Programming Fundamentals CT-175

Iterative Structures

Objectives

The objective of this lab is to familiarize students with iterative structures supported by C. By the end of this lab students will be able to write iterative programs by using simple and nested iterative structures.

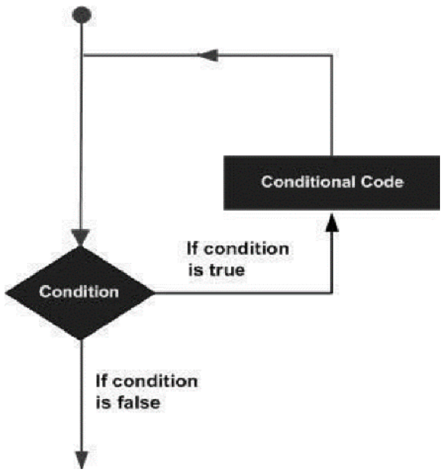
Tools Required

DevC++ IDE

Course Coordinator – Ms. Samia Masood Awan
Course Instructor – Ms. Samia Masood Awan
Lab Instructor – Ms. Samia Masood Awan
Department of Computer Science and Information Technology
NED University of Engineering and Technology

Iterative Structure in C

By iteration or repetition, we mean doing the same task again and again. Usually, we want to repeat the task until some condition is met or for a predefined number of iterations. In computer science this is commonly referred as loop. Loops are used to repeat a statement, a block, a function or any combination of these. A typical looping workflow is shown in the figure.



Loops are very powerful and important tool in programming. Consider adding the salary of 1000 employees of a company. Or finding out how many people have been vaccinated from a list of 30 million people, that is just the population of Karachi what about whole Sindh or Pakistan? One way is to copy paste the code or repeating the lines of code. Well, the easier way is using a loop. C programming language offers three kinds of iterative statements, that is, For, While, and Do-While.

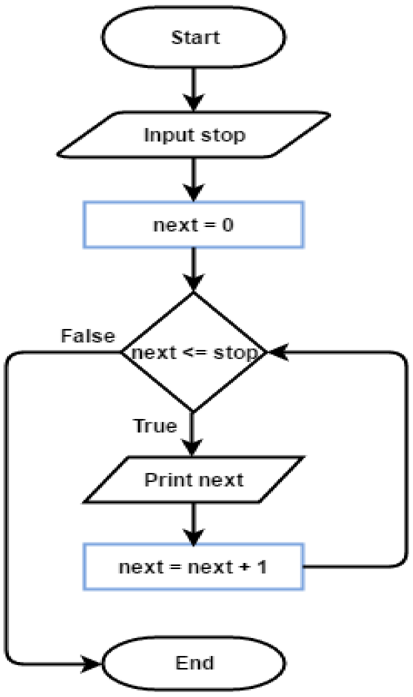
For Loop

The For loop is mostly used when we want to iterate for a specific number of times. Its syntax is as below

```
for ( initialization; repetition condition; update expression ){
        body of the loop;
}
```

- ✓ The initialization is an assignment statement that is used to initialize the loop control variable with a value. This is the first statement to be executed in the loop and only run once.
- ✓ The condition is a relational expression that determines when the loop exits. This runs after initialization, and verified before every iteration.
- ✓ The update expression either increment or decrement the loop control variable on each iteration.

```
#include <stdio.h>
int main ( ) {
  int stop, next;
  printf("Enter ending value:");
  scanf("%d",&stop);
  for(next=0;next<=stop;next++){
       printf("%d\t", next);
  }
  return 0;
}
```



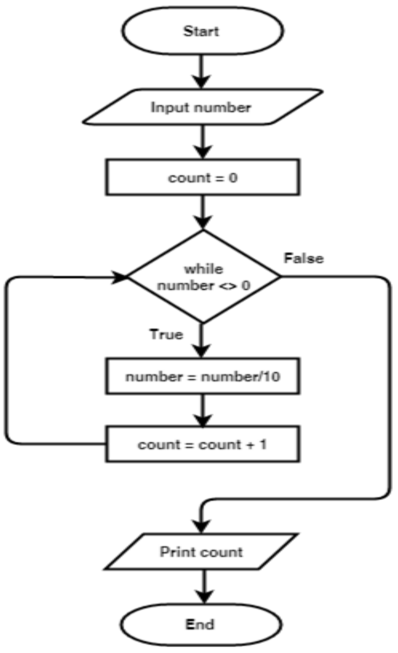
While Loop

While loop is used in situations when prior information about the number of iterations is not available. For example, as a game developer you don't have any knowledge how many times user will play the game so you want to loop over the choice if they want to play again or not. The syntax of while loop in C is given as follows.

```
initialization expression;
while ( loop repetition condition ) {
       Body of the loop;
       Update expression;
}
```

- ✓ The loop repetition condition (a condition to control the loop process) is tested; if it is true, the statement (loop body) is executed, and the loop repetition condition is retested.
- ✓ The statement is repeated as long as (while) the loop repetition condition is true. When this condition is tested and found to be false, the while loop is exited and the next program statement after the while statement is executed. If the condition is true forever the loop will run forever, we call such loop an infinite loop.
- ✓ It may not be executed at all if the condition is false right from start.

```
#include <stdio.h>
int main( ){
  int number=0, count =0;
  scanf("%d",&number);
  while (number != 0){
       number = number/10;
       count = count + 1;
  }
  printf("The number of digits are: %d",
  count);
  return 0;
}
```



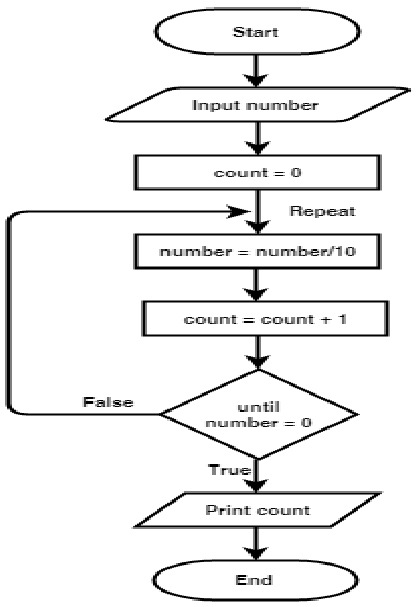
Do-While Loop

The do-while loop checks the loop repetition condition after running the body of the loop. This structure makes it most favorable in conditions where we are interested in running the body at least once. For example, as a web-developer you are writing a registration application. You would like to make sure if the username is already taken or not. Therefore, you will keep asking for a unique username until provided, in such situations you would want the user to provide the username first and then perform the check.

```
Initialization expression;
do{
     Body of the Loop;
     Update expression;
} while (loop repetition condition);
```

- ✓ First, the body of the loop is executed.
- ✓ Then, the loop repetition condition is tested if it is true, the body is again and the condition is retested until it remains true the loop continues. When this condition is tested and found to be false, the loop is exited and the next statement after the do-while is executed.

```
#include <stdio.h>
int main( ){
  int number=0, count =0;
  scanf("%d", &number);
  do{
      number = number/10;
      count = count + 1;
  } while (number != 0);
  printf("The number of digits are: %d",
  count);
  return 0;
}
```



Break Statement

Break statement is used to exit the body of the loop without meeting the loop repetition condition. The statements after the break never get executed and usually break is used to stop the loop before the loop termination condition is met. The controls execute the next statement after the loop body once it reaches the break statement in the loop body. Usually there is a condition upon which we want to exit the loop. For example, you have written a fund-raising application for a cancer patient, after the required funds are collected; you would like to break out of the loop without knowing how many iterations have passed.

```
for (initialization; condition; increment/decrement) {
         block of statements;
// this block will execute always with each iteration of loop
         if(condition){
                 block of statements;
                 break;
          }
         block of statements;
}
```

```
#include<stdio.h>
int main( ){
  int a, sum=0;
  while(1){
      scanf("%d",&a);
      if(a==-999) {
          printf("break statement executed!\n");
           break;
      }
      sum=sum+a;
  }
  printf("The sum is %d", sum);
  return 0;
}
```

Continue Statement

The continue statement is very powerful in situations where we want to execute some portion of the loop body and skip a statement or block of statements. The control skips the loop body as it reaches the continue statement and starts the next iteration. Usually there is a condition for which we want to skip certain statements. For example, as an AI developer you would work with a lot of datasets (e.g., thousands of images). Before training your AI model on these datasets you need to filter the corrupted or invalid images from the valid once. In such situations you can utilize continue on dependent situations instead of using multiple conditional statements.

```
for (initialization; condition; increment/decrement) {
         block of statements;
         if(condition){
                 block of statements;
                 continue;
          }
         block of statements;
}
```

```
#include<stdio.h>
int main( ){
  int i;
  for (int i=0; i<=10 ;i++){
       if((i==3)||(i==7)){
             printf(“Continue statement executed\n”);
             continue;
       }// if ends
       sum + = i;
  }// for ends
  printf(“The sum is %d”, sum);
  return 0;
}
```

Nested Loops

Nested loop means a loop statement inside another loop statement. That is why nested loops are also called as “loop inside loop”.

```
do{
      while(condition) {
            for ( initialization; condition; increment ) {
                  // statement of inside for loop
            }
           // statement of inside while loop
      }
     // statement of outer do-while loop
}while(condition);
```

```
// A program to print all prime factors of a number by using nested loops.
#include <math.h>
#include <stdio.h>
int main( ){
    int n = 315;
    while (n%2 == 0) {  //Print the number of 2s that divide n
        printf("%d", 2);
        n = n / 2;
    }
    for (int i=3;i<=sqrt(n);i+=2){// n must be odd at this point,  (Note i = i +2)
        while (n%i == 0) {  // while i divides n, print i and divide n
            printf("%d", i);
            n = n / i;
        }//while ends
    }//for ends
    if (n > 2)  //if n is a prime number greater than 2
    printf("%d",n);
    return 0;
}
```

Exercise

1. Write a program which will find the factorial of a given number. Exit the program if the input number is negative. **Test case:** Input number = 5,Factorial is=5*4*3*2*1
2. Write a program which will generate the Fibonacci series up to 1000. Also find the sum of the generated Fibonacci numbers divisible by 3, 5 and 7 only.
   Fibonacci series is:1 1 2 3 5 8 13 25..........
   <u>**Note:**</u> Do this task by using ***for loop*** and ***while loop***. Also identify which one is more efficient?
3. Write a program which will input a 5-digit number. If the sum of digits is even, find whether the input number is a prime or not. If the sum of digits is odd find, whether the number is palindrome or not?
   Example of prime number: A number which is only divisible by itself and 1 i.e., 7, 11, and13.
   Example of a Palindrome: A number whose reverse order is the same as the original number i.e., 11211, 44344.
4. Develop a user-registration system have the following options.
   a) Ask the user for a user-name (5 alphabets).
   b) Password should be 6 characters long with at least 1 numeric, 1 capital and 1 small letter.
   c) Display a "Account Created Successfully".
   d) Login the user with correct username and password.
   e) Display "Welcome username, you are now logged in".
   Note: Develop your application using break and continue statement.
5. (Calculating the Value of π) Calculate the value of π from the infinite series. Print a table that shows the value of π approximated by one term of this series, by two terms, by three terms, and so on. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14159?
6. (Diamond-Printing Program) Write a program that prints the following diamond shape. You may use printf statements that print either a single asterisk (*) or a single blank. Maximize your use of repetition (with nested for statements) and minimize the number of printf statements.

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