CL-1002 Programming Fundamentals

LAB - 06

Iterative Loops and 1D Arrays in C

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
Fall 2023

Learning Objectives:

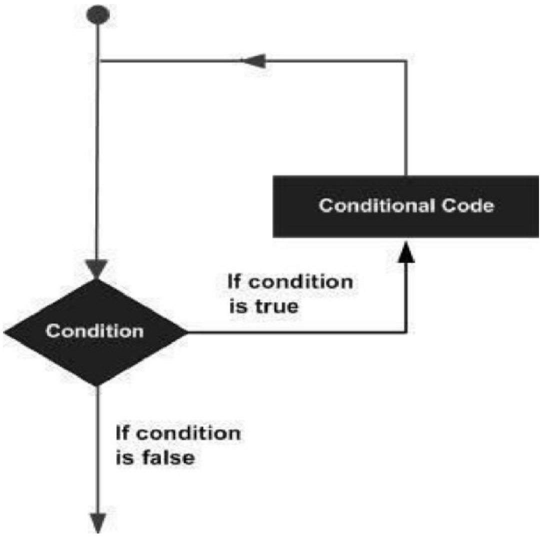
1. Iteration & iterative statements in C
2. While loop
3. Do-while loop
4. For loop
5. Continue Statement
6. Break Statement
7. 1D Arrays

Iterative Loops in C

1. Iteration and Types of Iterative 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 to as a **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 a very powerful and important tool in programming. Consider adding the salary of 1000 employees of a company. Or find out how many people have been vaccinated from a list of 30 million people, that is just the population of Karachi what about the whole of Sindh or Pakistan? One way is to copy-paste the code or repeat the lines of code. Well, the easier way is using a loop.



C programming language offers three kinds of iterative statements.

Types of Loops

1. While Loop
2. Do-while Loop
3. For Loop

2. While Loop

The while loop is usually used when we **don't know the number of iterations in advance**.

Application: As a game developer you don't have any knowledge of how many times users will play the game so you want to loop over the choice if they want to play again or not.

a. C- Syntax

```
while ( loop repetition condition )
// Body of the loop to be executed as long as the condition is true
{

        Body of the loop.

 }
```

b. Interpretation:

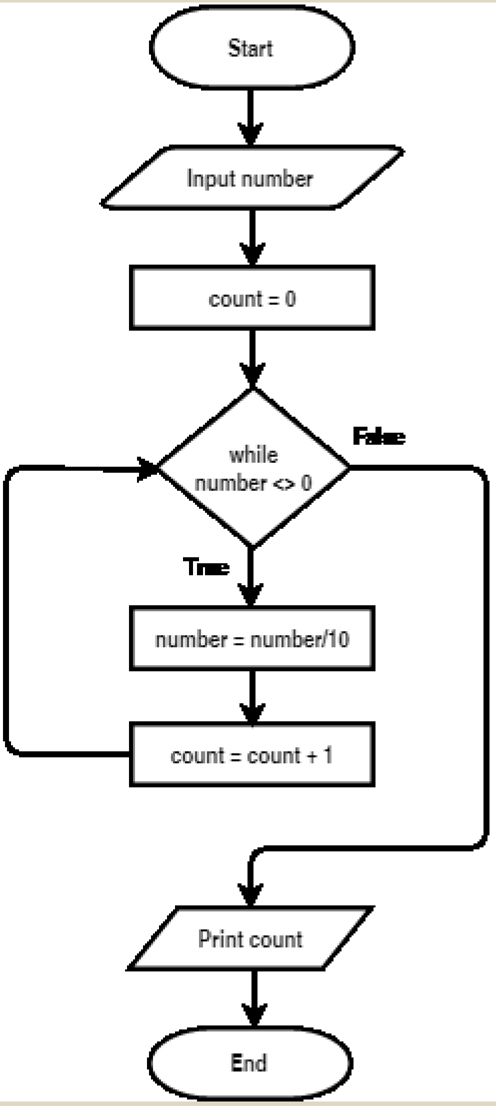
- The loop repetition condition (a condition to control the loop process) is tested; if it is true, the statement (loop body) is executed, and the loop repetition condition is retested.
- The statement is repeated as long as (while) the loop repetition condition is true. When this condition is tested 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 a loop an infinite loop.
- It may not be executed at all if the condition is false right from the start.

c. Example:

To count number of digits in a given integer.

Input: 13542

Output: 5

ALGORITHM	FLOWCHART
1. Start 2. Input number 3. count = 0 4. While number != 0 do 5. number=number/10 6. count = count + 1 7. While End 8. Print count 9. End	

C-IMPLEMENTATION

```
#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;
}
```

```
while ( loop repetition
condition )
{
        //Statement_Block;
}
```

3. 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**.

Application: 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.

a. C- Syntax

```
do
//Body of the Loop to be executed at least once
{

        Body of the Loop;

}
while ( loop repetition condition );
```

b. Interpretation

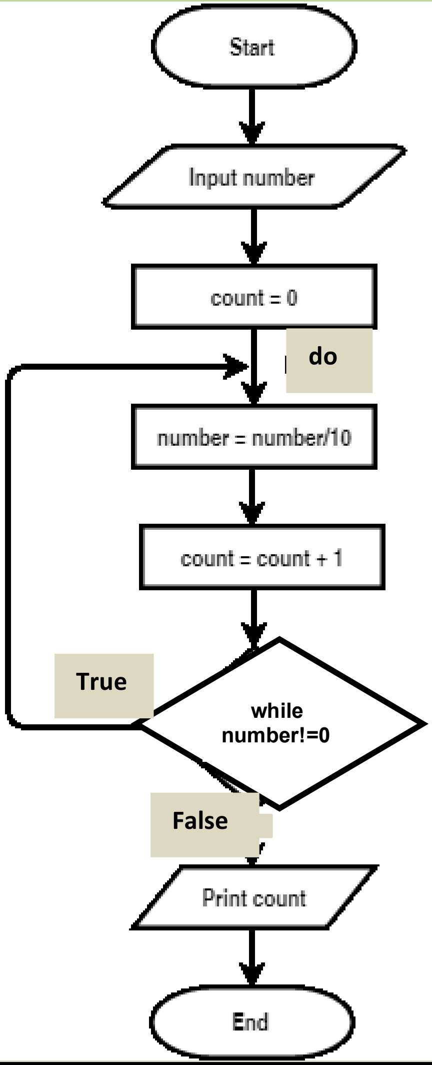
- First, the body of the loop is executed.
- Then, the loop repetition condition is tested if it is true, the body is again and the condition is retested until it remains true the loop continues. When this condition is tested and found to be false, the loop is exited and the next statement after the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

c. Example

Repeating the same example with do-while. Count number of digits in a given integer.

Input: 144452

Output: 6

ALGORITHM	FLOWCHART
1. Start 2. Input number 3. count = 0 4. do 5. number=number/10 6. count = count + 1 7. while number != 0 8. Print count 9. End	 Start Input number count = 0 do number = number/10 count = count + 1 True while number!=0 False Print count End

C-IMPLEMENTATION

```
#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;
}
```

```
do
{
        //Statement_Block;

} while ( loop repetition
condition );
```

4. For Loop

The for loop is mostly used when we want to **iterate for a specific number of times**.

Application: You are developing a first-person shooter game in the beginning of the mission user is given 3 lives only, you would like the character to revive if killed for 3 times only.

a. C- Syntax

```
for (initialization expression; loop repetition condition; update expression)
//Body of the Loop to be executed in each iteration
{
                Body of the Loop;
}
```

b. Interpretation

- 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.

c. Example

Print the numbers from 0 to desired value as shown below.

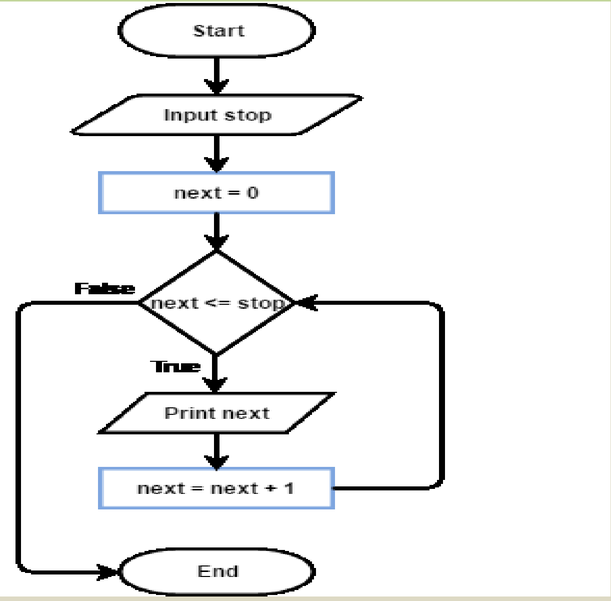
Input: 10

Output: 0 1 2 3 4 5 6 7 8 9 10

ALGORITHM

```
1. Start
2. Input stop
3. FOR next = 0 to stop
4. Print next
5. next = next + 1
6. END FOR
7. End
```

FLOWCHART



C-IMPLEMENTATION

```
#include <stdio.h>

int main ()
{
int stop, next;
printf(“Enter ending value:”);
scanf(“%d”,&stop);

  for(next = 0 ; next <= stop ; next=next+1)


{
     printf(“%d\t”, next);
  }

  return 0;
}
```

```
for (initialization expression
;
loop repetition condition ;
update expression )
{
        //Statement_Block;
}
```

Comparison of Loops

	for	**while**	**do-while**
Initialization of condition variable	In the parenthesis of the loop.	Before the loop.	Before the loop or in the body of the loop.
Test condition	Before the body of the loop.	Before the body of the loop.	After the body of the loop.
Updating the condition variable	After the first execution.	After the first execution.	After the first execution.
When to use	for is generally used when there is a specific number of iterations	while is generally used when the number of iterations is not known in advance.	Do-while is a loop with a post-condition. It is needed in cases, when the loop body is to be executed at least once.

5. 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.

Application: 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.

a. C Syntax

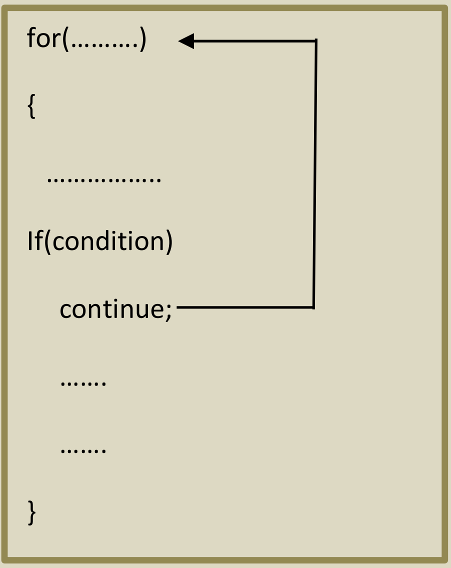
```
for (initialization; condition; increment/decrement) {

    block of statements;    // this block will execute always with each iteration of loop
    continue;
    block of statements;    // this block will be skipped.

}
```

b. Example:

```
#include<stdio.h>
int main()
{
     int i;
     for (int i=0; i<=10 ;i++)
     {
        if((i==3)||(i==7))
        {
                 continue;
        }
   printf("The sum is %d\n", i);
  }
}
```



for(..........)

{

....

If(condition)

continue;

....

....

}

6. Break Statement

Break statement is used to **exit the body of the loop** without meeting the loop repetition condition. The statement after the break never gets 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.

Application: 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.

a. C Syntax

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

    block of statements;    // this block will execute always with each iteration of loop
    break;
    block of statements;    // this block never gets executed.

}
```

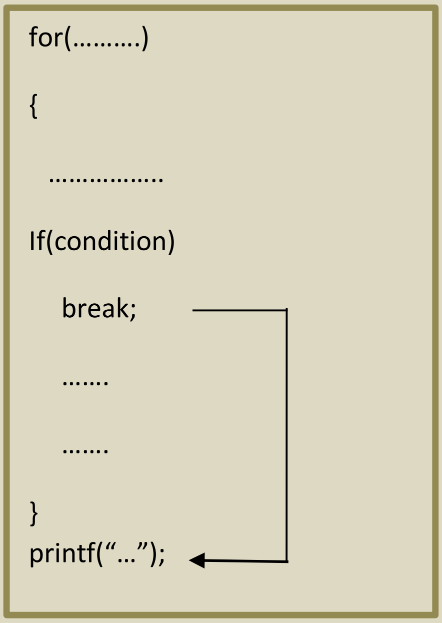
b. Example

```
#include <stdio.h>
int main()
{
   int a, sum = 0;
   for (;;)// infinite loop
   {
         scanf("%d", &a);
         if (a == -999)
         {
             break;
         }
         sum += a;
   }

  printf("The sum is %d", sum);

  return 0;

}
```



```
for(……….)

{

  ……………..

If(condition)

   break;

   …….

   …….

}
printf("…");
```

Arrays in C

An array is a collection of **data items of the same type**.

Significance of Array

Programming problems can be **solved efficiently by grouping** the data items together in the main memory rather than allocating an individual memory cell for each variable.

For Example: A program that processes exam scores for a class, would be easier to write if all the scores were stored in one area of memory and were able to be accessed as a group. C allows a programmer to group such related data items together into a single composite data structure called array.

7. One-Dimensional Arrays

In one-dimensional array, the components are arranged in the form of a list.

SYNTAX:

element-type aname [size]; /* uninitialized */
element-type aname [size] = { initialization list }; /* initialized */

INTERPRETATION:

- The general uninitialized array declaration allocates storage space for array aname consisting of size memory cells.
- Each memory cell can store one data item whose data type is specified by element-type (i.e., double, int , or char).
- The individual array elements are referenced by the subscripted variables aname [0] , aname [1] , . . , aname [size −1] .
- A constant expression of type int is used to specify an array's size. In the initialized array declaration shown, the size shown in brackets is optional since the array's size can also be indicated by the length of the initialization list .
- The initialization list consists of constant expressions of the appropriate element-type separated by commas.
- Element 0 of the array being initialized is set to the first entry in the initialization list , element 1 to the second, and so forth.

MEMORY REPRESENTATION

All arrays consist of **contiguous memory locations**. The lowest address corresponds to the first element and the highest address to the last element.

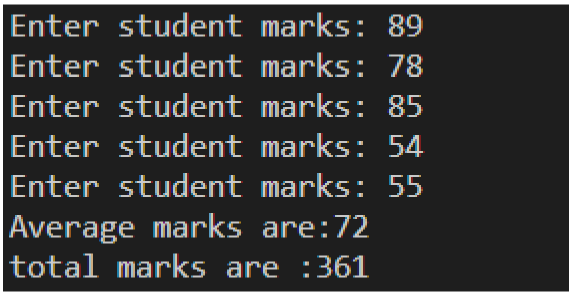
double x[5] = { 5.0, 2.0, 3.0, 1.0, -4.5};

Array x

x[0]	x[1]	x[2]	x[3]	x[4]

x[0]	x[1]	x[2]	x[3]	x[4]
5.0	2.0	3.0	1.0	-4.5

EXAMPLE 01:



```
Enter student marks: 89
Enter student marks: 78
Enter student marks: 85
Enter student marks: 54
Enter student marks: 55
Average marks are:72
total marks are :361
```

```
/* 1D Array 1st Example */

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

	int avg, sum = 0;
	int i;
	int marks[5]; // array declaration

	for(i = 0; i<=4; i++){
		printf("Enter student marks: ");
		scanf("%d", &marks[i]); /* stores the data in array*/
	}
	for(i = 0; i<=4; i++)
		sum = sum + marks[i]; /* reading data from array */

		avg = sum/5;
		printf("Average marks are:%d\n", avg);
		printf("total marks are :%d\n", sum);
		return 0;

	}
```

Example 02:

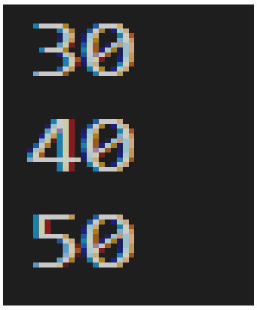
```
#define size 5
#include <stdio.h>
int main()
{
int i;

/*
    int arrOfNumbers[5];
   arrOfNumbers[0] = 10;
   arrOfNumbers[1] = 20;
   arrOfNumbers[2] = 30;
   arrOfNumbers[3] = 40;
   arrOfNumbers[4] = 50;
*/

   // 2nd alternattive
   //int arrOfNumbers[5] = {10,20,30,40,50};

   // 3rd alternative declaration
   //int arrOfNumbers[] = {10,20,30,40,50};

   // 4th alternative using macro
   int arrOfNumbers[size] = {10,20,30,40,50};
```



```
30
40
50
```

```
   for(i = 0; i < 5; i++)
   {
         /* The braces are not necessary; we use them to make the
         code clearer. */
         if(arrOfNumbers[i] > 20)
         printf("%d\n", arrOfNumbers[i]);
   }
   return 0;}
```

Example 03:

C does not check for index out of bound, it can be done by the programmer itself.

```
#include <stdio.h>
int main(void)
{
   int i, j = 30, arrOfNumbers[3];
   for(i = 0; i < 4; i++)
         arrOfNumbers[i] = 100;
         printf("%d\n", j);
return 0;
} /* The program does not throw any error. It will print 30 once.
```

Example 04:

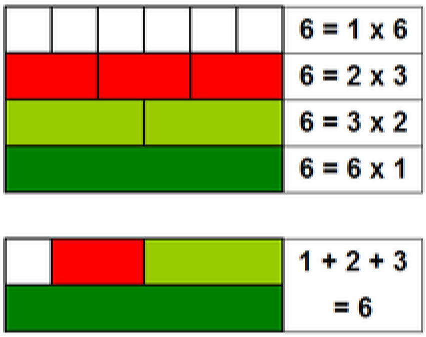
```
#include <stdio.h>
int main(void)
{
   int i, arrOfNumbers[10] = {0};
   for(i = 0; i < 10; i++){
         arrOfNumbers[++i] = 20;
      // arrOfNumbers[i] = 20;
      printf("The elements of array are:\t%d\n",arrOfNumbers[i]);
   }


   return 0;
}
```

EXERCISE

QUESTION# 01

Write a C Program that takes any number from the user and identifies if the number is a perfect number or not.



QUESTION# 02

Write a program that will generate the Fibonacci series up to 10000. Also find the sum of the generated Fibonacci numbers divisible by 3, 5 or 7 only.

An example of the Fibonacci series is: 1 1 2 3 5 8 13 25..........

Note: Do this task by using a ***for loop*** **DO NOT** use arrays for this.

QUESTION# 03

Write a C Program to compute the LCM and GCD of two numbers.

QUESTION# 04

Consider Two integers a and b taken as input from the user. Using Loops iterate the value of a till the value of b.

If the value of a<=9 the output should correspond to the English representation of the numbers i.e., 8=Eight, 9=Nine etc.

If the iteration exceeds 9 then the programs should print if the exceeded number is even or odd.

Example:

Input= 8,11

Output= Eight, Nine, Even, Odd

QUESTION# 05

Write a C program that produces the following output:

```
0       0       0       0
        1       1
2       2       2       2
        3       3
4       4       4       4
        5       5
6       6       6       6
```

Note: Only use single loops **(No Nested Loops)**

QUESTION# 06

Write a program for a match-stick game between the computer and a user. Your program should ensure that the computer always wins. Rules for the game are as follows:

1. There are 21 matchsticks.
2. The computer asks the player to pick 1,2,3 or 4 match sticks.
3. After the person picks, the computer does its picking.
4. Whoever is forced to pick up the last matchstick loses the game.

QUESTION# 07

Write a C Program that takes a user input array and prints the sum of its elements.
Input: {1,2,3,4,5,6,7,8,9}
Output: 45

QUESTION# 08

Write a program in C to read n number of values in an array and display it in reverse order.
Input: {1,2,3,4,5,6,7,8,9}
Output: 9 8 7 6 5 4 3 2 1

QUESTION# 09

Write a C Program to find the minimum and maximum number in an array.
Input: {4,1,6,8,10,21,8,9,2,6}
Output:

Minimum Number = 1
Maximum Number = 21