Lab 05

Objectives

After performing this lab, students will be able to

- Understand loops and create programs using them

Loops

Imagine a situation where you want to perform any task several times. For a simple example, suppose you want to print “Hello World” 10 times each on a separate line. Then, you would write ***cout*** 10 times (one ***cout*** for each line). Something like as shown below:

```
cout << "Hello World\n";
cout << "Hello World\n";
cout << "Hello World\n";
cout << "Hello World\n";
cout << "Hello World\n";
cout << "Hello World\n";
cout << "Hello World\n";
cout << "Hello World\n";
cout << "Hello World\n";
cout << "Hello World\n";
```

Now, imagine, you want to print something 5000 or 1 million times! Writing 5000 or 1 million lines would not be feasible. To tackle such situations, programming languages provide **loops** (also called as ***repetitive*** or ***iterative*** statements).

A loop is a program construct that repeats a statement or group of statements multiple times We write the statement/statements that are to be repeated inside the body of the loop. Each repetition of the loop is called an **iteration**.

There are two main categories of loops:

- Counter-controlled loops
- Sentinel-controlled loops

Counter-controlled loops

A counter-controlled loop is also known as definite repetition loop, since the number of iterations is known before the loop begins to execute.

The counter-controlled loop has the following components:

- a control variable
- the part that updates the control variable (usually the increment or decrement operation but can be any other operations as well like *, /) at each iteration of the loop.
- the loop terminating condition that checks if looping should continue.

Any task involving definite iteration can be solved using a counter-controlled loop for example printing the first 10 natural numbers.

The **for** loop is a common counter-controlled loop. Its general syntax is given below:

```
for (Initialization_Action; Boolean_Expression; Update_Action)
    Body_Statement
```

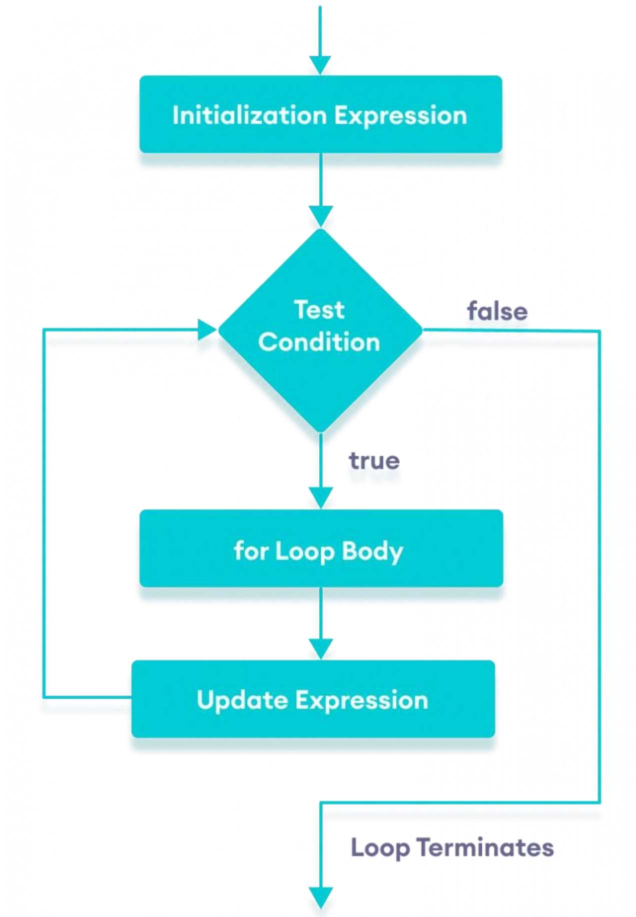
OR

```
for(initialization; condition; increment/decrement) {
    //body of the loop
}
```

Here

- ***initialization*** - initializes variables and is executed only once
- ***condition*** – controls the execution of the loop. If it is **true**, the body of for loop is executed. If it is **false**, the loop will not run (or when the condition becomes **false**, loop will stop executing).
- ***update/increment/decrement*** - updates the value of initialized variables and again checks the condition to determine if the loop should run once more or not.

Flowchart of for loop in C++



Example:

```
for (int i = 1; i <= 10; i++) {
    cout << "Hello World" << endl;
}
```

Output:

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Same like ***if*** statement, if there is only one line that needs to be repeated, then you do not need to put the brackets. However, if there are multiple lines that should repeat (or become the body of the loop), then you must write those in the brackets. Otherwise, only one (first) line will be considered the part of loop, the remaining will not be repeated because they would not be part of the loop.

Example 2: Printing Numbers From 1 to 5

```
#include <iostream>
using namespace std;
int main() {
  for (int i = 1; i <= 5; i++) {
    cout << i << " ";
  }
  return 0;
}
```

Output:

```
1 2 3 4 5
```

Here is how this program works

Iteration	Variable	Condition (i <= 5)	Action
1st	`i = 1`	`true`	`1` is printed. `i` is increased to `2`.
2nd	`i = 2`	`true`	`2` is printed. `i` is increased to `3`.
3rd	`i = 3`	`true`	`3` is printed. `i` is increased to `4`.
4th	`i = 4`	`true`	`4` is printed. `i` is increased to `5`.
5th	`i = 5`	`true`	`5` is printed. `i` is increased to `6`.
6th	`i = 6`	`false`	The loop is terminated

Example 3: Ask the user to input any range and then find the sum of all numbers from 1 to that range (sum of all natural numbers up to that range value)

```
// C++ program to find the sum of first n natural numbers
// positive integers such as 1,2, 3,...n are known as natural numbers
#include <iostream>
using namespace std;
int main() {
  int num, sum;
  sum = 0;
  cout << "Enter a positive integer: ";
  cin >> num;
  for (int count = 1; count <= num; ++count) {
    sum += count;
  }
  cout << "Sum = " << sum << endl;
  return 0;
}
```

Output:

```
Enter a positive integer: 10
Sum = 55
```

In the above example, we have two variables **num** and **sum**. The **sum** variable is assigned with 0 and the **num** variable is assigned with the value provided by the user.

Note that we have used a for loop.

```
for(int count = 1; count <= num; ++count)
```

Here

- **int count = 1**: initializes the **count** variable
- **count <= num**: runs the loop as long as **count** is less than or equal to **num**
- **++count**: increase the **count** variable by 1 in each iteration

When **count** becomes 11, the condition is **false** and **sum** will be equal to 0 + 1 + 2 + ... + 10.

Infinite loop

If the **condition** in a **for** loop is always **true**, it runs forever (until memory is full). For example,

```
// infinite for loop
for(int i = 1; i > 0; i++) {
   // block of code
}
```

In the above program, the **condition** is always **true** that will then cause the code to run for infinite time.

Sentinel-controlled loops

A sentinel-controlled loop is also called an indefinite repetition loop because the number of iterations is not known before the loop starts executing. In a sentinel-controlled loop, a special value called ***sentinel*** value is used to change the loop control expression from true to false in order to determine whether to execute the loop body. Sentinel-controlled loop is useful when we do not know in advance how many times the loop will be executed. An example of a sentinel-controlled loop is the processing of data from a text file (or database) of unknown size. Another example could be: continuously getting an integer as input from the user and only stopping when the user has entered a zero (0).

Two common sentinel-controlled loops in C++ are:

- while loop
- do-while loop

while loop

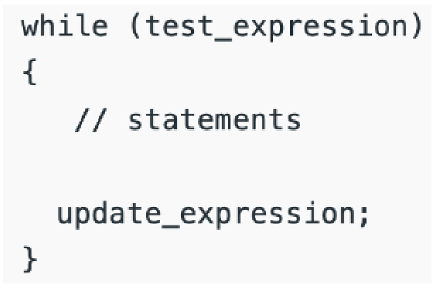
The general syntax of while loop is:

```
while (condition) {
    // body of the loop
}
```

Here

- A **while** loop evaluates the **condition**
- If the **condition** evaluates to **true**, the code inside the **while** loop is executed.
- The **condition** is evaluated again.
- This process continues until the **condition** becomes **false**.
- When the **condition** evaluates to **false**, the loop terminates.

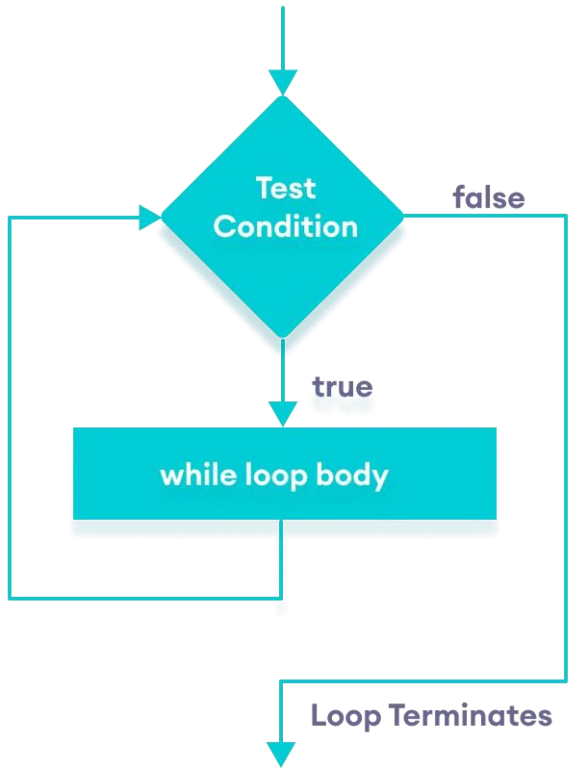
The **condition** must become **false** at some time in the future. Otherwise, it would become an infinite loop. Therefore, some people also write the general syntax as:



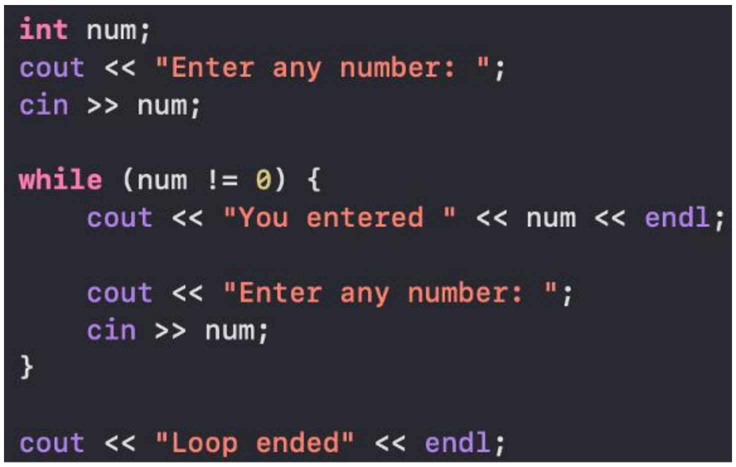
```
while (test_expression)
{
   // statements

  update_expression;
}
```

Where ***update_expression*** updates (usually increments/decrements) the loop variable being tested in the condition of the loop.

Flowchart of while loop

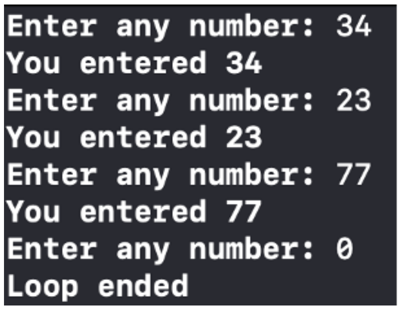


Example:



```
int num;
cout << "Enter any number: ";
cin >> num;

while (num != 0) {
    cout << "You entered " << num << endl;

    cout << "Enter any number: ";
    cin >> num;
}

cout << "Loop ended" << endl;
```

Output:



```
Enter any number: 34
You entered 34
Enter any number: 23
You entered 23
Enter any number: 77
You entered 77
Enter any number: 0
Loop ended
```

Example 2: Displaying numbers from 1 to 5

```
// C++ Program to print numbers from 1 to 5

#include <iostream>

using namespace std;

int main() {

  int i = 1;
  // while loop from 1 to 5

  while (i <= 5) {
    cout << i << " ";

    ++i;

  }

  return 0;

}
```

Output:

```
1 2 3 4 5
```

Here is how this program works

Iteration	Variable	Condition (i <= 5)	Action
1st	`i = 1`	`true`	`1` is printed. `i` is increased to `2`.
2nd	`i = 2`	`true`	`2` is printed. `i` is increased to `3`.
3rd	`i = 3`	`true`	`3` is printed. `i` is increased to `4`.
4th	`i = 4`	`true`	`4` is printed. `i` is increased to `5`.
5th	`i = 5`	`true`	`5` is printed. `i` is increased to `6`.
6th	`i = 6`	`false`	The loop is terminated

Example 3: Sum of positive numbers

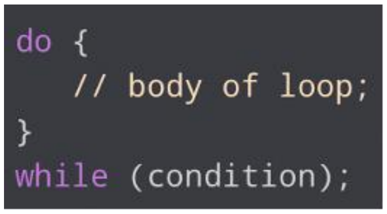
```
// program to find the sum of positive numbers

// if the user enters a negative number, the loop ends

// the negative number entered is not added to the sum

#include <iostream>
using namespace std;
int main() {

  int number;
  int sum = 0;
  // take input from the user

  cout << "Enter a number: ";

  cin >> number;

  while (number >= 0) {
    // add the current number with previous value of sum

    sum += number;

    // take input again to see if the number is positive

    cout << "Enter a number: ";

    cin >> number;

  }

  // display the sum

  cout << "\nThe sum is " << sum << endl;

  return 0;

}
```

Output:

```
Enter a number: 6

Enter a number: 12

Enter a number: 7

Enter a number: 0

Enter a number: -2

The sum is 25
```

do-while loop

The do...while loop is a variant of the while loop with one major difference: the body of do...while loop is executed once before the condition is checked. Its syntax is:

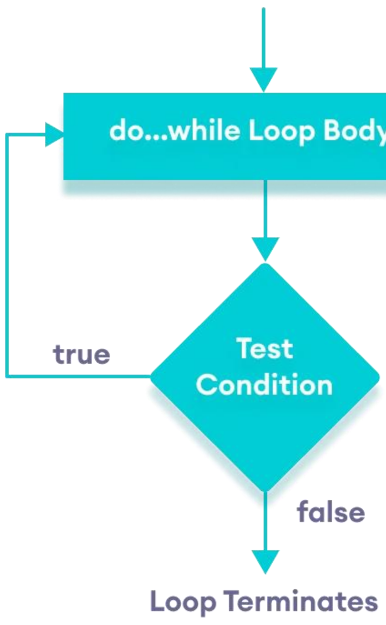


```
do {
   // body of loop;
}
while (condition);
```

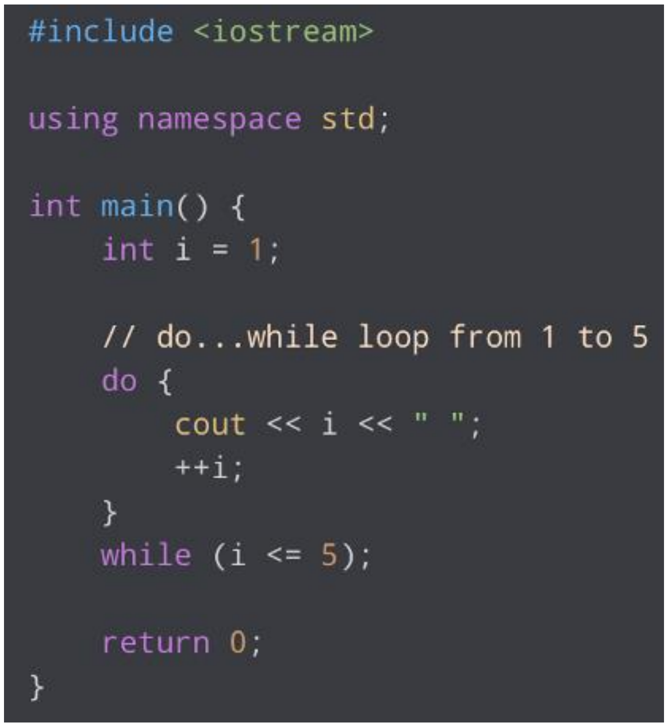
Here

- The body of the loop is executed at first. Then the condition is evaluated.
- If the condition evaluates to true, the body of the loop inside the do statement is executed again.
- The condition is evaluated once again. If the condition evaluates to true, the body of the loop inside the do statement is executed again.
- This process continues until the condition evaluates to false. Then the loop stops.

Flowchart of do-while loop



Example:



```
#include <iostream>

using namespace std;

int main() {
    int i = 1;

    // do...while loop from 1 to 5
    do {
        cout << i << " ";
        ++i;
    }
    while (i <= 5);

    return 0;
}
```

Output:

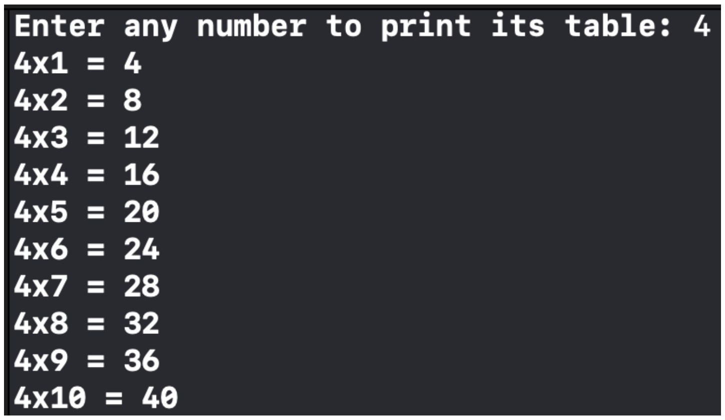
1 2 3 4 5

Here is how this program works

Iteration	Variable	Condition (i <= 5)	Action
	`i = 1`	`not checked`	`1` is printed. `i` is increased to `2`.
1st	`i = 2`	`true`	`2` is printed. `i` is increased to `3`.
2nd	`i = 3`	`true`	`3` is printed. `i` is increased to `4`.
3rd	`i = 4`	`true`	`4` is printed. `i` is increased to `5`.
4th	`i = 5`	`true`	`5` is printed. `i` is increased to `6`.
5th	`i = 6`	`false`	The loop is terminated

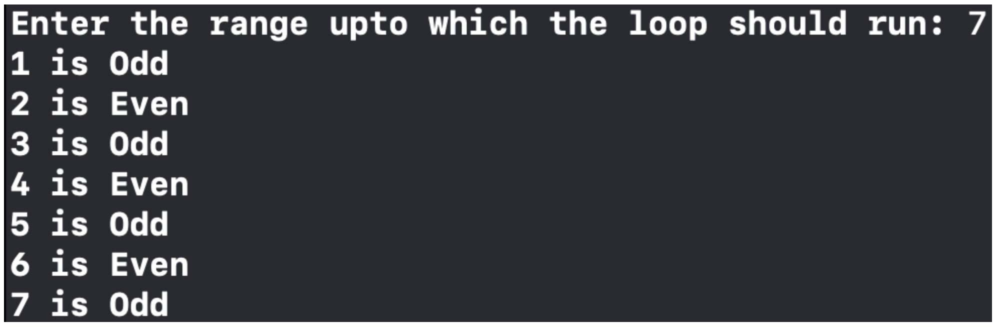
<u>Exercises</u>

1. Write a C++ program that should ask a user to input an integer number and then displays/prints its table as shown below:



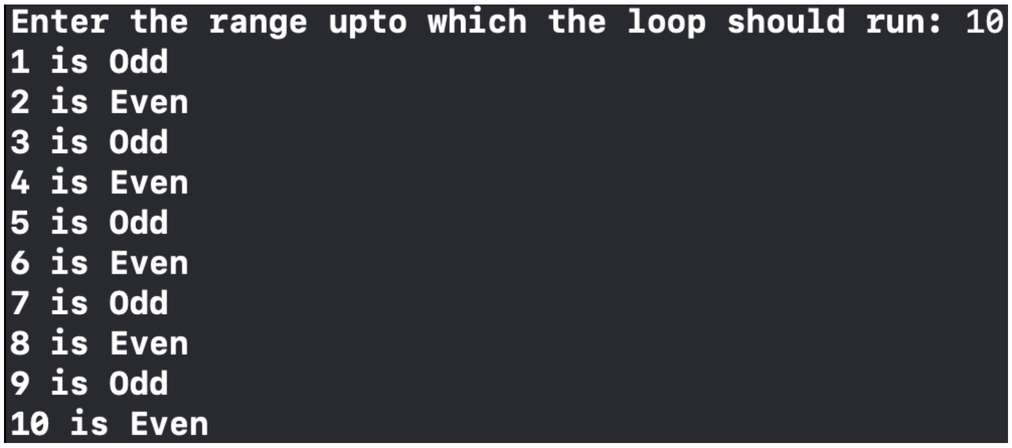
```
Enter any number to print its table: 4
4x1 = 4
4x2 = 8
4x3 = 12
4x4 = 16
4x5 = 20
4x6 = 24
4x7 = 28
4x8 = 32
4x9 = 36
4x10 = 40
```

2. Write a C++ program that asks a user to input the range (final value) up to which the loop should run and then for each number in that range, it should tell whether that (each) number is Odd or Even. For example, if the user enters 7 as the final value of range, then it should display like follows:



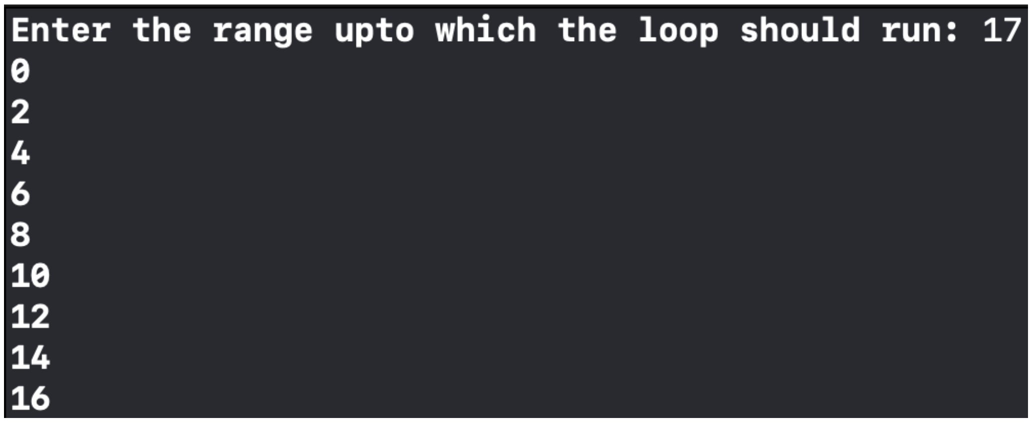
```
Enter the range upto which the loop should run: 7
1 is Odd
2 is Even
3 is Odd
4 is Even
5 is Odd
6 is Even
7 is Odd
```

And in other case, if the user inputs 10 as the final value of range, then



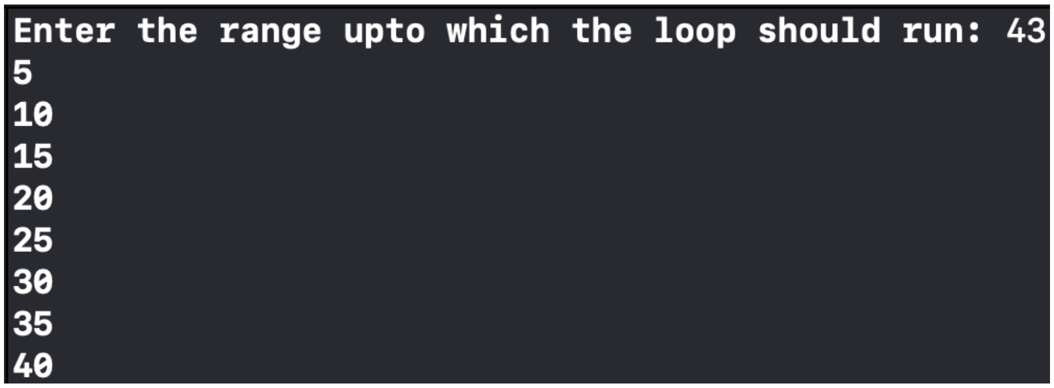
```
Enter the range upto which the loop should run: 10
1 is Odd
2 is Even
3 is Odd
4 is Even
5 is Odd
6 is Even
7 is Odd
8 is Even
9 is Odd
10 is Even
```

3. Write a C++ program that should ask the user to input the final range up to which the loop should run and then prints all the even numbers from zero (0) to that final value of range, like



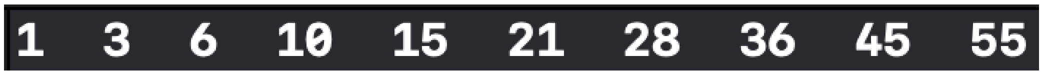
```
Enter the range upto which the loop should run: 17
0
2
4
6
8
10
12
14
16
```

4. Write a C++ program that should ask the user to input the final range up to which the loop should run and then prints ONLY the multiples of 5 (fully divided by 5 only), as



```
Enter the range upto which the loop should run: 43
5
10
15
20
25
30
35
40
```

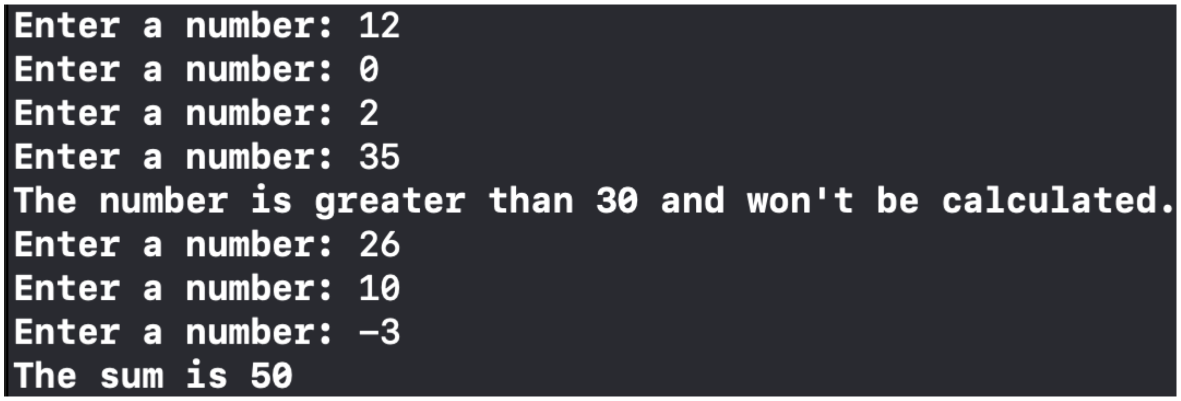
5. Create two separate programs (one using a for loop and the other using a while loop) to print a series of 10 numbers as shown below



```
1  3  6  10  15  21  28  36  45  55
```

6. Write a C++ program that should continuously ask the user to input any integer number until and unless a negative number is entered. When a negative number is entered, then the program should end and print the sum of all the entered numbers.

   However, it should also check that large numbers (greater than 30) should not be considered for calculating the sum, like shown below:



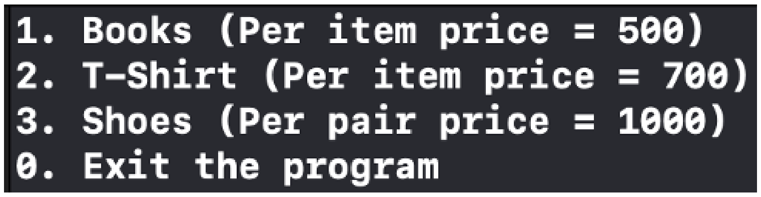
```
Enter a number: 12
Enter a number: 0
Enter a number: 2
Enter a number: 35
The number is greater than 30 and won't be calculated.
Enter a number: 26
Enter a number: 10
Enter a number: -3
The sum is 50
```

7. The Fibonacci sequence is a series where the next term is the sum of the previous two terms. The first two terms of the Fibonacci sequence are 0 and 1. The series is as follows:

   0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

   Write a C++ program (using loop) to print the 10 numbers of the Fibonacci series as shown above

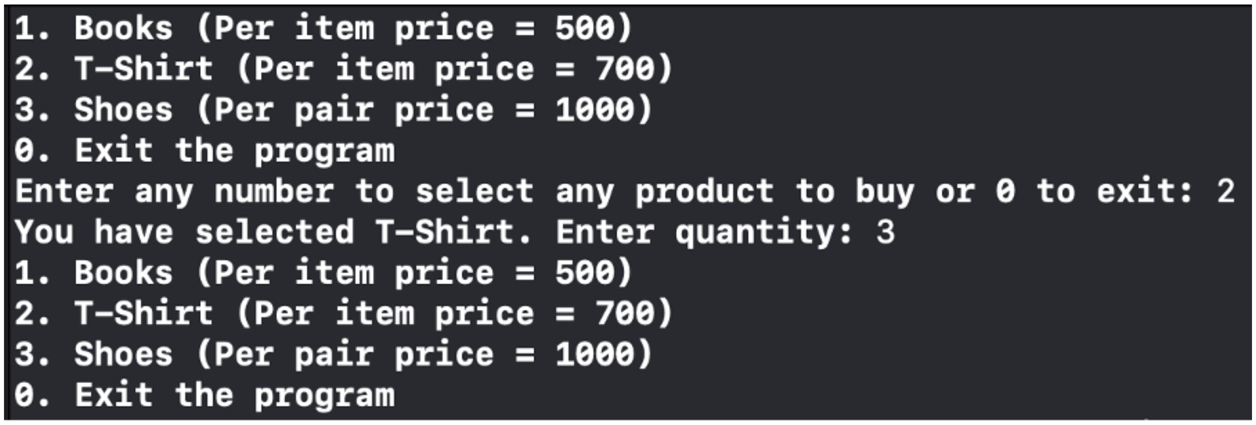
8. Create a program for a simple shopping application, where a user will be shown a Menu from which he/she can select any product to purchase or enter zero (0) to exit the program as shown below:



```
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
```

   As shown above, each product will also show its cost per item in the brackets.

   That Menu should be continuously shown to the user (so that the user can select multiple products (***OR even the same product again***) until and unless the user enters a zero (0), like



```
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
Enter any number to select any product to buy or 0 to exit: 2
You have selected T-Shirt. Enter quantity: 3
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
```

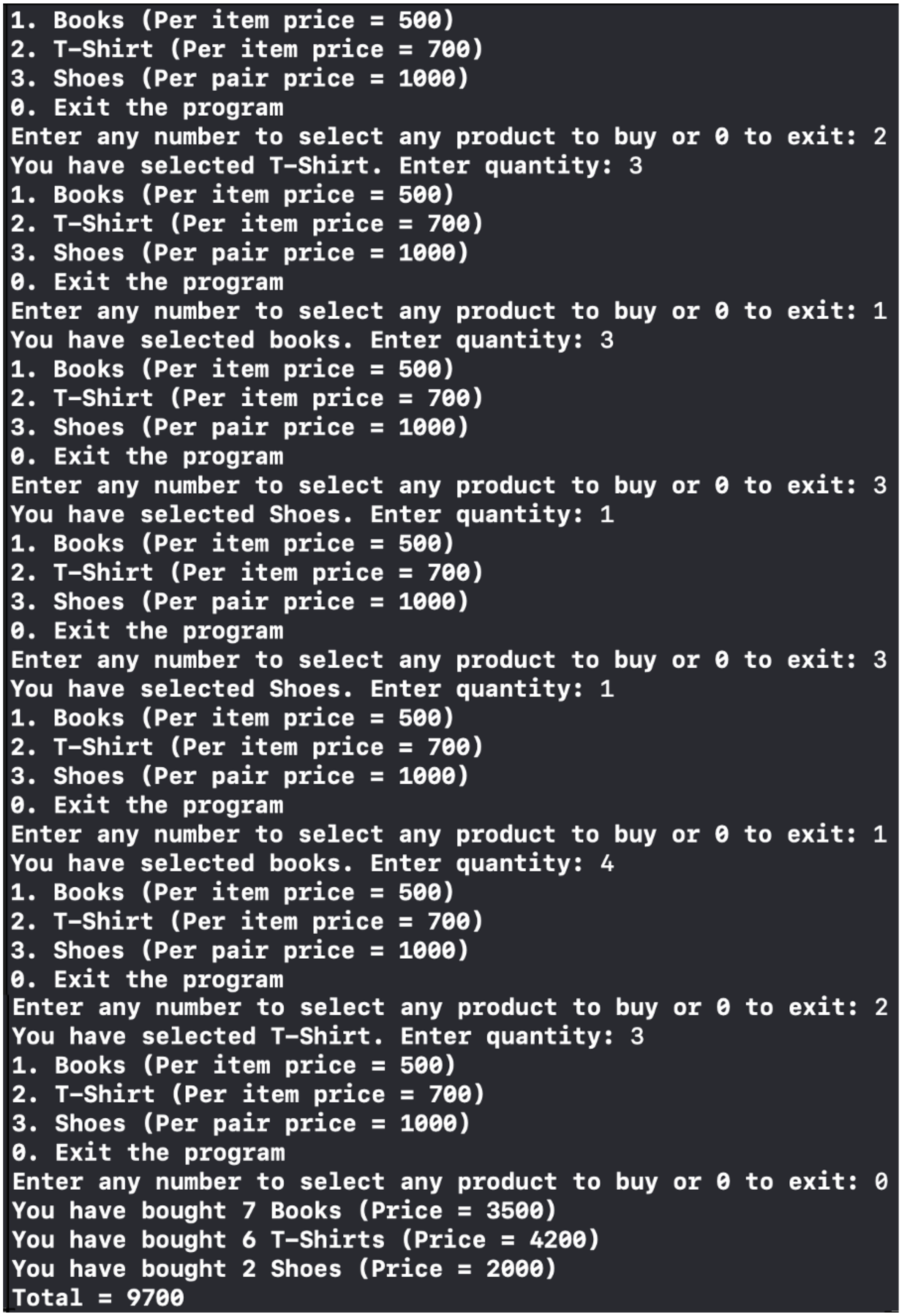
   When the user selects any product, then the program should ask the user to input the desired quantity of that product to be bought, as shown in the image above.

As discussed earlier, this process should continue and the user should be shown the Menu again and again so that he/she may select multiple products.

Finally, when the user enters a zero (0) to exit the program, then a detailed description should be shown that should include

- all the purchase products
- their quantity (for each product)
- the subtotal cost (of each product bought)
- and the total cost (of all the products)

A sample working is shown in the following image



```
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
Enter any number to select any product to buy or 0 to exit: 2
You have selected T-Shirt. Enter quantity: 3
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
Enter any number to select any product to buy or 0 to exit: 1
You have selected books. Enter quantity: 3
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
Enter any number to select any product to buy or 0 to exit: 3
You have selected Shoes. Enter quantity: 1
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
Enter any number to select any product to buy or 0 to exit: 3
You have selected Shoes. Enter quantity: 1
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
Enter any number to select any product to buy or 0 to exit: 1
You have selected books. Enter quantity: 4
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
Enter any number to select any product to buy or 0 to exit: 2
You have selected T-Shirt. Enter quantity: 3
1. Books (Per item price = 500)
2. T-Shirt (Per item price = 700)
3. Shoes (Per pair price = 1000)
0. Exit the program
Enter any number to select any product to buy or 0 to exit: 0
You have bought 7 Books (Price = 3500)
You have bought 6 T-Shirts (Price = 4200)
You have bought 2 Shoes (Price = 2000)
Total = 9700
```

9. Using loops, write a C++ program that displays a series of alphabets in descending order from ‘Z’ to ‘A.’
10. Modify your program in previous part so that the program will display consonants only, no vowels.
11. Write a C++ program that receives the total number of integers (N) from the user. Then, the program will ask the user for N real numbers. By applying for loop, your program should find and display the largest and the lowest of the numbers. Give a proper message for invalid user input, as shown below:

#Sample Program Run#1
```
How many numbers do you have? > -3
Sorry, you have entered an invalid input.
Thank You.
```

#Sample Program Run#2
```
How many numbers do you have? > 0
Opps, you don't have any number for me to process.
Thank You.
```

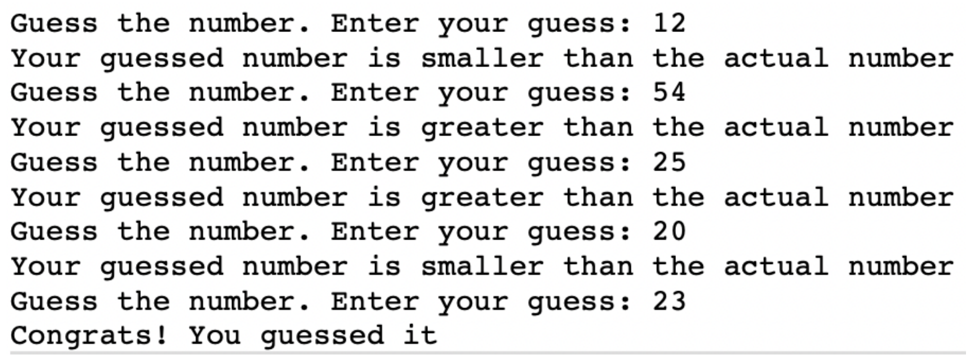
#Sample Program Run#3
```
How many numbers do you have? > 5
Please enter a number_1 --> 12.5
Please enter a number_2 --> -3.2
Please enter a number_3 --> 0
Please enter a number_4 --> 5
Please enter a number_5 --> 8.5

The lowest number is -3.20
The highest number is 12.50
Thank You.
```

12. Create a Guessing Game program, where you will guess a number. For now, assume any fixed number of choice as the actual number to be guessed (means, hardcode that value inside the code). Then, continuously ask the user to guess the number (I.e., input the number) unit and unless he/she guesses it correctly.

    Display the message(s) to the user as described below:

- If the entered number is smaller than the actual number, then you should display a message saying the user that his/her entered number is smaller than the actual number
- If the entered number is greater than the actual number, then you should display a message saying the user that his/her entered number is greater than the actual number

- If the user guesses the number correctly (I.e., enters the actual number), then display a Congrats message, as shown below:



```
Guess the number. Enter your guess: 12
Your guessed number is smaller than the actual number
Guess the number. Enter your guess: 54
Your guessed number is greater than the actual number
Guess the number. Enter your guess: 25
Your guessed number is greater than the actual number
Guess the number. Enter your guess: 20
Your guessed number is smaller than the actual number
Guess the number. Enter your guess: 23
Congrats! You guessed it
```

13. Extend the Guessing Game program created above. Other rules are the same as stated in the previous part. However, in this program, the actual number's value should be fixed between 1 and 100 and the user should be allowed only five (05) attempts.

    If the user guesses the number within five attempts, then display a Congrats message. Otherwise, the program should end after five wrong attempts, and display a sorry message to the user and the actual number as well.