Lab Manual

**CS111L – Programming Fundamentals Lab**

# Lab No: **07**

Topic: Iterative Control Structures

(For loop, While loop, Do-While loop)

Class: **BSSE** Semester:  **1** Session: Fall 2024

# Instructor: Mr. Zain Abubaker

# Lab Instructor: Mr. Zain Abubaker

A blue logo with a bird and a cross

Description automatically generated**Air University Islamabad**

**FACULTY OF COMPUTING & ARTIFICAL INTELLIGENCE**

**Department of Creative Technologies**

# Instructions

**Submission:** Use proper naming convention for your submission file. Name the submission file as Lab\_NO\_DEGREE\_ROLLNUM (e.g. Lab\_01\_BSSE\_00000). Submit the file on Google Classroom within the deadline. Failure to submit according to the above format would result in deduction of 10% marks. Submissions on the email will not be accepted.

**Plagiarism:** Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in the assignment, all of the remaining assignments, or even an F grade in the course. Copying from the internet is the easiest way to get caught!

**Deadline:** The deadlines to submit the assignment are hard. Late submission with marks deduction will be accepted according to the course policy shared by the instructor. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

**Comments:** Comment your code properly. Bonus marks (maximum 10%) will be awarded to well comment code. Write your name and roll number (as a block comment) at the beginning of the solution to each problem.

Objectives

In this lab, you will learn:

o About Structure ADT.

o How to define a Structure, initialize and refer to individual members of a Structure.

**Tip:** For timely completion of the assignment, start as early as possible. Furthermore, work smartly - as some of the problems can be solved using smarter logic.

**1.** Note: Follow the given instructions to the letter, failing to do so will result in a zero.

# Objectives

In this lab, you will learn:

* The use of repetition or loop.
* The basic control flow of first iterative control structure i.e. for loop.
* The basic difference between **break** and **continue** statements.
* How the jump statements alters the flow of the program.
* Nested Loops
* Patterns using nested loops

Concepts

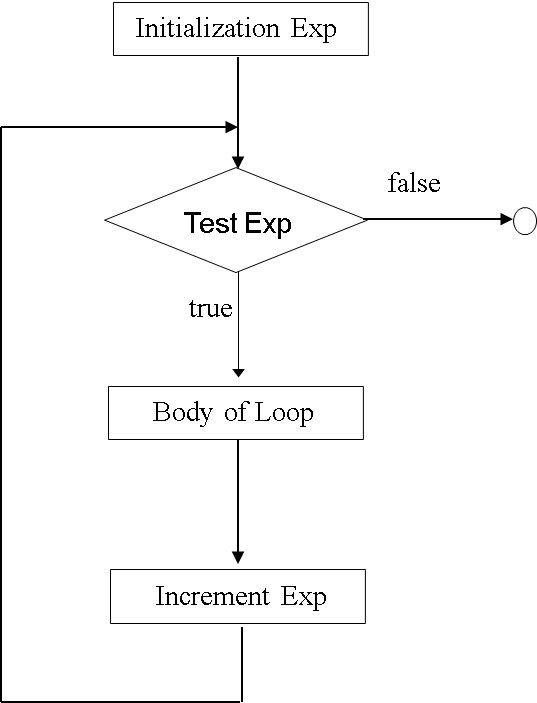
**Iteration statements (loops)**

Loops repeat a statement a certain number of times, or while a condition is fulfilled. They are introduced by the keywords while, do-while, and for.

**The for loop**

The **for** loop is designed to iterate a number of times. Its syntax and flow chart depiction is as follows:

**for (initialization; condition; increment) statement;**



**Fig 1: Flow Chart Depiction of the for loop**

As shown in Fig 1, this loop repeats statement while condition is true but, in addition to that, the **for** loop provides specific locations to contain an initialization and an increase expression, executed before the loop begins the first time, and after each iteration, respectively. Therefore, it is especially useful to use upon counter variables as condition.

It works in the following way:

1. **Initialization** is executed. Generally, this declares a counter variable, and sets it to some initial value. This is executed a single time, at the beginning of the loop.
2. **Condition** is checked. If it is true, the loop continues; otherwise, the loop ends, and statement is skipped, going directly to step 5.
3. **Statement** is executed. As usual, it can be either a single statement or a block enclosed in curly braces { }.
4. **increment** is executed, and the loop gets back to step 2.
5. **The loop ends**: execution continues by the next statement after it.

Here is the countdown example using a **for** loop:

|  |  |
| --- | --- |
| *// countdown using a for loop* | 10, 9, 8,7,6,5, 4, 3, 2, 1, liftoff! |
| *#include <iostream>* |  |
| *using namespace* std; |  |
| *int* main () |  |
| { |  |
| *for* ( *int* n=10; n>0; n--)  { |  |
| cout << n << ", "; |  |
| } |  |
| cout << "liftoff!\n"; |  |
| } |  |

The three fields in a for-loop are optional. They can be left empty, but in all cases the semicolon signs between them are required.

For example,

**for (; n<10 ;)** is a loop without *initialization* or *increase* (equivalent to a while-loop);

**for (; n<10; ++n)** is a loop with *increase*, but no *initialization* (maybe because the variable was already initialized before the loop).

A loop with no *condition* is equivalent to a loop with true as condition (i.e., an infinite loop).

Because each of the fields is executed in a particular time in the life cycle of a loop, it may be useful to execute more than a single expression as any of *initialization*, *condition*, or *statement*. Unfortunately, these are not statements, but rather, simple expressions, and thus cannot be replaced by a block. As expressions, they can, however, make use of the comma operator (,): This operator is an expression separator, and can separate multiple expressions where only one is generally expected. For example, using it, it would be possible for a **for** loop to handle two counter variables, initializing and increasing both:

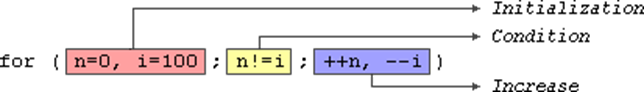
*for* ( n=0, i=100 ; n!=i ; ++n, --i )

{

*// whatever here...*

}

As shown in Fig 2, this loop will execute **50** times if neither **n** nor **i** are modified within the loop: n starts with a value of 0, and i with 100, the condition is n!=i (i.e., that n is not equal to i). Because n is increased by one, and i decreased by one on each iteration, the loop's condition will become false after the 50th iteration, when both n and i are equal to 50.



**Fig 2: Scenario of the for loop**

**Range-based for loop**

The for-loop has another syntax, which is used exclusively with ranges:

for (declaration:range)

{ statement; }

|  |
| --- |
| #include <iostream>  int main() {  for (int n : {0,2,4, 8,10})  {  std::cout << n << ' ';  }  return 0;  } |

This kind of for loop iterates over all the elements in range, where declaration declares some variable able to take the value of an element in this range. Ranges are sequences of elements, including arrays, containers, and any other type supporting the functions begin and end; Most of these types have not yet been introduced in this course, but we are already acquainted with at least one kind of range: strings, which are sequences of characters.

**The while loop**

The simplest kind of loop is the while-loop. Its syntax is: while (expression) statement

The while-loop simply repeats statement while expression is true. If, after any execution of statement, expression is no longer true, the loop ends, and the program continues right after the loop.

|  |
| --- |
| // 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;  } |

**The do-while loop**

The **do-while** loop is a variant of the while loop with one important difference:

the body of do...while loop is executed once before the condition is checked

|  |
| --- |
| // C++ Program to print numbers from 1 to 5  #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;  } |

do-while loops are useful when you want to ensure that a specific code block is executed at least once, regardless of whether the condition is initially true or false. They are commonly used when you need to gather input from a user and validate it. For example, you might use a do-while loop to repeatedly ask the user for a value until they enter a valid one.

**Jump Statements** that allow altering the flow of a program by performing jumps to specific locations in the code. There are two jump statements i.e. **break** and **continue.**

**The break Statement:**

The **break** leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end. For example, let's stop the countdown before its natural end

|  |  |
| --- | --- |
| // break loop example | 10, 9, 8, 7, 6, 5, 4, 3, countdown |
| #include <iostream> | aborted! |
| using namespace std; |  |
| int main () |  |
| { |  |
| for (int n=10; n>0; n--) |  |
| { |  |
| cout << n << ", "; |  |
| if (n==3) |  |
| { |  |
| cout << "countdown aborted!"; |  |
| break; |  |
| } |  |
| } |  |
| } |  |
|  |  |
|  |  |
|  |  |

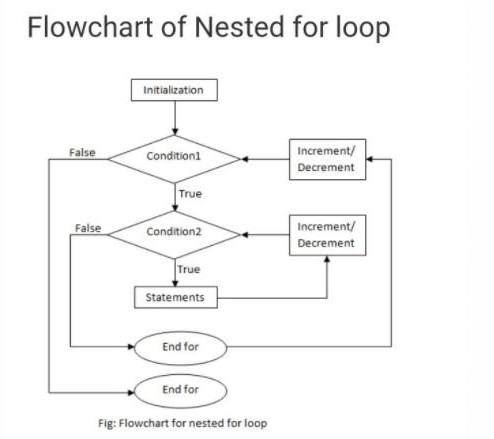
**The continue statement:**

The continue statement causes the program to skip the rest of the loop in the current iteration, as if the end of the statement block had been reached, causing it to jump to the start of the following iteration. For example, let's skip number 5 in our countdown:

|  |  |
| --- | --- |
| // continue loop example | 10, 9, 8, 7, 6, 4, 3, 2, 1, liftoff! |
| #include <iostream> |  |
| using namespace std; |  |
| int main () |  |
| { |  |
| for (int n=10; n>0; n--)  { |  |
| if (n==5) continue; |  |
| cout << n << ", "; |  |
| } |  |
| cout << "liftoff!\n"; |  |
| } |  |

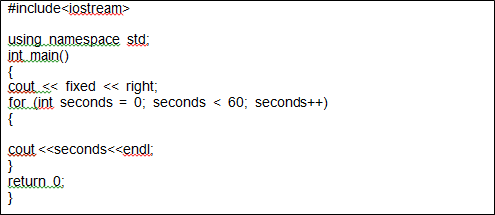
**Nested Loop:**

A nested loop is a loop that appears inside another loop.

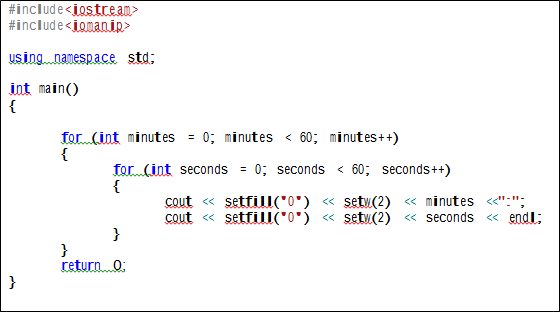


A clock is a good example of something that works like a nested loop. The second hand, minute hand, and hour hand all spin around the face of the clock. Each time the hour hand increments, the minute hand increments 60 times. Each time the minute hand increments, the second hand increments 60 times.

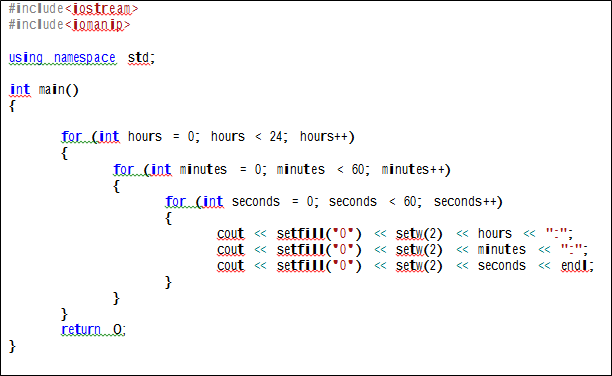
Here is a program segment with a for loop that partially simulates a digital clock. It displays the seconds from 0 to 59:



We can add a minutes variable and nest the loop above inside another loop that cycles through 60 minutes:



To make the simulated clock complete, another variable and loop can be added to count the hours:



The output of the previous program segment follows:

00:00:00

00:00:01

00:00:02

. (The program will count through each second of 24 hours.)

.

. 23:59:59

The innermost loop (seconds hand) will iterate 60 times for each iteration of the middle loop (minute hand). The middle loop (minute hand) will iterate 60 times for each iteration of the outermost loop (hour hand). When the outermost loop has iterated 24 times, the middle loop will have iterated 1,440 times and the innermost loop will have iterated 86,400 times!

The simulated clock example brings up a few points about nested loops:

* An inner loop goes through all of its iterations for each iteration of an outer loop.
* Inner loops complete their iterations faster than outer loops.
* To get the total number of iterations of a nested loop, multiply the number of iterations of all the loops

Often programmers need to use a loop within a loop, or nested loops. Sample Program given below provides a simple example of a nested loop. This program finds the average number of hours per day spent programming by each student over a three-day weekend. The outer loop controls the number of students and the inner loop allows the user to enter the number of hours worked each of the three days for a given student.

Note that the inner loop is executed three times for each iteration of the outer loop.

// This program finds the average time spent programming by a student each

// day over a three day period.

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

{

int numStudents;

float numHours, total, average;

int count1 = 0, count2 = 0; // these are the counters for the loops

cout << "This program will find the average number of hours a day that each given student spent programming over a long weekend" << endl << endl;

cout << "How many students are there ?" << endl << endl; cin >> numStudents;

for (count1 = 1; count1 <= numStudents; count1++)

{

total = 0;

for (count2 = 1; count2 <= 3; count2++)

{

cout << "Please enter the number of hours worked by student " << count1 << "on day " << count2 << "." << endl;

cin >> numHours;

total = total + numHours;

}

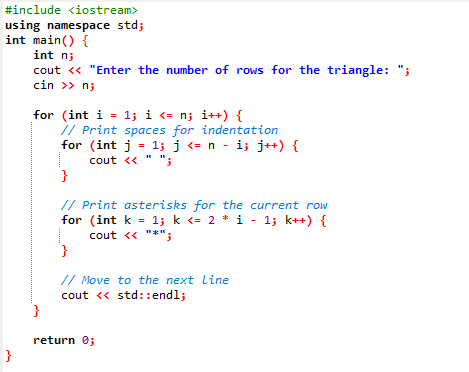
average = total / 3;

cout << "\nThe average number of hours per day spent programming by student " << count1 <<" is " << average<< endl << endl << endl;

}

return 0;

}



## Lab Tasks:

**Exercise 1:** Write a program to print first ‘n’ natural numbers using for-loop and while loop.

Input:

Input upper limit: 10 Output: 1 2 3 4 5 6 7 8 9 10

**Exercise 2:** Write a C++ program that will print the following pattern using nested loop.

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*\*

**Exercise 3:** Write a program to display 7 days of three week using for loop.

Week 1

Day1 Day2 Day3 Day4 Day5 Day6 Day7

Week 2

Day1 Day2

Day3 Day4 Day5 Day6 Day7

Week 3

Day1 Day2 Day3 Day4 Day5 Day6 Day7

**Exercise 4:** Write a program to display a multiplication table for 5\*5. Make it generic.

## Tasks:

### Exercise 1:

### Write a C++ program to print numbers from 10 to 20

### Write a C++ program to print a sequence of number from a given range. Ask user to input the starting and ending number.

### Write a C++ program to print sum of all even numbers from 50 to 250 using for loop

### Develop a C++ program to display the first N terms of the Fibonacci sequence using a for

### loop, where N is user-input.

### Exercise 2:

1. Write a C++ program to print a number, its additive inverse and multiplicative inverse in a tabular form.
2. Write a C++ program to print sum of all even numbers from 50 to 250 using for loop
3. while loop
4. Write a C++ program using a while loop to calculate the factorial of a given positive integer (n).
5. Write a C++ program that uses break to exit a loop prematurely when a certain condition is met.

### Exercise 3:

### Write a C++ program to print even numbers from 0 to 50 using do while loop

1. Write a C++ program using a do-while loop to calculate and display the sum of the first N odd numbers (e.g., 1, 3, 5, ...) where N is provided by the user.