Iqra National University Peshawar Pakistan
Department of Computer Science

Spring Semester, Final Term Exam, June 2020

Paper :	**Programming Fundamentals**	Date and Starting Time:	**26/June/2020, 9:00 am**
Program:	**BS (CS & SE)**	Uploading Date and End Time:	**26/June/2020, 3:00 pm**
Teacher Name:	**Dr. Fazal-e-Malik**	Marks	**50**

NAME : AMAD AFRIDI

ID : 13119

CLASS : BS SE B

SUBJECT : PROGRAMMING FUNDAMENTAL

Note: Attempt all Questions. Help can be taken from net where ever is required.

Q.1

a) What is the purpose of ***if statement***? Discuss its two different forms with examples. **5**

b) Write a C++ program to read two numbers from keyboard and then find the LARGEST number of them. **5**

Answer1:

Q1.A: What is the purpose of *if statement*? Discuss its two different forms with examples.

AnsA: This is one of the most frequently used loop in C Programming.
Syntax of for loop:

```
for (initialization; condition test; increment or decrement)
{
       //Statements to be executed repeatedly
}
```

Step 1: First initialization happens and the counter variable gets initialized.
Step 2: In the second step the condition is checked, where the counter variable is tested for the given condition, if the condition returns true then the C statements inside the body of for loop gets executed, if the condition returns false then the for loop gets terminated and the control comes out of the loop.
Step 3: After successful execution of statements inside the body of loop, the counter variable is incremented or decremented, depending on the operation (++ or –).

Example of For loop

```
#include <stdio.h>
int main()
{
   int i;
   for (i=1; i<=3; i++)
   {
       printf("%d\n", i);
   }
   return 0;
}
```

Q1.B: **Write a C++ program to read two numbers from keyboard and then find the LARGEST number of them.**

AnsB:

```
#include <iostream>
```

```
using namespace std;

int main()
{
    int num1, num2;
    cout<<"Enter first number:";
    cin>>num1;
    cout<<"Enter second number:";
    cin>>num2;
    if(num1>num2)
    {
```

```
        cout<<"First number "<<num1<<" is the largest";
    }
    else
    {
        cout<<"Second number "<<num2<<" is the largest";
    }
    return 0;
}
```

Q.2 a) What are the Logical Operators? Explain them

b) Write a C++ program to get Temperature in Fahrenheit ***F*** and then find the Atmosphere according to the below rules:

- If temperature ***F*** is above 40 degree Fahrenheit then display……..Very Hot.
- If temperature ***F*** is between 35 & 40 degree Fahrenheit then display……Tolerable.
- If temperature ***F*** is between 30 & 35 degree Fahrenheit then display……Warm.
- If temperature ***F*** is less than 30 degree Fahrenheit then display……..……Cool.

Answer2:

Q2.A: What are the Logical Operators? Explain them

AnsA:

The Logical Operators

Logical operators are mainly used to control program flow. Usually, you will find them as part of an `if`, a `while`, or some other control statement

The Logical operators are:

`op1 && op2`
-- Performs a logical AND of the two operands.
`op1 || op2`
-- Performs a logical OR of the two operands.
`!op1`
-- Performs a logical NOT of the operand.

The concept of logical operators is simple. They allow a program to make a decision based on multiple conditions. Each operand is considered a condition that can be evaluated to a true or false value. Then the value of the conditions is used to determine the overall value of the `op1 operator op2` or `!op1` grouping. The following examples demonstrate different ways that logical conditions can be used.

The `&&` operator is used to determine whether both operands or conditions are true `and.pl`.

For example:

```
if ($firstVar == 10 && $secondVar == 9) {

    print("Error!");

};
```

If either of the two conditions is false or incorrect, then the print command is bypassed.

The || operator is used to determine whether either of the conditions is true.

For example:

```
if ($firstVar == 9 || $firstVar == 10) {

    print("Error!");
```

If either of the two conditions is true, then the print command is run.

Caution If the first operand of the || operator evaluates to true, the second operand will not be evaluated. This could be a source of bugs if you are not careful.

For instance, in the following code fragment:

```
if ($firstVar++ || $secondVar++) { print("\n"); }
```

variable `$secondVar` will not be incremented if `$firstVar++` evaluates to true.

The `!` operator is used to convert true values to false and false values to true. In other words, it inverts a value. Perl considers any non-zero value to be true-even string values. For example:

```
$firstVar = 10;

$secondVar = !$firstVar;


if ($secondVar == 0) {

   print("zero\n");

};
```

is equal to *`0-`* and the program produces the following output:

```
zero
```

You could replace the 10 in the first line with "ten," 'ten,' or any non-zero, non-null value.

Q2.B: Write a C++ program to get Temperature in Fahrenheit *F* and then find the Atmosphere according to the below rules:

- **If temperature *F* is above 40 degree Fahrenheit then display……Very Hot.**
- **If temperature *F* is between 35 & 40 degree Fahrenheit then display……Tolerable.**
- **If temperature *F* is between 30 & 35 degree Fahrenheit then display……Warm.**

If temperature *F* is less than 30 degree Fahrenheit then display……Cool

AnsB:

```
//C++ program for converting degree Celsius into Fahrenheit and vice versa

#include<iostream>

using namespace std;



int main()

{

   float fahr, cel;

   char option;



   cout << "Choose from following option:" << endl;

   cout << "1. Celsius to Fahrenheit." << endl;

   cout << "2. Fahrenheit to Celsius." << endl;

   cin >> option;
```

```
//option for converting celsius into fahernheit
if (option == '1')
{
   cout << "Enter the temperature in Celsius: ";
   cin >> cel;

   fahr = (1.8 * cel) + 32.0; //temperature conversion formula
   cout << "\nTemperature in degree Fahrenheit: " << fahr << " F" << endl;
}
//option for converting Fahrenheit into Celsius
else if (option == '2')
{
   cout << "Enter the temperature in Fahrenheit: ";
   cin >> fahr;

   cel = (fahr - 32) / 1.8; //temperature conversion formula
   cout << "\nTemperature in degree Celsius: " << cel << " C" << endl;
}
else
   cout << "Error Wrong Input." << endl;
```

```
   return 0;

}
```

Q.3 a) What does ***Looping*** mean? Explain different loops in C++.

b) Write a C++ program to read a number from keyboard and then determine whether it is ***Even or Odd*** number?

Answer3:

Q3.A: What does *Looping* mean? Explain different loops in C++.

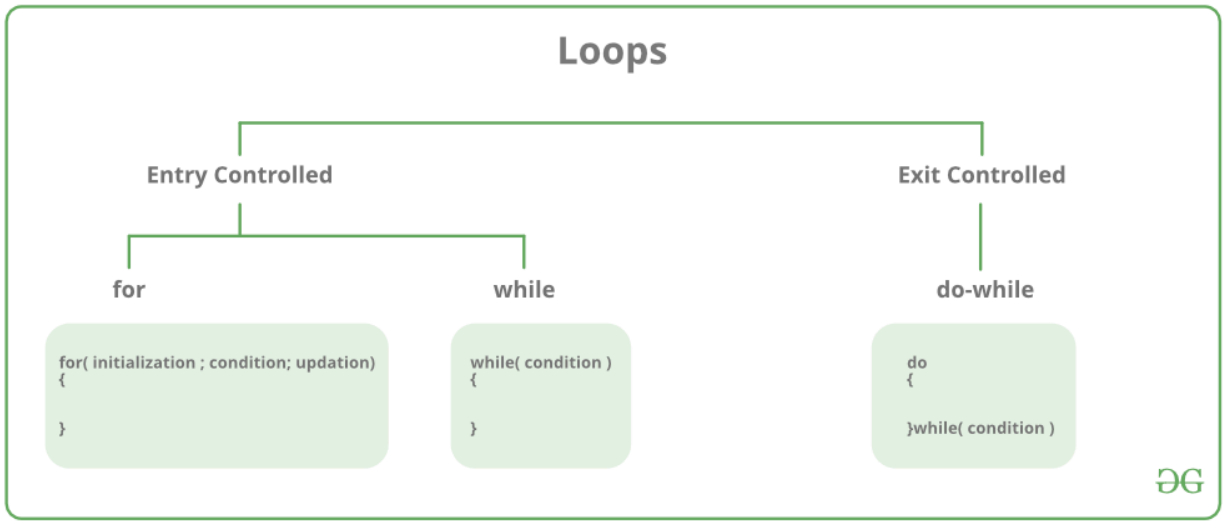
AnsA:

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below.
In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.

- An operation is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.
- **Counter not Reached:** If the counter has not reached the desired number, the next instruction in the sequence returns to the first instruction in the sequence and repeat it.
- **Counter reached:** If the condition has been reached, the next instruction “falls through” to the next sequential instruction or branches outside the loop.

There are mainly two types of loops:

1. **Entry Controlled loops**: In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.
2. **Exit Controlled Loops**: In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false. **do – while loop** is exit controlled loop.



for Loop

A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

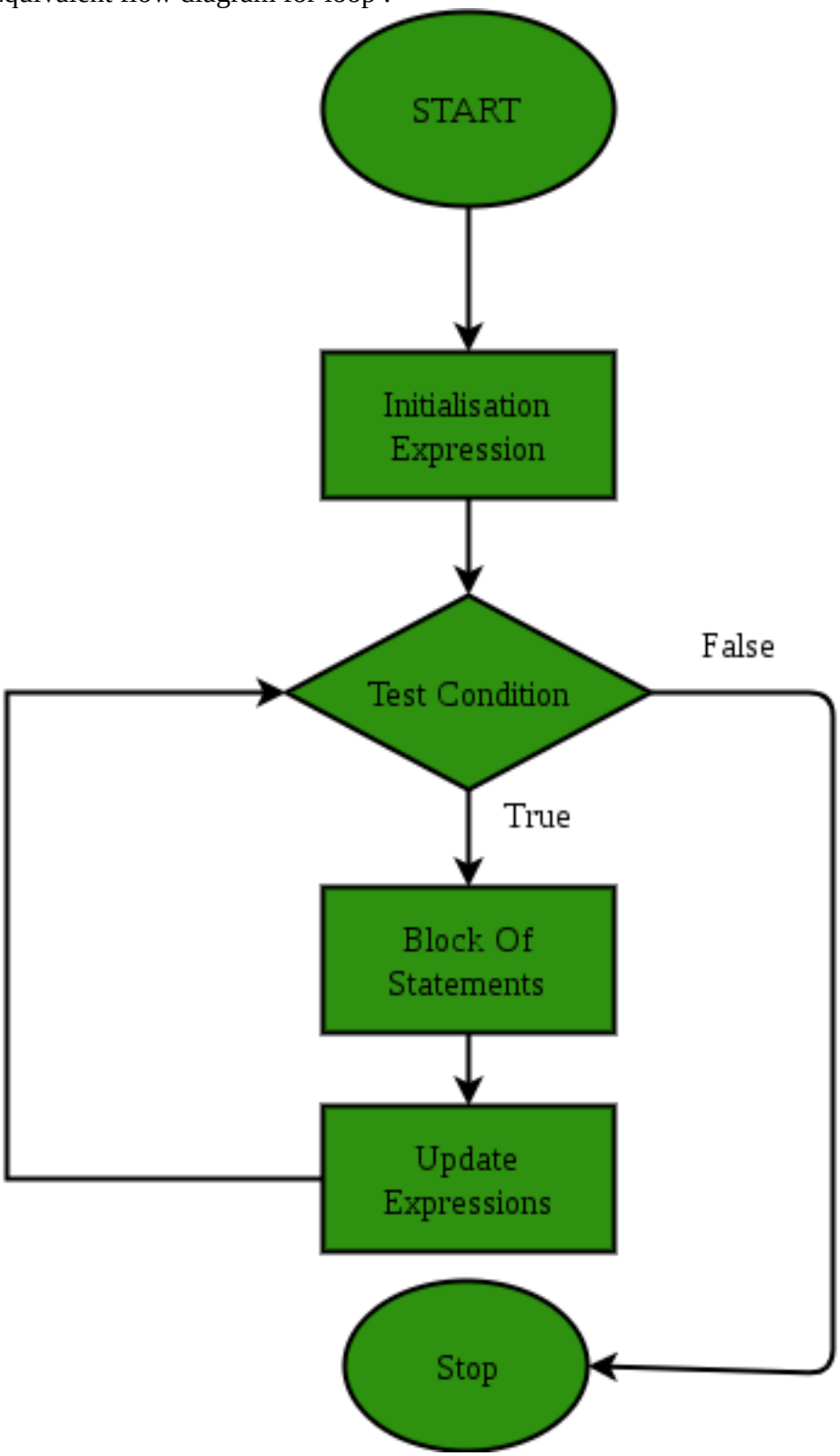
Syntax:

```
for (initialization expr; test expr; update expr)
{
     // body of the loop
     // statements we want to execute
}
```

In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated . Steps are repeated till exit condition comes.

- **Initialization Expression**: In this expression we have to initialize the loop counter to some value. for example: int i=1;
- **Test Expression**: In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop. For example: i <= 10;
- **Update Expression**: After executing loop body this expression increments/decrements the loop variable by some value. for example: i++;

Equivalent flow diagram for loop :



Example:

- C
- C++

filter_none
edit
play_arrow
brightness_4

```
// C program to illustrate for loop
#include <stdio.h>

int main()
{
    int i=0;

    for (i = 1; i <= 10; i++)
    {
        printf( "Hello World\n");
    }

    return 0;
}
```

Output:

```
Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World
```

While Loop

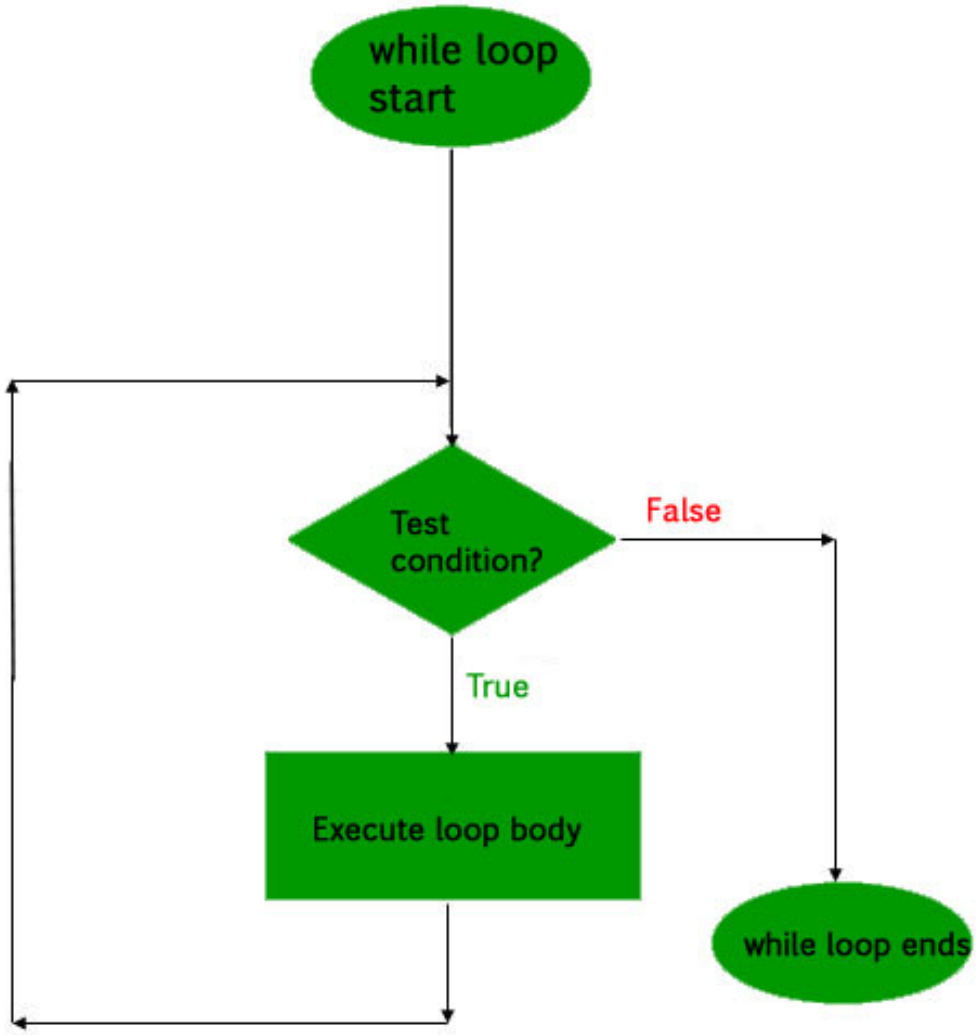
While studying **for loop** we have seen that the number of iterations is known beforehand, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where we do not know the exact number of iterations of loop beforehand. The loop execution is terminated on the basis of test condition.

Syntax:
We have already stated that a loop is mainly consisted of three statements – initialization expression, test expression, update expression. The syntax of the three loops – For, while and do while mainly differs on the placement of these three statements.

```
initialization expression;
while (test_expression)
{
   // statements

  update_expression;
}
```

Flow Diagram:



Example:

- C

- C++

filter_none
edit
play_arrow
brightness_4

```
// C program to illustrate while loop
#include <stdio.h>

int main()
{
    // initialization expression
    int i = 1;

    // test expression
    while (i < 6)
    {
        printf( "Hello World\n");

        // update expression
        i++;
    }

    return 0;
}
```

Output:

```
Hello World

Hello World

Hello World

Hello World

Hello World
```

do while loop

In do while loops also the loop execution is terminated on the basis of test condition. The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body, i.e do while loop is exit controlled whereas the other two loops are entry controlled loops.

Note: In do while loop the loop body will execute at least once irrespective of test condition.

Syntax:

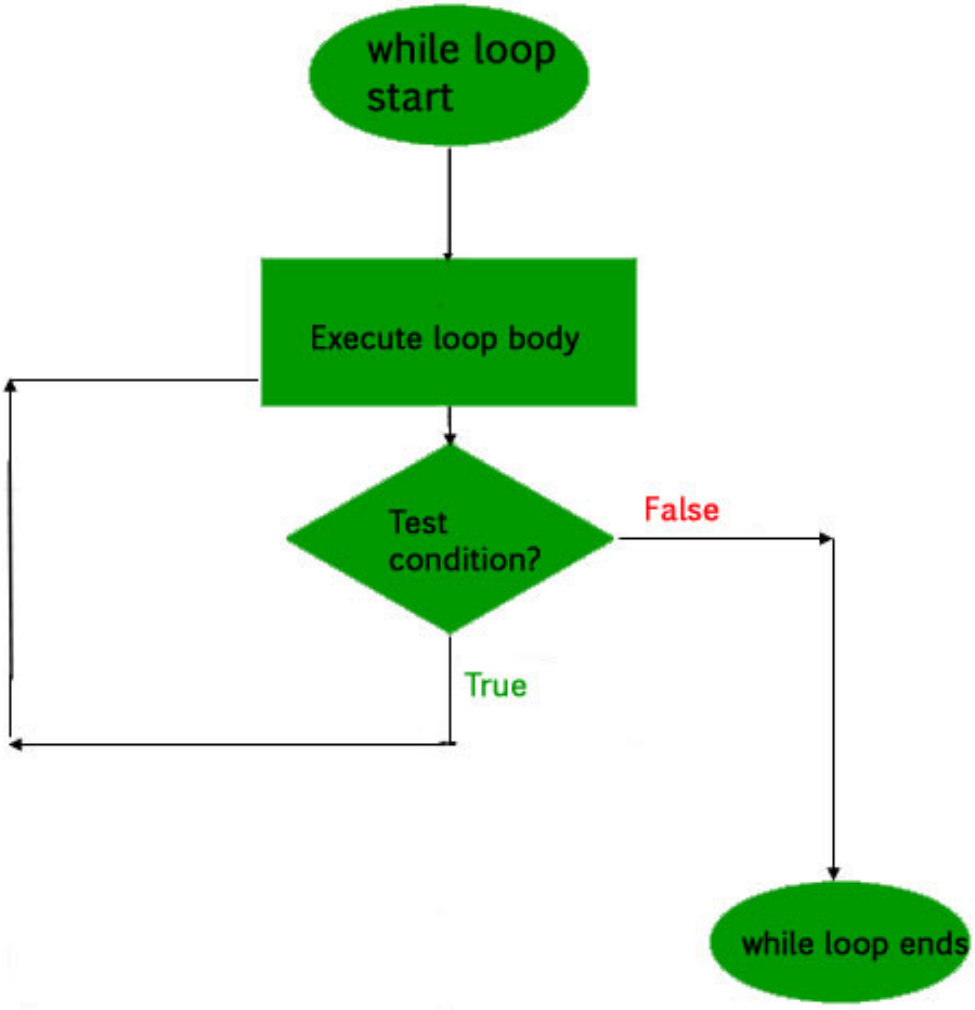
```
initialization expression;
do
{
   // statements

   update_expression;
```

```
} while (test_expression);
```

Note: Notice the semi – colon(“;”) in the end of loop.

Flow Diagram:



Example:

- C
- C++

filter_none
edit
play_arrow
brightness_4

```
// C program to illustrate do-while loop
#include <stdio.h>

int main()
```

```
{
    int i = 2; // Initialization expression

    do
    {
        // loop body
        printf( "Hello World\n");

        // update expression
        i++;

    }  while (i < 1);   // test expression

    return 0;
}
```

Output:

```
Hello World
```

In the above program the test condition (i<1) evaluates to false. But still as the loop is exit – controlled the loop body will execute once.

What about an Infinite Loop?

An infinite loop (sometimes called an endless loop) is a piece of coding that lacks a functional exit so that it repeats indefinitely. An infinite loop occurs when a condition always evaluates to true. Usually, this is an error.

- C
- C++

filter_none
edit
play_arrow
brightness_4

```
// C program to demonstrate infinite loops
// using for and while
// Uncomment the  sections to see the output

#include <stdio.h>

int main ()
{
    int i;

    // This is an infinite for loop as the condition
    // expression is blank
    for ( ; ; )
    {
       printf("This loop will run forever.\n");
    }
```

```
    // This is an infinite for loop as the condition
    // given in while loop will keep repeating infinitely
    /*
    while (i != 0)
    {
        i-- ;
        printf( "This loop will run forever.\n");
    }
    */

    // This is an infinite for loop as the condition
    // given in while loop is "true"
    /*
    while (true)
    {
        printf( "This loop will run forever.\n");
    }
    */
}
```

Output:

```
This loop will run forever.

This loop will run forever.

.................
```

Q3.B: Write a C++ program to read a number from keyboard and then determine whether it is *Even or Odd* number?

AnsB:

```
#include <iostream>
using namespace std;

int main()
{
```

```
    int n;

    cout << "Enter an integer: ";
    cin >> n;

    if ( n % 2 == 0)
        cout << n << " is even.";
    else
        cout << n << " is odd.";

    return 0;
}
```

Q.4 a) A) What is the purpose of using ***break and continue statements***?

B) Write a C++ program to find the sum of the following numbers:

1+2+3+……..+10

Answer4:

Q4.A: What is the purpose of using *break and continue statements*?

AnsA:

A)

The break statement in C

In any loop `break` is used to jump out of loop skipping the code below it without caring about the test condition.

It interrupts the flow of the program by breaking the loop and continues the execution of code which is outside the loop.

The common use of break statement is in switch case where it is used to skip remaining part of the code.

```
for (int-exp; test-exp; update-exp)

{

  statement1;

  if (condition)

     break;

  statement2;

}
```

The continue statement in C

Like a `break` statement, `continue` statement is also used with `if` condition inside the loop to alter the flow of control.

When used in `while`, `for` or `do...while` loop, it skips the remaining statements in the body of that loop and performs the next iteration of the loop.

Unlike `break` statement, `continue` statement when encountered doesn't terminate the loop, rather interrupts a particular iteration.

```
while (test_condition)

{

  statement1;

  if (condition )

     continue;
```

```
  statement2;

}
```

Q4.B:Write a C++ program to find the sum of the following numbers:
1+2+3+……..+10

AnsB:

Code to add 1+2+3….10

```
#include <iostream>
using namespace std;
int main()
{
    int i,sum=0;
    cout << "\n\n Find the first 10 natural numbers:\n";
    cout << "---------------------------------------\n";
    cout << " The natural numbers are: \n";
    for (i = 1; i <= 10; i++)
    {
        cout << i << " ";
           sum=sum+i;
    }
     cout << "\n The sum of first 10 natural numbers: "<<sum << endl;
}
```

Q.5 What is an array? Explain On-Dimensional and Two-Dimensional Arrays with examples. **10**

Answer5:

Array:

- Offers a simple way of grouping like variables for easy access
- It is a group of elements having same data type
- An array is a collective name given to a group of 'similar quantities'
- Arrays in C share a few common attributes
    - Variables in an array share the same name
    - Variables in an array share the same data type
    - Individual variables in an array are called *elements*
    - Elements in an array are accessed with an index number

```
main( )
{
    int x ;
    x = 5 ;
    x = 10 ;
    printf ( "\nx = %d", x );
}
```

- Ordinary variables are capable of holding only one value at a time
- There are situations in which we would want to store more than one value at a time in a single variable

Array Declaration

A one-dimensional array is a structured collection of components (often called array elements) that can be accessed individually by specifying the position of a component with a single index value.

Here is the syntax template of a one-dimensional array declaration:

```
DataType ArrayName [ConstIntExpression];
```

In the syntax template,

Data Type describes what is stored in each component of the array. Array components may be of any type, but for now we limit our discussion to simple data types (e.g. integral and floating types).

ConstInt Expression indicates the size of the array declared. That is, it specifies the number of array components in the array. It must have a value greater than 0. If the value is *n*, the range of the index values is 0 to *n-1*. For example, the declaration

```
int number[50];
```

creates the number array which has 50 components, each capable of holding one int value. In other words, the number array has a total of 50 components, all of type int

Two dimensional (2D) arrays in C programming with example

An array of arrays is known as 2D array. The two dimensional (2D) array in C programming is also known as matrix. A matrix can be represented as a table of rows and columns. Before we discuss more about two Dimensional array lets have a look at the following C program.

Simple Two dimensional(2D) Array Example

For now don't worry how to initialize a two dimensional array, we will discuss that part later. This program demonstrates how to store the elements entered by user in a 2d array and how to display the elements of a two dimensional array.

```
#include<stdio.h>
int main(){
   /* 2D array declaration*/
   int disp[2][3];
   /*Counter variables for the loop*/
   int i, j;
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("Enter value for disp[%d][%d]:", i, j);
         scanf("%d", &disp[i][j]);
      }
   }
   //Displaying array elements
   printf("Two Dimensional array elements:\n");
```

```
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("%d ", disp[i][j]);
         if(j==2){
            printf("\n");
         }
      }
   }
   return 0;
}
```

Output:

```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
Enter value for disp[0][2]:3
Enter value for disp[1][0]:4
Enter value for disp[1][1]:5
Enter value for disp[1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

…….. THE END ……