Iterative Statements in C

- There are three type of Loops available in 'C' programming language.
  - while loop
  - for loop
  - do.while

While Loop

The syntax of the while loop is:

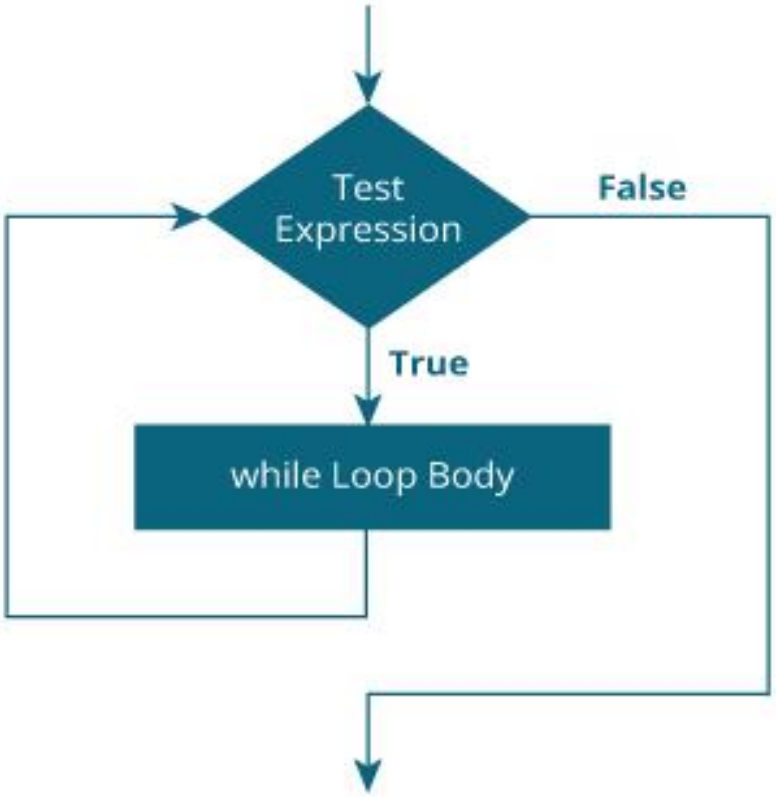
```
while (testExpression)
{
   // statements inside the body of the loop
}
```

While Loop

How while loop works?

- The while loop evaluates the test expression inside the parenthesis ().
- If the test expression is true, statements inside the body of while loop are executed. Then, the test expression is evaluated again.
- The process goes on until the test expression is evaluated to false.
- If the test expression is false, the loop terminates (ends).

While Loop



While Loop

```
// Print numbers from 1 to 5

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

    while (i < 6)
    {
        printf("%d\n", i);
        ++i;
    }

   return 0;
}
```

Output:
1
2
3
4
5

While Statement in Python and C

```
while condition expression:
        body of while
else:
        statement(s)
```

```
while (expression)
{
  // execute statements
}
```

do...while loop

- The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

- The syntax of the do...while loop is:

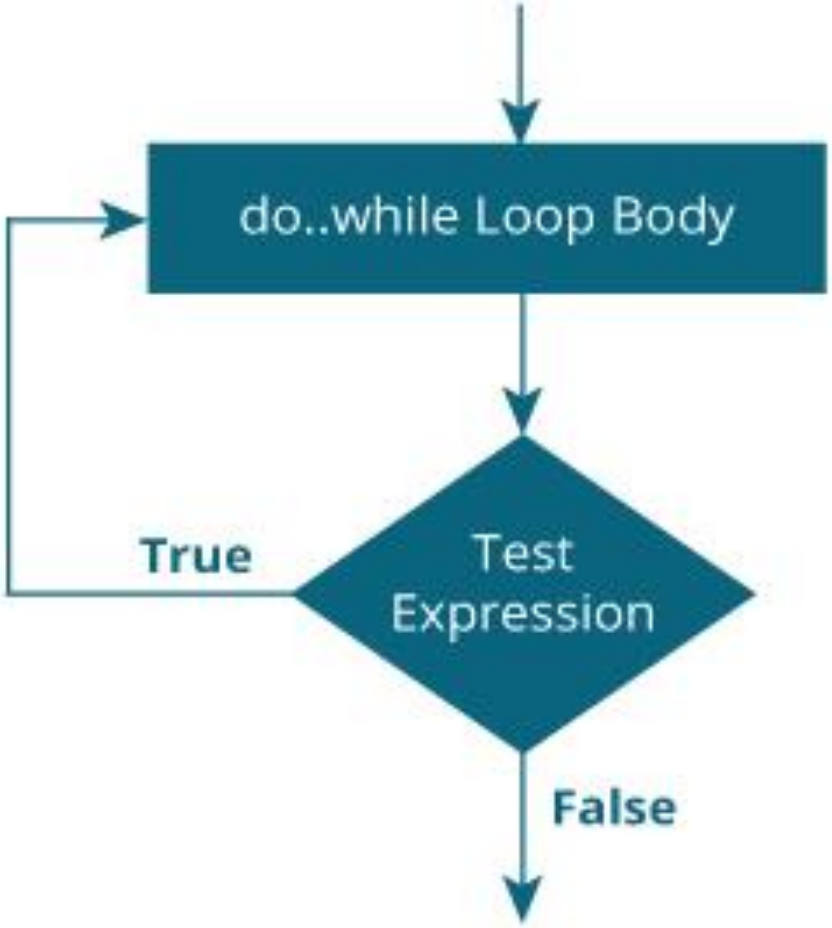
```
i=1;
do
{
  printf("%d",i);
  i++;
}
while (i<6);
```

do...while loop

How do...while loop works?

- The body of do...while loop is executed once. Only then, the test expression is evaluated.
- If the test expression is true, the body of the loop is executed again and the test expression is evaluated.
- This process goes on until the test expression becomes false.
- If the test expression is false, the loop ends.

do...while loop



do...while loop

```
// Program to add numbers until the user enters zero
#include <stdio.h>
void main()
{
   int number, sum = 0;
   // the body of the loop is executed at least once
   do
   {
      printf("Enter a number: ");
      scanf("%d", &number);
      sum += number;
   }
   while(number != 0);
   printf("Sum = %.d",sum);
}
```

Output:

```
Enter a number: 1
Enter a number: 2
Enter a number: 3
Enter a number: 4
Enter a number: 0
Sum = 10
```

For loop

The syntax of the for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
   // statements inside the body of loop
}
```

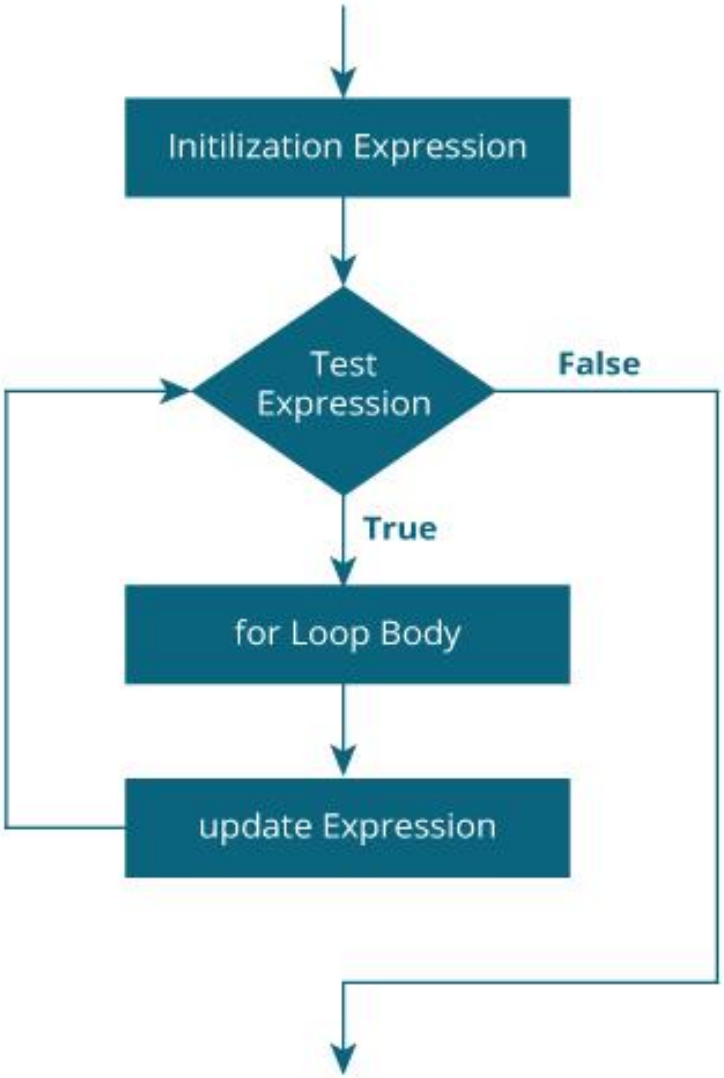
```
for(int i=1; i<3; i++)
{

}
```

For Loop

How for loop works?

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates. JAO SAO JUM YORM

For Loop


Initilization Expression
Test Expression
False
True
for Loop Body
update Expression

For loop

```
// Print numbers from 1 to 10
#include <stdio.h>

void main() {
  int i;

  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
}
```

Output

1 2 3 4 5 6 7 8 9 10

For Loop

```
// Program to calculate the sum of first n natural numbers
// Positive integers 1,2,3...n are known as natural numbers

#include <stdio.h>
void main()
{
   int num, count, sum = 0;
   printf("Enter a positive integer: ");
   scanf("%d", &num);

   // for loop terminates when num is less than count
   for(count = 1; count <= num; ++count)
   {
      sum += count;
   }
   printf("Sum = %d", sum);
}
```

For loop

for (initializationStatement; testExpression; updateStatement)

-All three components are optional
-But semicolons b/w the options are mandatory

Recall for loop Statement in Python

```
for val in sequence:
        body of for
else:
        statement(s)
```

```c
#include<stdio.h>
void main()
{
int i = 0;
for(;i<10;)
{
printf("Hello");
i++;
}
}
```

HelloHelloHelloHelloHelloHelloHelloHelloHelloHello

```c
#include<stdio.h>
void main()
{
for(;;)
printf("Hello");
}
```

- When the conditional expression is absent, it is assumed to be true.
- A loop becomes an infinite loop if a condition never becomes false.

```c
#include<stdio.h>
void main()
{
int i = 0;
for()
{
printf("Hello");
i++;
}
}
```

Error

GCD of Two Numbers

The greatest common divisor (GCD) of two integers is the product of the integers' common factors. Write a program that inputs two numbers and find their GCD by repeated division. For example, consider the numbers 252 and 735. find the remainder of one divided by the other.

$$
\begin{array}{r}
0 \\
735\overline{)\,252} \\
0 \\
\hline
252
\end{array}
$$

GCD of Two Numbers

Now we calculate the remainder of the old divisor divided by the remainder found

$$
\begin{array}{r}
2 \\
252\overline{)\,735} \\
504 \\
\hline
231
\end{array}
$$

Repeat the process until remainder is zero

The Divisor when remainder is zero is the GCD

$$
\begin{array}{r}
1 \\
231\overline{)\,252} \\
231 \\
\hline
21
\end{array}
\qquad
\begin{array}{r}
11 \\
21\overline{)\,231} \\
21 \\
\hline
21 \\
21 \\
\hline
0
\end{array}
$$

21 is the GCD

GCD problem

Input	Output	Logic Involved
Two numbers	GCD of the numbers	Euclidean algorithm, binary GCD algorithm, repeated division method

Algorithm to Find GCD

Step 1: Read the numbers from the user

Step 2: Let dividend = number1 and divisor = number2

Step 3: Repeat step 4 to step 6 while remainder not equal to zero

Step 4: remainder = number1 modulus number2

Step 5: dividend = divisor

Step 6: divisor = remainder

Step 7: GCD = divisor

Step 8: print GCD

Implementation

- We have to learn how to repeat statements
- In some cases the number of times to repeat a statement is known, in weather report example it is ten times we have to repeat some statements
- In some other cases the conditions are not direct as a number but as a terminating condition that may be based on I/O. In our GCD problem, the statements are to be repeated till reminder becomes zero

Break and Continue Statements

Break and Continue Statements

- Interrupt iterative flow of control in loops
- Break causes a loop to end
- Continue stops the current iteration and begin the next iteration

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

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

```
for (intial expression; test expression; update expression) {
    statement/s
    if (test expression) {
        break;
    }
    statements/
}
```

```
while test expression:
    body of while
    if condition:
        continue
    body of while

statement(s)
```

```
for var in sequence:
    body of for
    if condition:
        continue
    body of for

statement(s)
```

```
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, the loop terminates
#include <stdio.h>
void main() {
  int i;
  double number, sum = 0.0;
  for (i = 1; i <= 10; ++i) {
    printf("Enter a n%d: ", i);
    scanf("%lf", &number);

    // if the user enters a negative number, break the loop
    if (number < 0.0) {
      break;
    }
    sum += number; // sum = sum + number;
  }
  printf("Sum = %.2lf", sum);
}
```

```
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, it's not added to the
result
#include <stdio.h>
void main() {
  int i;
  double number, sum = 0.0;
  for (i = 1; i <= 10; ++i) {
    printf("Enter a n%d: ", i);
    scanf("%lf", &number);
   if (number < 0.0) {
      continue;
    }
    sum += number; // sum = sum + number;
  }
  printf("Sum = %.2lf", sum);
}
```

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

```c
//Program to find square root of a number
//Only positive numbers are allowed
#include<stdio.h>
#include<math.h>
void main()
{
int num = 0,counter;
double root;
//Loop to get ten numbers
for(counter=0;counter<10;counter++)
{
        scanf("%d",&num);
        //find root and print
        root = sqrt(num);
        printf("%.2f\n",root);

}
}
```

```c
//Program to find square root of a number
//Only positive numbers are allowed
#include<stdio.h>
#include<math.h>
void main()
{
int num = 0,counter;
double root;
//Loop to get ten numbers
for(counter=0;counter<10;counter++)
{
        scanf("%d",&num);
        //When number is less than zero
        if(num<0)
        {
        printf("Negative not allowed\n");
        //break loop
        break;
        }
        else
        {
        //Otherwise find root and print
        root = sqrt(num);
        printf("%.2f\n",root);
        }
}
}
```

```c
//Program to count non digits
#include<stdio.h>
#define MAX 10
void main()
{
int counter,non_Digits=0;
char ch;
for(counter=0;counter<MAX;counter++)
{
        //Read a character
        scanf("%c\n",&ch);
        //Check if the character is not digit
        if(isdigit(ch))
        {
        //Not a digit continue to read next character
        continue;
        }
        //If it is not a digit then increment the counter for non_Digits
        else
        non_Digits++;
}
printf("%d",non_Digits);
}
```

When to Use Which Loop ?

- If you know (or can calculate) how many iterations you need, then use a counter-controlled (for) loop.
- Otherwise, if it is important that the loop complete at least once before checking for the stopping condition,
- or if it is not possible or meaningful to check the stopping condition before the loop has executed at least once, then use a do-while loop.
- Otherwise use a while loop.

The comma operator

- C has a comma operator, that basically combines two statements so that they can be considered as a single statement.
- About the only place this is ever used is in for loops, to either provide multiple initializations or to allow for multiple incrementations.
- For example:

```
int i, j = 10, sum;
for( i = 0, sum = 0; i < 5; i++, j-- )
        sum += i * j;
```