LOOPS IN C

by
Dr. Sumit Srivastava
Dept. of Computer Science & Engineering

Unit -II

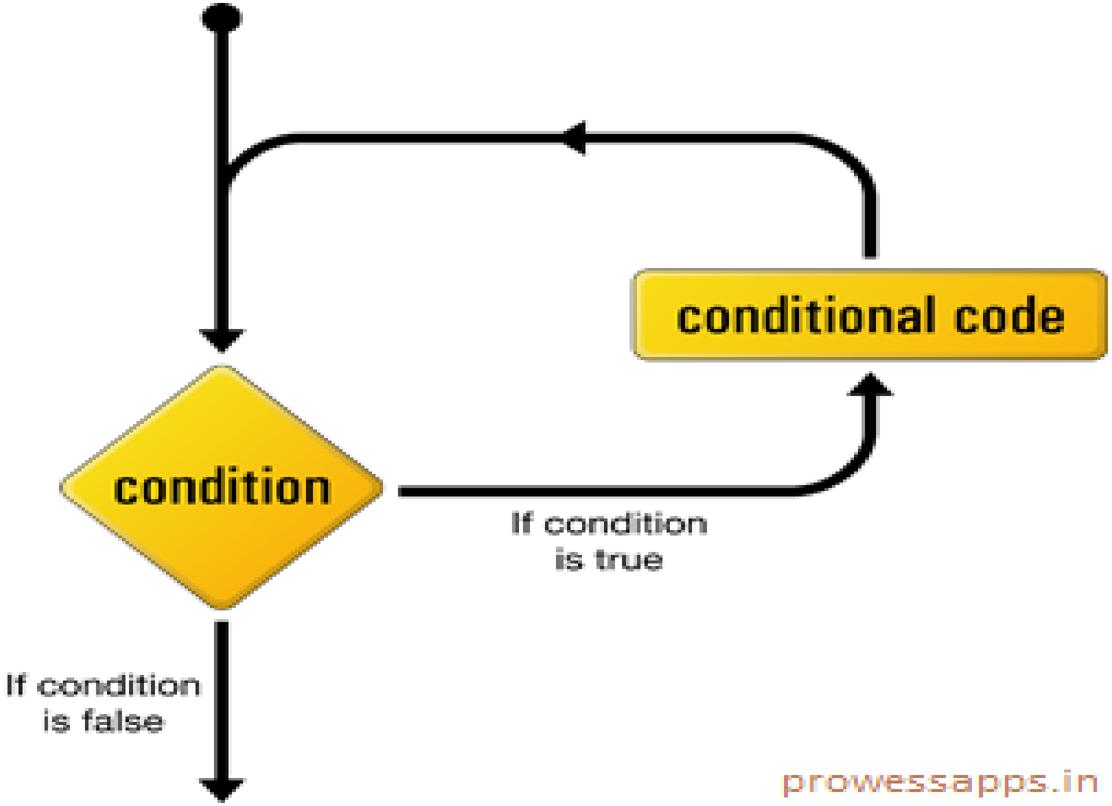
Loops in C Programming

- A loop statement allows us to execute a statement or group of statements multiple times.

- It repeats some portion of the program either a specified number of times or until a particular condition is being satisfied.

- Types of Iterative/Looping statements:
  - for loop
  - while loop
  - do-while-loop

Loops in C Programming

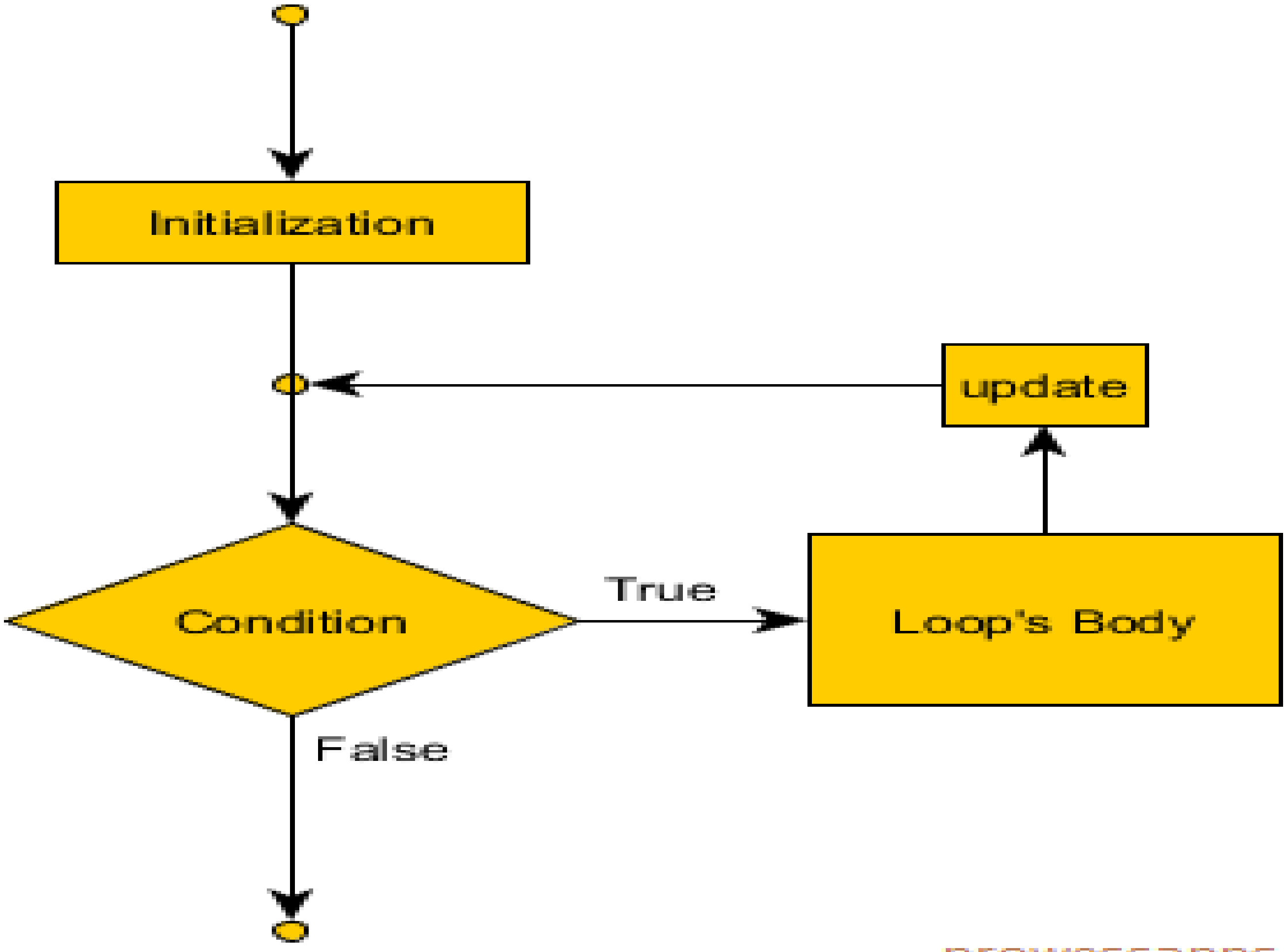


For loop in C

- 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:

```
for (initialization; condition; updation)
 {
      body;
  }
```

For loop



For loop

- expression1 is an initialization, expression2 is the conditional expression and expression3 is an updation.
- In the for loop, expression1 is used to initialize the variable, expression2 is evaluated and if the condition is true, then the body of for loop will be executed and then the statements under expression3 will be executed.
- This process is repeated as long as the for loop condition is true, once the condition is false control will return to the statements following the for loop and execute those statements.

Example:

WAP to print hello 5 times using for loop.

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

Output

```
1. Hello
2. Hello
3. Hello
4. Hello
5. Hello
```

While loop in C

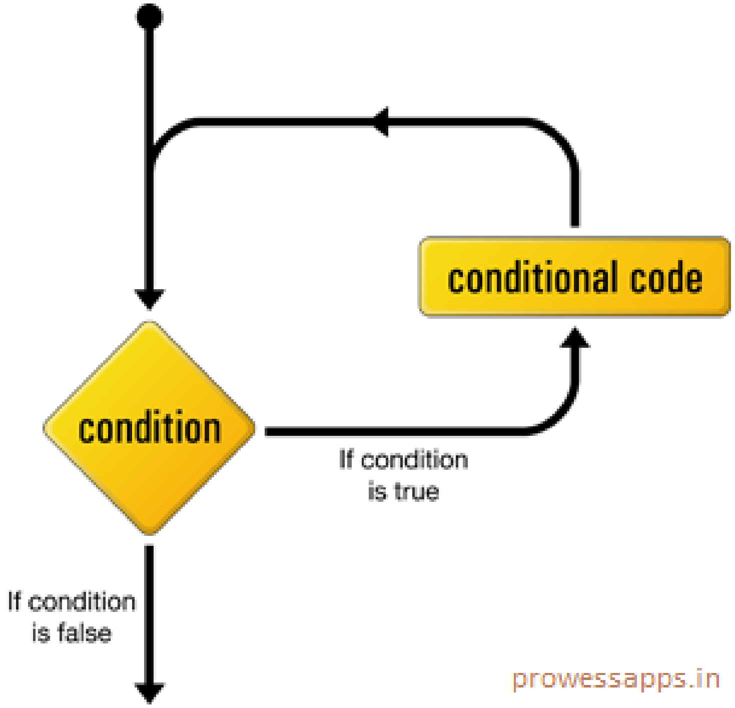
- Similar to for Loop, while statement creates a loop that repeats until the test expression becomes false.
- A while loop is also known as an entry loop because in a while loop the condition is tested first then the statements underbody of the while loop will be executed.
- If the while loop condition is false for the first time itself then the statements under the while loop will not be executed even once.

While loop in C

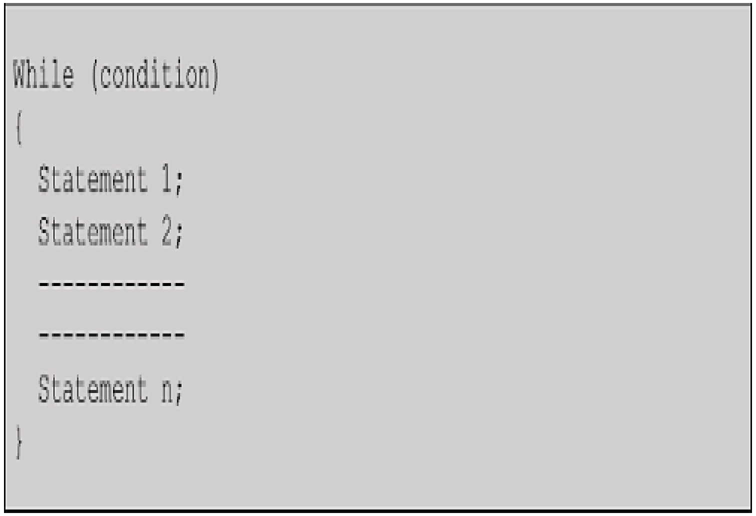
Syntax

```
initialization;
while(condition)
  {
      body;
      updation;
   }
```

Flow Diagram

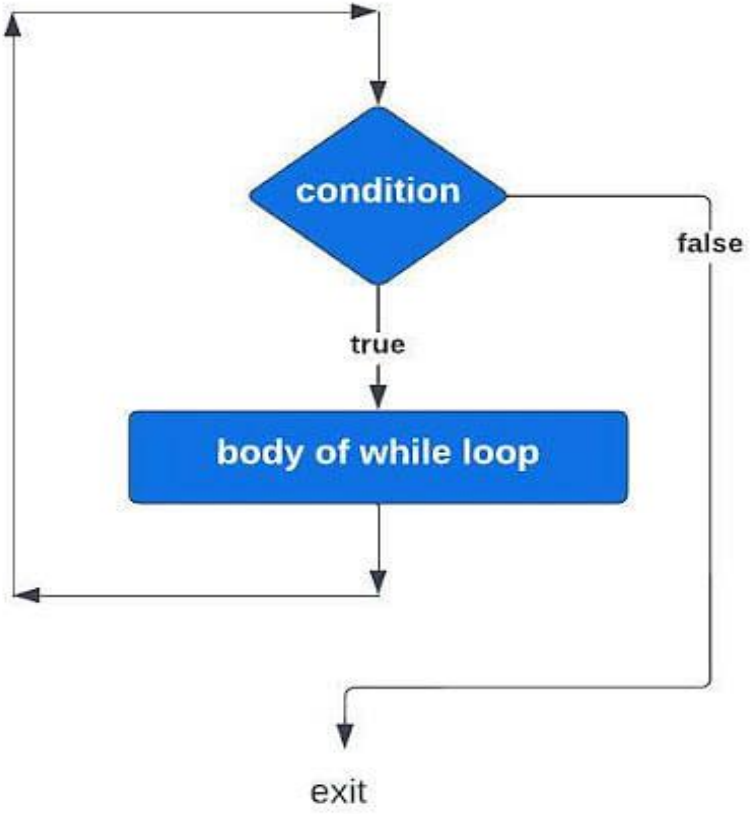


Syntax



```
While (condition)
{
  Statement 1;
  Statement 2;
  ------------
  ------------
  Statement n;
}
```

Flow Diagram



While loop in C: Example

WAP to print hello 5 times using while.

Output

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

```
1. Hello
2. Hello
3. Hello
4. Hello
5. Hello
```

do... while loop

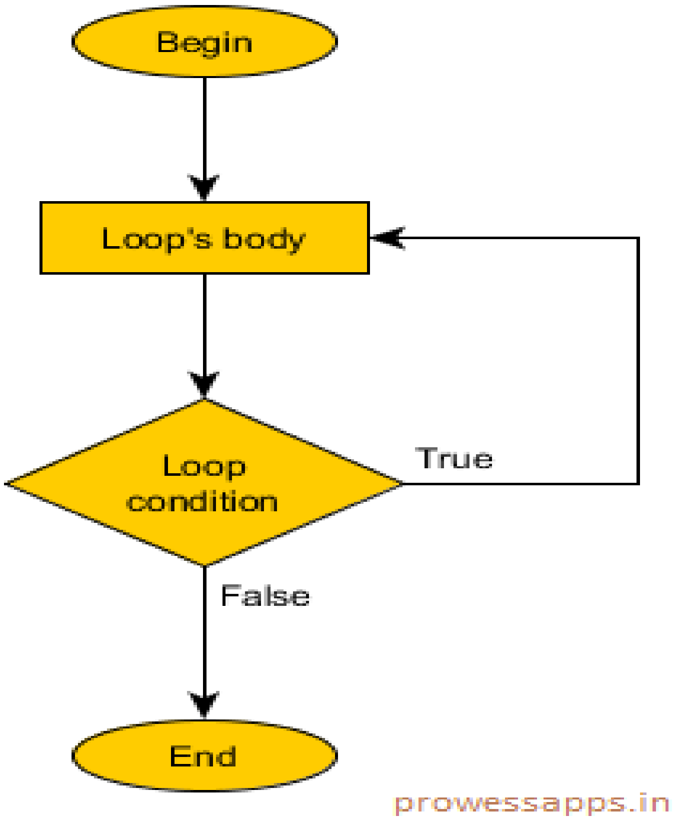
- Like while loop, do-while is also an iterative statement, but it tests the condition at the end of the loop body.
- The do-while is also known as an exit loop because in the do-while loop, the statements will be executed first and then the condition is checked.
- If the condition of the while loop is true then the body of the loop will be executed again and again until the condition is false. Once the condition is false, the control will transfer outside the do-while loop and execute statements followed soon after the do-while loop.

do... while loop

Syntax

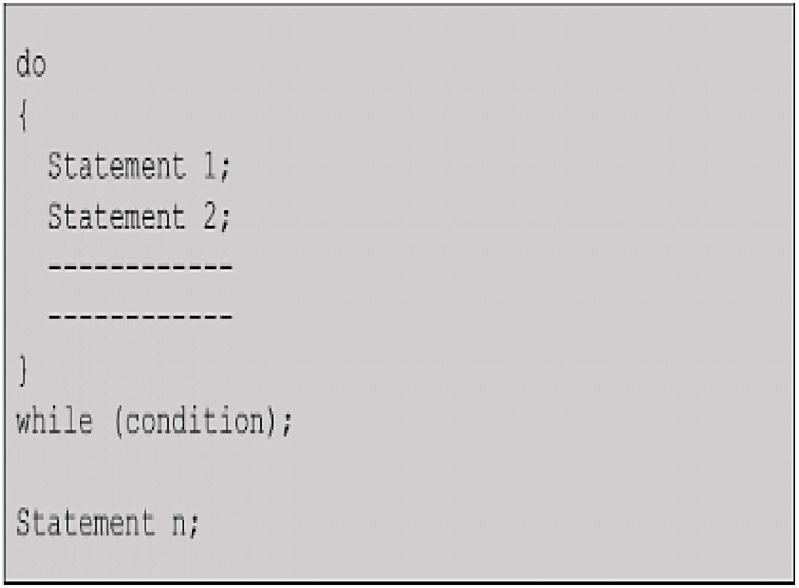
```
initialization;
do
  {
        body;
        updation;
   }
 while(condition);
```

Flow Diagram



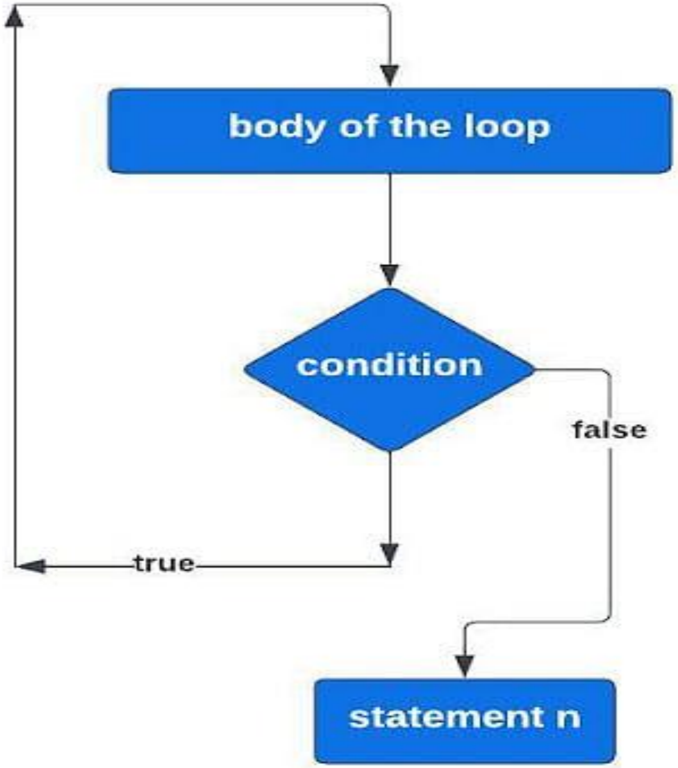
do... while loop

Syntax



```
do
{
   Statement 1;
   Statement 2;
   ------------
   ------------
}
while (condition);

Statement n;
```

Flow Diagram



do... while loop (Example)

WAP to print hello 5 times using do while.

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

Output

```
1. Hello
2. Hello
3. Hello
4. Hello
5. Hello
```

nested loops in C

- C programming allows to use one loop inside another loop.
- 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 for loop in C

□ 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 while loop

- □ The syntax for a nested while loop statement is as follows –

```
while(condition) {

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

nested do...while loop

- □ The syntax for a nested do...while loop statement is as follows −

```
do {
   statement(s);

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

}while( condition )
```

Loop Control 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.

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

Break Statment

- **Break :**

The **break** statement in C has the following two usage:

1. in loops

2. in switch – case

- The break statement stops the current iteration of loop and exit (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).

- Break, only terminates the current loop in which it occurs.

- It can be used to **terminate** a **case in switch**.

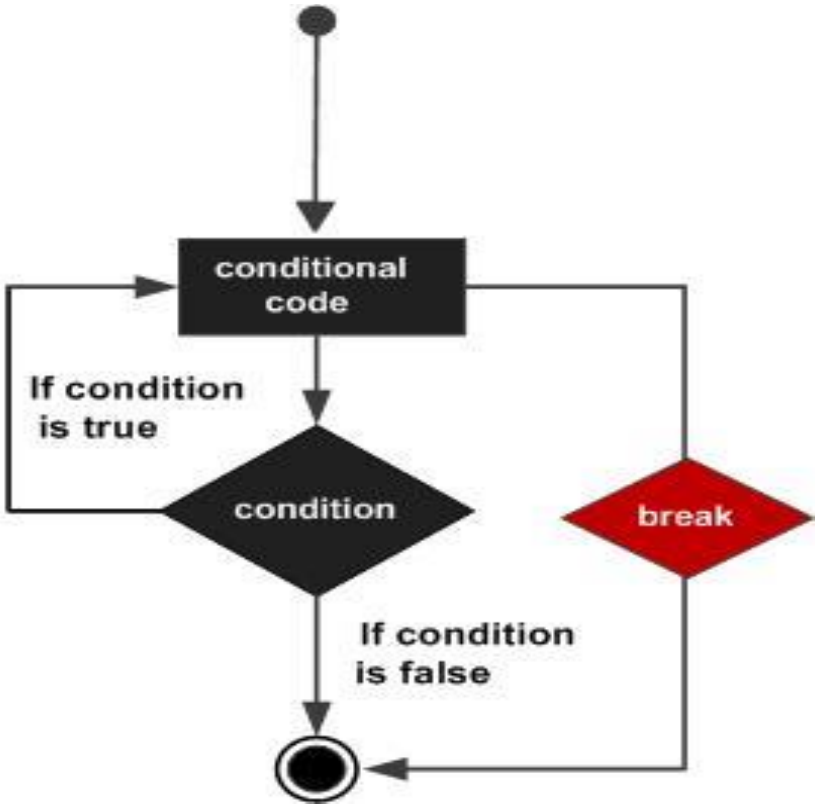
Break Statement

Syntax

```
break;
```

- 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.

Flow Diagram



Break: Example

Program

```
#include < stdio.h >
 int main( )
{
  int i;
  for (i=1; i<10; i++)
    {
         if(i==4)
              {
         break;
               }
         printf("%d. Hello",i);
     }
   return 0;
  }
```

Output

```
1. Hello
2. Hello
3. Hello
```

Break: 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++;

      if( a > 15) {
         /* terminate the loop using break statement */
         break;
      }
   }

   return 0;
}
```

Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Continue Statement

- The continue statement stops the current iteration of loop and continue the next iteration.

Continue Statement

- The continue statement stops the current iteration of loop and continue the next iteration.
- The continue statement in 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.

Continue Statement

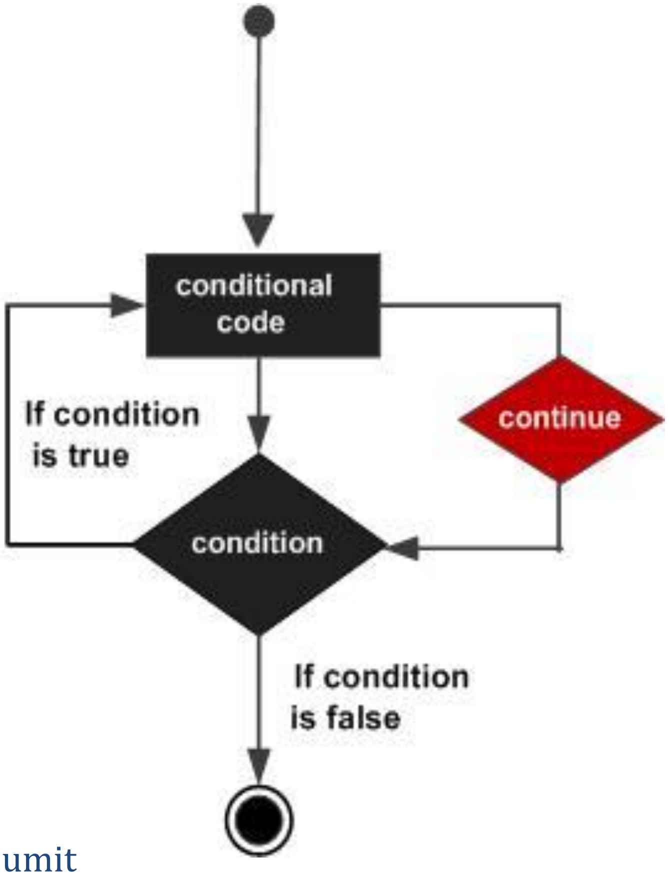
- For the for loop, continue statement causes the conditional test and increment portions of the loop to execute.
- For the while and do...while loops, continue statement causes the program control to pass to the conditional tests.

Continue : Example

Syntax

```
continue;
```

Flow Diagram



Continue : Example

Program

```
#include < stdio.h >
 int main( )
{
  int i;
  for (i=1; i<10; i++)
      {
         if (i==4 || i==7)
           {
             continue;
            }
          printf("%d. Hello",i);
        }
   return 0;
  }
```

Output

```
1. Hello
2. Hello
3. Hello
5. Hello
6. Hello
8. Hello
9. Hello
```

Continue : Example

```
#include <stdio.h>
int main () {

    /* local variable definition */
   int a = 10;

   /* do loop execution */
    do {

         if( a == 15) {
            /* skip the iteration */
            a = a + 1;
            continue;
          }

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

      }while( a < 20 );
      return 0;
    }
```

Output

```
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
```

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.

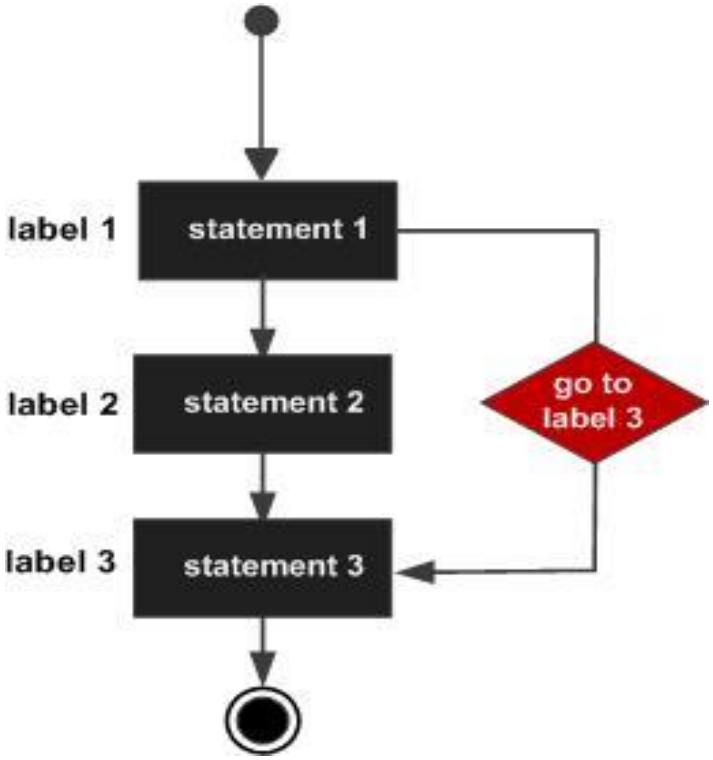
goto Statement

Syntax

```
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.

Flow Diagram



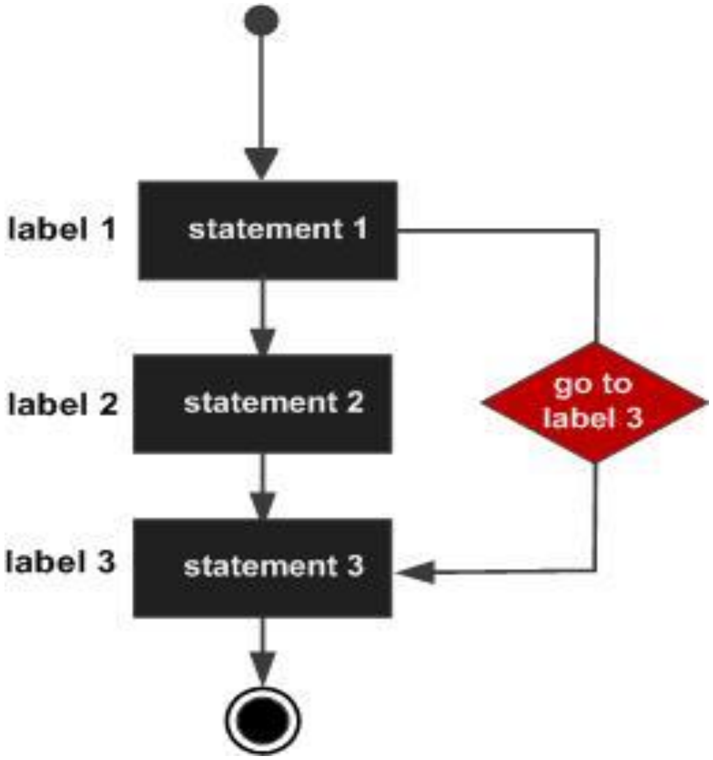
goto Statement

Syntax

```
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.

Flow Diagram



goto statement: Example

```
#include <stdio.h>
int main () {

    /* local variable definition */
   int a = 10;

   /* do loop execution */
    LOOP:do {

         if( a == 15) {
             /* skip the iteration */
            a = a + 1;
            goto LOOP;
               }

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

      }while( a < 20 );
      return 0;
    }
```

□ Output:

```
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
```

Input / Output in C

Input / Output in C

printf: This function is used to display output to the output screen. The general syntax of this function is as follows.

```
printf("format string",var_list);
```

TYPE FORMATTERS:

DATA_TYPE	**FORMATTER**
int	%d %X %x %o %u %p
float	%f
double	%lf %Lf
char	%c
string	%s

Input / Output in C

TYPE FORMATTERS

DATA_TYPE	FORMATTER
int	%d %X %x %o %u %p
float	%f
double	%lf %Lf
char	%c
string	%s

Escape Sequence

SEQ	DETAIL
\n	New Line
\t	Tab
\b	Back Space
\f	Form Feed
\r	Carriage Return
\a	Alarm
\\	Back Slash
\"	Double Quote

Input / Output in C

scanf: This function allows us to enter data from keyboard that will be formatted in a certain way.

The general form of scanf ( ) statement is as follows:

```
scanf ("format string",var_address);
```

Input / Output in C

Example

```
#include < stdio.h >
 int main( )
{
 int a, b;
 printf ("Enter value a: ");
 scanf ("%d", &a);
 printf ("Enter value b: ");
 scanf ("%d", &b);
 printf ("User Input is:\n");
 printf ("A=%d & B=%d", a, b);
 return 0;
}
```

OUTPUT

```
Enter value a: 17
Enter value b: 23

User Input is:
A=17 & B=23
```

Input / Output in C

- getchar: This function allows us to enter character from keyboard.
- The general form of getchar( ) statement is as follows:

```
char ch = getchar( );
```

Input / Output in C

Example

```
#include< stdio.h >
 int main( )
{
char choice;
printf("Enter Any Char: ");
choice = getchar();
printf("Input is: %c",choice);
return 0;
}
```

OUTPUT

```
Enter Any Char: F
Input is: F
```

THANK YOU