

In any programming language including C, loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

Types of Loop

There are 3 types of Loop in C language, namely:

1. while loop
2. for loop
3. do while loop

while loop

while loop can be addressed as an **entry control** loop. It is completed in 3 steps.

- Variable initialization.(e.g `int x = 0;`)
- condition(e.g `while (x <= 10)`)
- Variable increment or decrement (`x++` or `x--` or `x = x + 2`)

Syntax :

```
variable initialization;
while(condition)
{
    statements;
    variable increment or decrement;
}
```

Example: Program to print first 10 natural numbers

```
#include<stdio.h>

void main( )
{
    int x;
    x = 1;
    while(x <= 10)
    {
        printf("%d\t", x);
        /* below statement means, do x = x+1, increment x by 1*/
        x++;
    }
}
```

for loop

for loop is used to execute a set of statements repeatedly until a particular condition is satisfied.

We can say it is an **open ended loop**. General format is,

```
for(initialization; condition; increment/decrement)
{
    statement-block;
}
```

In `for` loop we have exactly two semicolons, one after initialization and second after the condition. In this loop we can have more than one initialization or increment/decrement, separated using comma operator. But it can have only one **condition**.

The `for` loop is executed as follows:

1. It first evaluates the initialization code.
2. Then it checks the condition expression.
3. If it is **true**, it executes the for-loop body.
4. Then it evaluate the increment/decrement condition and again follows from step 2.
5. When the condition expression becomes **false**, it exits the loop.

Example: Program to print first 10 natural numbers

```
#include<stdio.h>

void main( )
{
    int x;
    for(x = 1; x <= 10; x++)
    {
        printf("%d\t", x);
    }
}
```

Nested `for` loop

We can also have nested `for` loops, i.e one `for` loop inside another `for` loop. Basic syntax is,

```
for(initialization; condition; increment/decrement)
{
    for(initialization; condition; increment/decrement)
    {
        statement ;
    }
}
```

Example: Program to print half Pyramid of numbers

```
#include<stdio.h>

void main( )
{
    int i, j;
    /* first for loop */
    for(i = 1; i < 5; i++)
    {
        printf("\n");
        /* second for loop inside the first */
        for(j = i; j > 0; j--)
        {
            printf("%d", j);
        }
    }
}
```

do while loop

In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of do-while loop. do statement evaluates the body of the loop first and at the end, the condition is checked using while statement. It means that the body of the loop will be executed at least once, even though the starting condition inside while is initialized to be **false**. General syntax is,

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

Example: Program to print first 10 multiples of 5.

```
#include<stdio.h>

void main()
{
    i
    n
    t

    a
    ,

    i
    ;

    a
    =

    5
    ;
    i = 1;
    do
    {
        printf("%
        d\t",
        a*i);i++;
    }
    while(i <= 10);
}
```

5 10 15 20 25 30 35 40 45 50

Q. Give differences between **while** and **do-while** statement.

Solution: The differences between while and do-while statement are shown below:

while statement	do-while statement
-----------------	--------------------

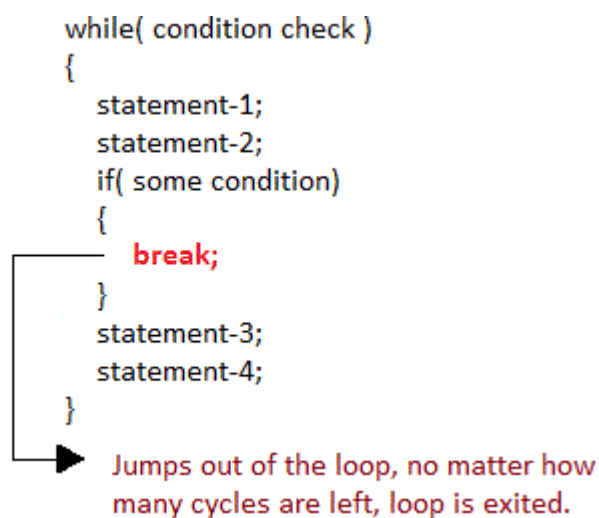
1. The statement block in while statement is executed when the values of the condition is true .	1. The statement block is executed at least once irrespective of the value of the condition.
2. The condition is evaluated at the beginning of the loop. The statement block is executed if the value of the condition is true .	2. The condition is evaluated at the end of the loop. The statement block is executed again if the value of the condition is true .

Jumping Out of Loops

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becomes **true**. This is known as jumping out of loop.

1) break statement

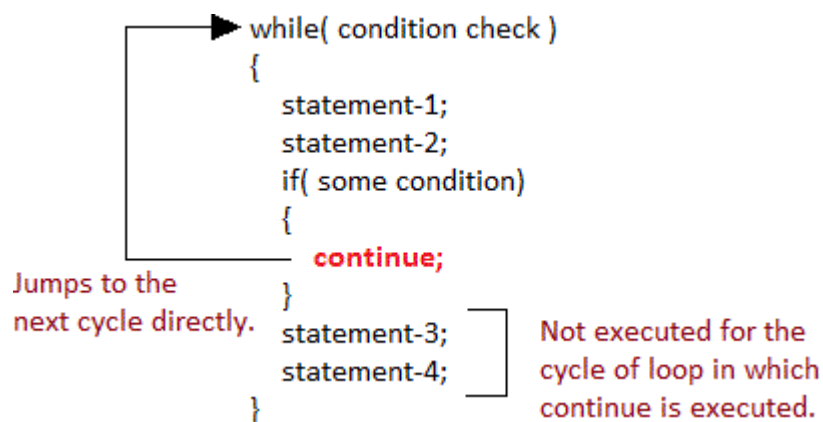
When `break` statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.



2) c
o
n
t
i
n
u
e

s
t
a
t
e
m
e
n
t

It causes the control to go directly to the test-condition and then continue the loop process. On encountering `continue`, cursor leave the current cycle of loop, and starts with the next cycle.



Q. List a few unconditional control statement in C.

Solution: Statements, which are used to transfer the control to any part of the program without checking a condition, are referred as unconditional statement in C.

Unconditional control statement in C are:

- (i) **break** statement
- (ii) **continue** statement
- (iii) **goto** statement
- (iv) **exit()** function.

goto Statement:

The **goto** statement is an unconditional transfer of control statement. It is used to transfer the control from one part of the program to another. The place to which the control is transferred is identified by a statement **label**. It has the following form.

goto label;

Where **label** is the statement label which is available anywhere in the program.

Eg.

```
_____  
goto display;  
_____  
_____  
display;  
_____;
```

When this statement is executed, the control is transferred to the statement label display which is followed by a comma.

break statement:

The **break** statement is used to transfer the control to the end of a statement block in a loop. It is an unavoidable statement to transfer the control to the end of a **switch** statement after executing any one statement block. It can be used within a for, while, do-while, or switch statement. It has the following form.

break;

Note that a program can be written without this statement other than in a **switch** statement.

Eg.

Consider the following example.

for(i

= 1; i<= n; i++)

{

break;

}



control transferred to the end of the statement block.

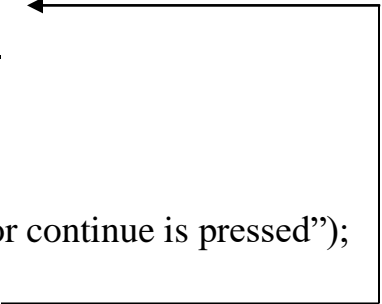
continue Statement:

The **continue** statement is used to transfer the control to the beginning of a statement block in a loop. It has the following form.

continue;

Eg.

```
for(i=1; i<= 20; i++)  
{  
  Ch= getch();  
  if (ch == "C")  
  {  
    printf("\n C for continue is pressed");  
    continue;  
  }  
  -----  
}
```



Control is transferred to the beginning of the block

When this statement is executed, the control is transferred to the beginning of the loop such that the loop is repeated 20 times irrespective of the key pressed.

Exit () Function:

The exit () function is used to transfer the control to the end of a program (i.e. to terminate the program execution). It uses one argument in () and the value is zero for normal termination or non zero for abnormal termination.

Eg.

```
if( n < 0 )  
{  
    printf("\n Factorial is not available for negative numbers");exit(0);  
}  
_____;
```

Note that the program execution is terminated when the value of the variable `n` is negative.

Return statement:

The keyword **return** is used to return any value from the function called. This statement terminates the function and returns a value to its caller. The **return** statements may also be used to exit from any function without returning any value. The **return** statement may or may not include an expression. The general format of using **return** statement is given below:

```
return;                /* without expression */
return(expression) /* with expression */
```

The above expression can be an integer value, a float or a char, this depends on the data type of the function declared.

Ex: Explain the functions of the following statements in C.

- i) `break`
- ii) `return`
- iii) `continue`

OR

In what situations will you use a „break“ statement and a „continue“ statement? Explain with an example.

Solution: The uses of „return“, „break“ and „continue“ statements are given below:

Return statement: The keyword **return** is used to return any value from the function called. This statement terminates the function and returns a value to its caller. The **return** statements may also be used to exit from

any function without returning any value. The **return** statement may or may not include an expression. The general format of using **return** statement is given below:

```
return;                /* without expression */
return(expression) /* with expression */
```

The above expression can be an integer value, a float or a char, this depends on the data type of the function declared.

break statement: The break statement causes an immediate exit from the innermost loop structure or to exit from a switch. It can be used within a for, while, do-while, or switch statement.

- If we need to come out of a running program immediately without letting it to perform any further operation in a loop, then we can use break control statement.

The general format of the break statement is:

break; <Enter>

Following program segment is written to explain the use of break with while loop structure:

```
#include<stdio.h>main( )
{
    int i, value;
    i=0;
    while(i<=10)
    {
        printf("Enter a number\n");
        scanf("%d",&value);
        if value == 0 || value<0)
        {
            printf("Zero or negative value found.\n");break;
        }
        i++;
    }
```

```

    }
}

```

The above program segment processes only the positive integers. Whenever the zero or negative value is found the program will display the message “Zero or negative value found” as an error and it will not execute the loop further.

continue statement: In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed. The keyword **continue** allows us to do this. When the keyword **continue** is encountered inside any C loop control automatically passes to the beginning of the loop.

A **continue** is usually associated with an **if**. As an example, let’s consider the following program.

```

main( )
{
    int i, j;
    for(i=1;i<=2;i++)
    {
        for(j=1;j<=2;j++)
        {
            if(i==j)
                continue;
            printf(“\n%d %d\n”,i,j);
        }
    }
}

```

The output of the above program would be:

```

1    2
2    1

```

Note that when the value of **i** equals that of **j**, the **continue** statement takes the control to the for loop(inner) bypassing rest of the statements pending execution in the **for** loop(inner).

Ex: What are the uses of the following in C?

- (i) **break** (ii) **continue**

Solution:

- (i) **break:** A **break** statement is used to terminate the execution of a statement block. The next statement which follows this statement block will be executed. The keyword is **break**. When a **break** statement is executed in a loop, the repetition of the loop will be terminated.
- (ii) **continue:** A **continue** statement is used to transfer the control to the beginning of a statement block. The keyword is **continue**. When a **continue** statement is executed in a loop, the execution is transferred to the beginning of the loop.

Ex: What is the similarity between break and continue statements?

Solution: Both break and continue are jump statements.

Ex: What is the function of break statement in a while, do-while or for loop?

Solution: If a break statement is included in a while, do-while or for loop, then control will immediately be transferred out of the loop when the break statement is encountered. This provides a convenient way to terminate the loop if an error or other irregular condition is detected.

Ex: Why the use of the goto statement should generally be avoided in a „C“ program?

Solution: The structured feature in „C“ requires that the entire program be written in an orderly, and sequential manner. For this reason, use of the goto statement should generally be avoided in a „C“ program.

Ex: Differentiate between break and exit().

Solution: break just terminates the execution of loop or switch in which it is written, whereas as exit() terminates the execution of the program itself.

Q. Distinguish between the **break** and **continue** statements in C.

Solution: The differences between the **break** and **continue** statements in C are listed below:

break statement	continue statement
1. A break statement is used to terminate the execution of a statement block. The next statement which follows this statement block will be executed. The keyword is break .	1. A continue statement is used to transfer the control to the beginning of a statement block. The keyword is continue .
2. When a break statement is executed in a loop, the repetition of the loop will be terminated.	2. When a continue statement is executed in a loop, the execution is transferred to the beginning of the loop.

Q. What are the differences between **break** and **exit()** function.

Solution: The differences between break and exit() function are shown below:

break statement	exit() function
1. A break statement is used to terminate the execution of a statement block. The next statement which follows this statement block will be executed.	1. An exit() function is used to terminate the execution of a C program permanently.
2. It is used in a program as break;	It is a built-in function and is used/called with necessary argument as exit(0);

Q. Write a program to calculate the sum of digits of a given 5 digit number.

Solution:

```
/* Sum of digits of a 5 digit number */#include<stdio.h>
#include<conio.h>main()
{
    int num, a, n;
    int sum=0; /*sum initialised to zero as otherwise it will contain a garbage value
    */

    clrscr();
    printf("\nEnter a 5-digit number ");
    scanf("%d",&num);
    a=num%10; /*last digit extracted as remainder */n=num/10;
    /*remaining digits*/
    sum=sum+a; /*sum updated with addition of extracted digit */while(n!=0)
```

```

    {
        a=n%10; /* 4th digit */
        n=n/10;
        sum=sum+a;
    }
    printf("\nThe sum of the 5 digits of %d is %d",num, sum);printf("\n\nPress
    any key to exit...");
    getch();
    return;
}

```

Q. Write a program to print first n prime numbers using while loop.

Solution:

```

/* Generate first n prime numbers */
#include<stdio.h> #include<math.h>
main()
{
    int n,num,d,t,count;
    printf("How many primes are required? ");
    scanf("%d",&n);
    printf("%d\n",n);
    printf("First %d primes are:\n");
    printf("%5d",2); /* 2 is the first and only even prime number */count=1;
    num=3; /* now start from 3 and test only odd numbers */
    while(count<n)
    {
        t=sqrt(num);
        d=2;
        while(d<=t)
        {
            if(num%d==0)
                break;
            d++;
        }
        if(d>t)

```

```

        {
            printf("%5d",num);count++;
        }
        num+=2;
    }
}

```

RUN:

How many primes are required? 50First

50 primes are:

```

2   3   5   7   11  13  17  19  23  29  31  37  41  43  47   53
59  61  67  71  73  79  83  89  97  101 103 107 109 113 127 131
137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223
227 229

```

Q. Give an alternative for multiple **if** statement in C.

Solution: A switch statement can be used as an alternative for multiple if or nested if statements.

Consider the following example: if(n == 1)

y = 1+x;

else

if(n == 2)

y = 1+x/n;

else

if(n == 3)

y = 1 +pow(x,n);

else

y = 1 + n * x;

The above program can also be written using a switch statement. switch(n)

```

{
case 1:  y = 1 + x; break;
case 2: y = 1 + x/n; break;

```

```
case 3: y = 1 + pow(x,n); break;
default: y = 1 + n * x; break;
}
```

Q. Write a C program, which accepts a number and prints the sum of digits of this number.

Solution:

```
/* Accept a number and sum up the digit */
/* for example, 173456=1+7+3+4+5+6=26 */
#include<stdio.h>
main( )
{
    int n, rem, quo, sum=0; printf("Enter a
integer number: \n");scanf("%d",&n);
    while(n>0)
    {
        rem=n%10;
        quo=n/10;
        sum=sum+rem;
        n=quo;
    }
    printf("The sum of digit of digits which you entered is =
%d",sum);
}
```

Q. What is the use of **break** statement?

Solution: A **break** statement is used to transfer the control to the end of a statement block. The next statement, which follows this statement block, will be executed. Consider the following example.

```
for(i = 1; i<= n; i++)
```