,j;
    while (i <= 5)
    {
        j=1;
        while (j <= i )
        {
            printf("%d ",j);
            j++;
        }
        printf("\n");
        i++;
    }
    return 0;
}
```



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

Nested Loops

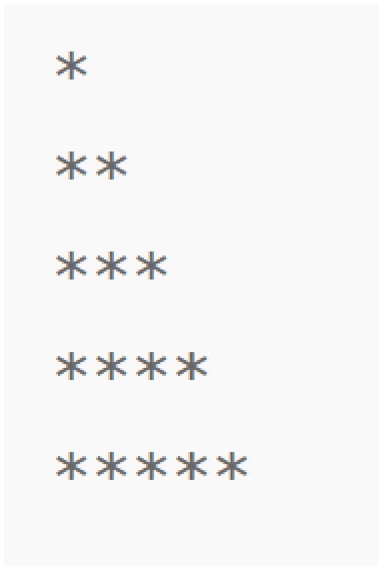
```
#include <stdio.h>
int main()
{
    int i=1,j;
    do
    {
        j=1;
        do
        {
            printf("*");
            j++;
        }while(j <= i);
        i++;
        printf("\n");
    }while(i <= 5);
    return 0;
}
```

Example

```
*

**

***

****

*****
```

Nested Loops

```
#include<stdio.h>
main(){
int i,j,n=6;
for(i=1;i<=n;i++)
{
 for(j=1;j<=i;j++)
{
   printf("*");
 }
 printf("\n");
}
}
```



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

Nested Loops

Practice

Write Programs for printing bellow output

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

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

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

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

Loop Control Statements

break and continue Statements

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C supports the following control statements.

Control Statement	Description
break statement	Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
goto statement	Transfers control to the labeled statement.

break Statement

The **break** statement in C programming has the following two usages:

- When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax

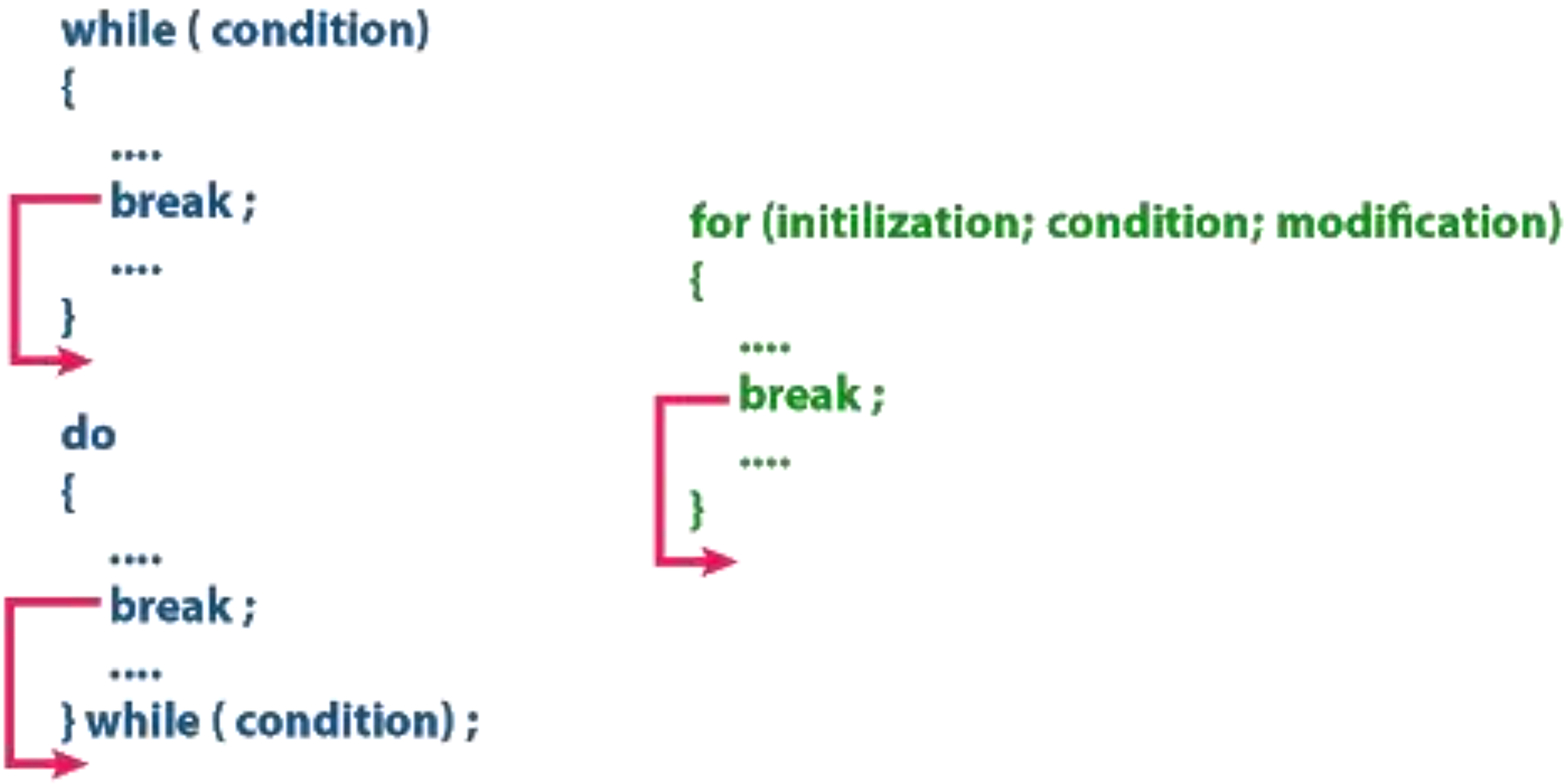
The syntax for a **break** statement in C is as follows:

```
break;
```

break Statement

- In any loop break is used to jump out of loop skipping the code below it without caring about the test condition.
- It interrupts the flow of the program by breaking the loop and continues the execution of code which is outside the loop.
- The common use of break statement is in switch case where it is used to skip remaining part of the code.

break Statement

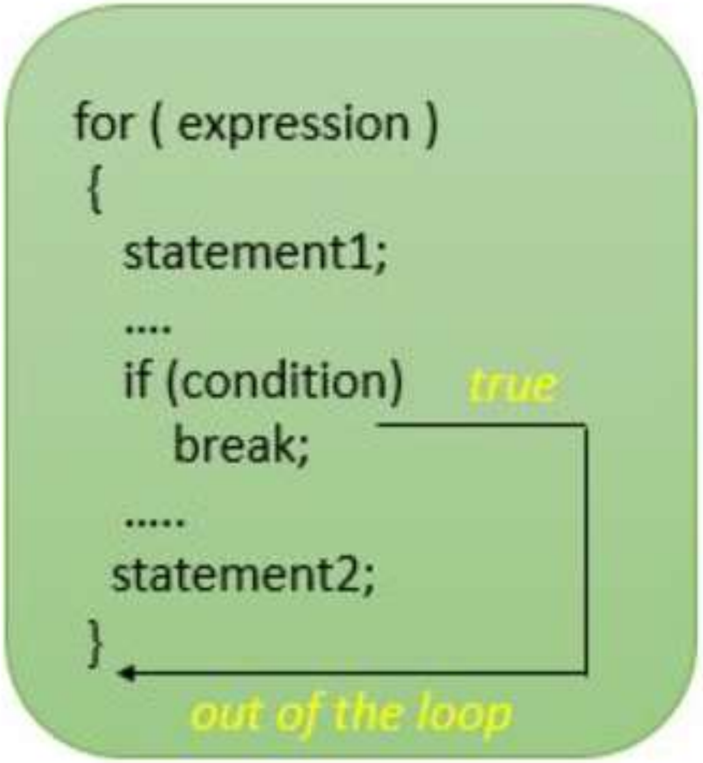


```
while ( condition)
{
    ....
    break ;
    ....
}
```

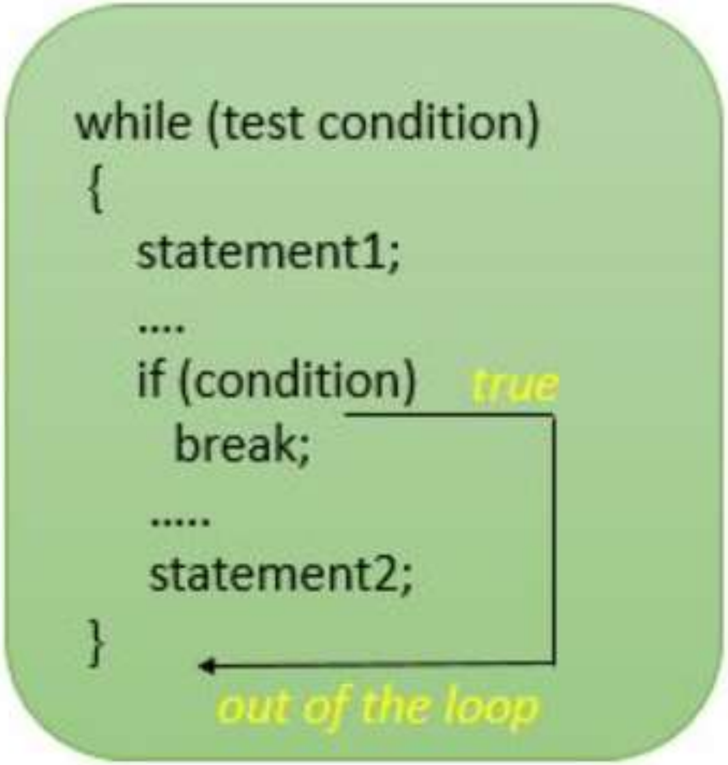
```
do
{
    ....
    break ;
    ....
} while ( condition) ;
```

```
for (initilization; condition; modification)
{
    ....
    break ;
    ....
}
```

break Statement



```
for ( expression )
{
  statement1;
  ....
  if (condition)
     break;
  .....
 statement2;
}
```



```
while (test condition)
{
   statement1;
   ....
   if (condition)
     break;
    .....
    statement2;
}
```

break Statement

Example: C program to take input from the user until user enters zero.

```
#include <stdio.h>
int main ()
{
 int a;
while (1)
 {
printf("enter the number:");
scanf("%d", &a);
if ( a == 0 )
 break;
 }
 return 0;
}
```

```
#include <stdio.h>
int main () {
   int a = 10;
   /* while loop execution */
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
      if( a > 15) {
        /* terminate the loop using break statement */
         break;
      }
   }
```

break Statement

When the above code is compiled and executed, it produces the following result:

```
value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15
```

Example: C program to print the day of the week according to the number of days entered

```c
#include <stdio.h>

int main()
{
   int n;
   printf("Enter the number of the day :");
   scanf(" %d ", &n);
   switch (n)
   {
    case 1:
       printf("Sunday");
       break;
    case 2:
       printf("Monday");
       break;
    case 3:
       printf("Tuesday");
       break;
    case 4:
       printf("Wednesday");
       break;
    case 5:
       printf("Thrusday");
       break;
    case 6:
       intf("Friday");
       break;
    case 7:
       printf("Saturday");
       break;
    default:
       printf("You entered wrong day");
       exit(0);
  }
  return 0;
 }
```

Continued…

continue Statement

The **continue** statement in C programming works somewhat like the **break** statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, **continue** statement causes the program control to pass to the conditional tests.

Syntax

The syntax for a **continue** statement in C is as follows:

```
continue;
```

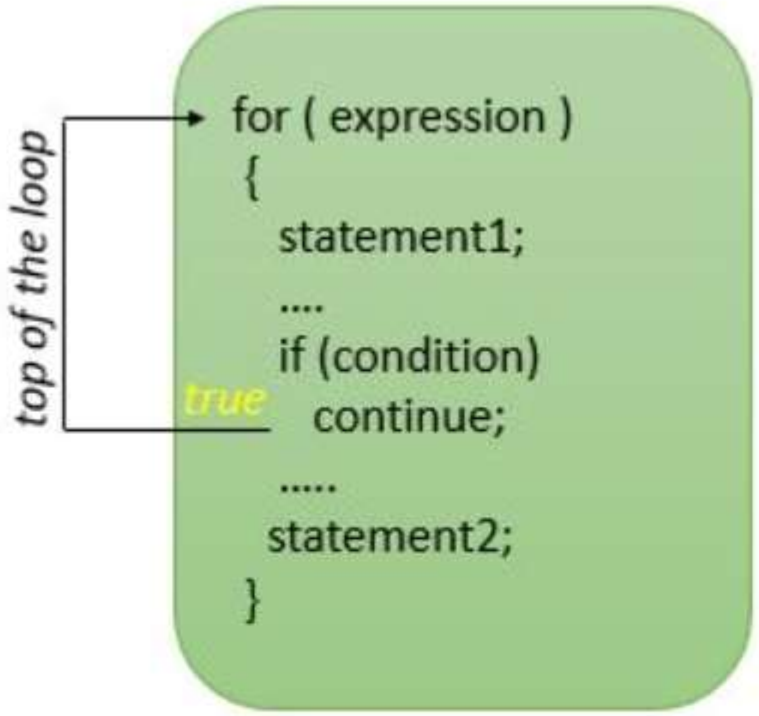
continue Statement

- ❖ Like a break statement, continue statement is also used with if condition inside the loop to alter the flow of control.

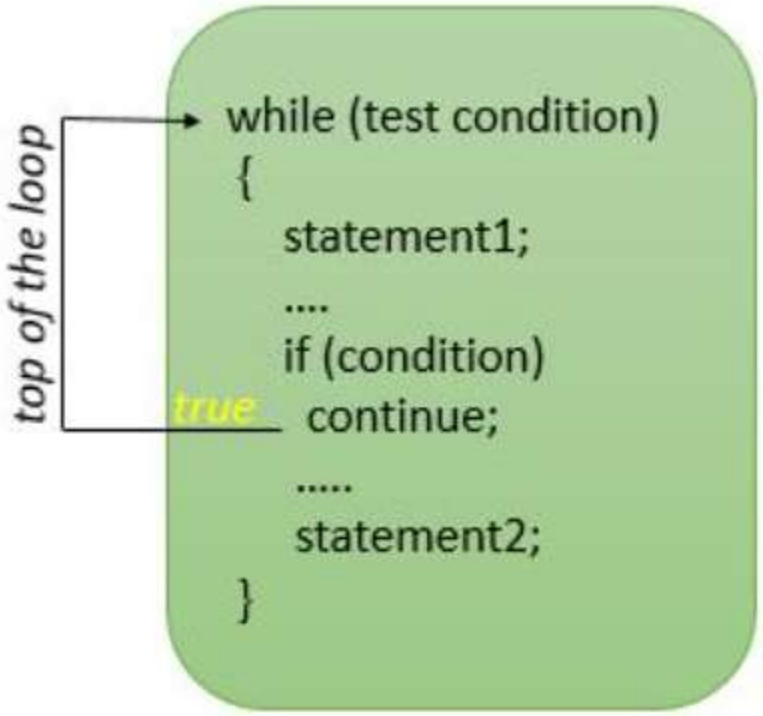
- ❖ When used in while, for or do...while loop, it skips the remaining statements in the body of that loop and checks the condition for next iteration.

- ❖ Unlike break statement, continue statement when encountered doesn’t terminate the loop, rather interrupts a particular iteration.

continue Statement



```
for ( expression )
{
  statement1;
  ....
  if (condition)
   continue;
  .....
 statement2;
}
```



```
while (test condition)
{
   statement1;
   ....
   if (condition)
    continue;
    .....
    statement2;
}
```

continue Statement

```
#include <stdio.h>
int main (){
 int a, sum = 0;
for (a = 0; a < 10; a++) {
 if ( a % 2 == 0 )
continue;
sum = sum + a;
 }
 printf("sum = %d",sum);
return 0; }
```

```
int main () {
   int a = 10;
   do {
      if( a == 15)  {
         /* skip the iteration */
         a = a + 1;
         continue;
      }
      printf("value of a: %d\n", a);
      a++;
   }while( a < 20 );
   return 0;
}
```

continue Statement

When the above code is compiled and executed, it produces the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

COMPUTER PROGRAMMING

Loop Control Statements

goto statements

goto Statement

A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

NOTE: Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

Syntax

The syntax for a **goto** statement in C is as follows:

```
goto label;
..
.
label: statement;
```

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

```
main( )
{
    int  goals ;

    printf ( "Enter the number of goals scored against India" ) ;
    scanf ( "%d", &goals ) ;

    if ( goals <= 5 )
        goto sos ;
    else
    {
        printf ( "About time soccer players learnt C\n" ) ;
        printf ( "and said goodbye! adieu! to soccer" ) ;
        exit(  ) ;  /* terminates program execution */
    }

    sos :
        printf ( "To err is human!" ) ;

    }
```

And here are two sample runs of the program...

```
Enter the number of goals scored against India 3
To err is human!
Enter the number of goals scored against India 7
About time soccer players learnt C
and said goodbye! adieu! to soccer
```