**Programming Fundamentals**

Name: \_faiz ur Rehman siddiqui \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Roll Number: \_\_\_\_\_\_\_\_22164\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Course Code: \_\_\_cmc111l-s25bc(cs)-136-iu04spring-2025-A1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Computer Science Department

List of Experiments

|  |  |  |  |
| --- | --- | --- | --- |
| **Week** | **Topic Covered** | **Remarks** | **Mapping** |
| 1 | To get familiar with Programming Environment and Fundamentals of Programming Language |  | CLO1,2 |
|
| 2 | To get familiar with Operators and Math library |  | CLO1,2 |
|
| 3 | To get familiar with the use of basic control structures in C++: if-else and else-if statements. |  | CLO1,2 |
|
| 4 | To get familiar with the use of **nested** if**-**else statements for more complex decision-making scenarios.  . |  | CLO1,2 |
|
| 5 | To understand the structure and procedure of switch cases in C++ |  | CLO1,2 |
|
| 6 | To understand the structure and procedure of Loops in C++ |  | CLO1,2 |
|
| 7 | To develop understanding of nested for loop phenomenon |  | CLO1,2 |
|
| 8 | The purpose of this lab is to get familiar with the concept of numbers and arrays |  | CLO1,2 |
| 9. | The purpose of this lab is to get a familiar with the concept and description multidimensional arrays. |  | CLO1,2 |
|
| 10 | To be familiarized with Introduction of Pre-define Function (PDF) and  User-define Function (UDF). |  | CLO1,2 |
|
| 11 | The purpose of this lab is to get understanding about structures in C++ |  | CLO1,2 |
|
| 12 | To familiarize with the usage of the Filling in C++. |  | CLO1,2 |
|
| 13 | PROBLEM BASED LEARNING |  | CLO1,2 |
|
| 14 | OPEN ENDED LAB |  | CLO1,2 |
|
| **Final Term Examination Week** | | | |

***Rubric for Assessment (Software)***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Criteria*** | ***Exceeds Expectations (>=88%)*** | ***Meets Expectations (74%-87%)*** | ***Developing (60%-73%)*** | ***Unsatisfactory (<60%)*** |
| ***Software Skills*** | ***Ability to use software with its standard and advanced features without assistance*** | ***Ability to use software with its standard and advanced features with minimal assistance*** | ***Ability to use software with its standard features with assistance*** | ***Unable to use the software*** |
| ***Programming/ Simulation*** | ***Ability to program/ simulate the lab tasks***  ***with simplification*** | ***Ability to program/ simulate the lab tasks***  ***without errors*** | ***Ability to program/ simulate lab tasks with errors*** | ***Unable to program/simulate*** |
| ***Results*** | ***Ability to achieve all the desired results with alternate ways*** | ***Ability to achieve all the desired results*** | ***Ability to achieve most of the desired results with errors*** | ***Unable to achieve the desired results*** |
| ***Laboratory Manual*** | ***All sections of the report are very well written and technically accurate.*** | ***All sections of the report are technically accurate.*** | ***Few sections of the report contain technical errors.*** | ***All sections of the report contain multiple technical errors.*** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Score Allocation** | | | | | |
| **S.No** | **Experimental Setup:** | **Procedure:** | **Experimental Results:** | **Laboratory Manual:** | **Score:** |
| 1 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 2 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 3 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 4 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 5 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 6 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 7 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 8 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 9 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 10 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 11 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| 12 | (\_\_\_) /3 | (\_\_\_) /2 | (\_\_\_) /3: | (\_\_\_) /2 | **(\_\_\_) /10** |
| **Total Obtained Score** | | | | | **(\_\_\_)120** |

***Formula= (Total Obtained Score / 120)* x 30 *Examined by****: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* ***Score: \_\_\_\_\_\_\_\_\_\_\_ out of 30             (Signature of Instructor)***

### ***Instructor:* Sumaira Ahmed**

***Prerequisites: Information and Communication Technology (lAB)***

***Objectives:***

This lab aims to cover overview of Computer Programming, Principles of Structured and Modular Programming, Overview of Structured Programming Languages, Algorithms and Problem Solving, Program Development: Analyzing Problem, Designing Algorithm/Solution, Testing Designed Solution, Translating Algorithms into Programs, Fundamental Programming Constructs.

***Contents:***

Basics of Input and Output, Selection and Decision (If, If-Else, Nested If-Else, Switch Statement and Condition Operator), Repetition (While and For Loop, Do-While Loops), Break Statement, Continue Statement, Control Structures, Functions, Arrays, Pointers, Records, Files (Input-Output), Testing & Debugging.

***Learning Outcomes:***

|  |  |  |
| --- | --- | --- |
| **CLO/GA Mapping and Assessment** | |  |
| **CLOs** | **BT Level** | **Related GAs** |
| ***CLO1: Apply*** knowledge of C++ programming constructs and evaluate basic graphical user interfaces and animations using C++ graphics, incorporating fundamental principles of computer graphics programming. | C3 | GA4- (Design/Development of Solutions) |
| CLO2: ***Join***in lab activities such as projects and lab assignments, lab tasks as an individual and an effective team member. | A3 | GA6 - (Individual and Team Work) |

|  |  |  |  |
| --- | --- | --- | --- |
| **CLO Assessment Mechanism** | | | |
| **Assessment tool** | **CLO 1** | **CLO 2** | **Assessment %** |
| Lab Manual | 15% | 15% | 30% |
| OEL | 5% | 5% | 10% |
| PBL | 5% | 5% | 10% |
| Final Exam | 25% | 25% | 50% |
| Total CLO weightage | 50% | 50% | Total 100% |

|  |  |  |
| --- | --- | --- |
| **Grading Policy** | | |
| Marks | \*GPA | Grade |
| Greater than 88 | 4 | A |
| Between 81 to 87 | 3.5 | B+ |
| Between 74 to 80 | 3 | B |
| Between 67 to 73 | 2.5 | C+ |
| Between 60 to 66 | 2 | C |
| Below 60 | 0 | F |
| \*GPA = Grade Point Average | | |

|  |  |  |
| --- | --- | --- |
| **Course Textbook / Reference Books and Supplementary Reading Material** | | |
| **S No** | **Book Title** | **Author(s)** |
|  | C++ Programming: From Problem Analysis to Program Design (6th Edition) | D. S. Malik |
|  | The C++ Programming Language (4th Edition) | Bjarne Stroustrup |
|  | C++ How to Program (10th Edition) | Paul Deitel & Harvey Deitel |

***Administrative Instructions:***

* Title and Group members name for Lab/Course project should be submitted by 5thweek of lab.
* According to institute policy, 75% attendance is *mandatory* to appear in the final examination but 100% will be expected. Approved leaves will not be considered towards attendance.
* Every student should bring a calculator, book and manual in each lab.
* Every student is expected to be in the lab before the scheduled starting time.
* In any case there will be no rescheduling and makeup of labs.

***Total Lab Assessment***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Assessment Criteria** | **Criteria Marks** | | | **Obtained Marks** | | | **Total** |
| **CLOs** | **CLO1** | **CLO2** | **CLO3** | **CLO1** | **CLO2** | **CLO3** |
| Lab Manual+ Lab Performance | 15 | 15 | - |  |  | - | 30 |
| Problem Based Learning | 5 | 5 | - |  |  | - | 10 |
| Open Ended Lab | 5 | 5 | - |  |  | - | 10 |
| Final Exam | 25 | 25 | - |  |  | - | 50 |
| Total | 50 | 50 | - |  |  | - | 100 |

**General Laboratory Procedure**

While there is no specific document to be submitted at the beginning of the Lab –unless your instructor advises you otherwise-, you are expected to read the experiment fully before you come to the laboratory? Interestingly, you can even try parts of the experiment at home. Here is a list of programs that will equip you with a virtual lab at your home:

**Troubleshooting**

Things will not always go as expected; this is the nature of the learning process. While conducting the Experiment **think before you do anything.** If you do so you will avoid wasting time going down dead-end streets. Be logical and systematic. First, look for obvious errors that are easy to fix. Is your measuring device correctly set and connected? Are you looking at the proper scale? Is the power supply set for the correct voltage? Is the signal generator correctly set and connected? How are the variables in the code set? Is there a syntax error? And so on. Next, check for obvious misconnections or broken connections, at least in simple circuits.

As you work through your circuit, use your Lab Manual record tests and changes that you make as you go along; don't rely on your memory for what you have tried. Identify some test points in the system at which you know what the signal should be and work your way backwards from the output through the test points until you find a good signal.

**Neatness**

When you have finished for the day, return all modules to their proper storage bins, return all test leads and probes to their storage racks, return all equipment to its correct location, and clean up the lab station. If appropriate switch off the unneeded equipment. Save your files in the Computer and on any USB device for your records because you might not get the same PC System again for the next experiment. Also email your file contents to your email address as a backup.

**Laboratory Safety**



Always pay attention to what you are doing and you’re surrounding during the experiments, notify the Instructor for any unlikely event or mishap, and leave the Laboratory with the permission of Instructor immediately.

All students must read and understand the information in this document with regard to laboratory safety and emergency procedures prior to the first laboratory session.

**Your personal laboratory safety depends mostly on YOU**. Efforts have been made to address situations that may pose a hazard in the lab but the information and instructions provided cannot be considered all-inclusive.

Students must adhere to written and verbal safety instructions throughout the academic term. Since additional instructions may be given at the beginning of laboratory sessions, it is important that all students arrive at each session on time.  With good judgment, the chance of an accident in this course is very small. Nevertheless, research and teaching workplaces (labs, shops, etc.) are full of potential hazards that can cause serious injury and or damage to the equipment. Working alone and unsupervised in laboratories is forbidden if you are working with hazardous substances or equipment. With prior approval, at least two people should be present so that one can shut down equipment and call for help in the event of an emergency. Safety training and/or information should be provided by a faculty member, teaching assistant, lab safety contact, or staff member at the beginning of a new assignment or when a new hazard is introduced into the workplace.

**Emergency Response**

1. It is your responsibility to read safety and fire alarm posters and follow the instructions during an emergency
2. Know the location of the fire extinguisher, eye wash, and safety shower in your lab and know how to use them.
3. Notify your instructor immediately after any injury, fire or explosion, or spill.
4. Know the building evacuation procedures.

**Common Sense**

Good common sense is needed for safety in a laboratory. It is expected that each student will work in a responsible manner and exercise good judgment and common sense. If at any time you are not sure how to handle a particular situation, ask your Teaching Assistant or Instructor for advice **DO NOT TOUCH ANYTHING WITH WHICH YOU ARE NOT COMPLETELY FAMILIAR**!!! It is always better to ask questions than to risk harm to yourself or damage to the equipment. 

**Personal and General laboratory safety**

1. Never eat, drink, or smoke while working in the laboratory.
2. Read labels carefully.
3. Do not use any equipment unless you are trained and approved as a user by your supervisor.
4. Wear safety glasses or face shields when working with hazardous materials and/or equipment.
5. Wear gloves when using any hazardous or toxic agent.
6. Clothing: When handling dangerous substances, wear gloves, laboratory coats, and safety shield or glasses. Shorts and sandals should not be worn in the lab at any time. Shoes are required when working in the machine shops.
7. If you have long hair or loose clothes, make sure it is tied back or confined.
8. Keep the work area clear of all materials except those needed for your work. Coats should be hung in the hall or placed in a locker. Extra books, purses, etc. should be kept away from equipment that requires air flow or ventilation to prevent overheating.
9. Disposal - Students are responsible for the proper disposal of used material if any in appropriate containers.
10. Equipment Failure - If a piece of equipment fails while being used, report it immediately to your lab assistant or tutor. Never try to fix the problem yourself because you could harm yourself and others.
11. If leaving a lab unattended, turn off all ignition sources and lock the doors.
12. Never pipette anything by mouth.
13. Clean up your work area before leaving.
14. Wash hands before leaving the lab and before eating.
15. Unauthorized person(s) shall not be allowed in a laboratory for any reason

**Electrical safety**

1. Obtain permission before operating any high voltage equipment.
2. Maintain an unobstructed access to all electrical panels.
3. Wiring or other electrical modifications must be referred to the Electronics Shop or the Building Coordinator.
4. Avoid using extension cords whenever possible. If you must use one, obtain a heavy- duty one that is electrically grounded, with its own fuse, and install it safely. Extension cords should not go under doors, across aisles, be hung from the ceiling, or plugged into other extension cords.
5. Never, ever modify, attach or otherwise change any high voltage equipment.
6. Always make sure all capacitors are discharged (using a grounded cable with an insulating handle) before touching high voltage leads or the "inside" of any equipment even after it has been turned off. Capacitors can hold charge for many hours after the equipment has been turned off.
7. When you are adjusting any high voltage equipment or a laser which is powered with a high voltage supply, USE ONLY ONE HAND. Your other hand is best placed in a pocket or behind your back. This procedure eliminates the possibility of an accident where high voltage current flows up one arm, through your chest, and down the other arm.
8. Discard damaged cords, cords that become hot, or cords with exposed wiring.
9. Before equipment is energized ensure, (1) circuit connections and layout have been checked by a Teaching Assistant (TA) and (2) all colleagues in your group give their assent.
10. Know the correct handling, storage and disposal procedures for batteries, cells, capacitors, inductors and other high energy-storage devices.
11. Experiments left unattended should be isolated from the power supplies. If for a special reason, it must be left on, a barrier and a warning notice are required.
12. Equipment found to be faulty in any way should be reported to the Lab Engineer immediately and taken out of service until inspected and declared safe.
13. Voltages above 50 V rms AC and 120 V DC are always dangerous. Extra precautions should be considered as voltage levels are increased.
14. Never make any changes to circuits or mechanical layout without first isolating the circuit by switching off and removing connections to power supplies.
15. Know what you must do in an emergency.
16. Emergency Power Off: Every lab is equipped with and Emergency Power Off System.
17. Only authorized personnel are permitted to reset power once the Emergency Power Off system has been engaged.

**Electrical Emergency Response**

The following instructions provide guidelines for handling two types of electrical emergencies:

1. When someone suffers serious electrical shock, he or she may be knocked unconscious. If the victim is still in contact with the electrical current, immediately turn off the electrical power source. If you cannot disconnect the power source, depress the Emergency Power Off switch.
2. Do not touch a victim that is still in contact with a live power source; you could be electrocuted.
3. Have someone call for emergency medical assistance immediately. Administer first-aid, as appropriate.
4. If an electrical fire occurs, try to disconnect the electrical power source, if possible. If the fire is small and you are not in immediate danger; and you have been properly trained in fighting fires, use the correct type of fire extinguisher to extinguish the fire. When in doubt, push in the Emergency Power Off button.
5. NEVER use water to extinguish an electrical fire.

**Mechanical safety**

1. When using compressed air, use only approved nozzles and never direct the air towards any person.
2. Guards on machinery must be in place during operation.
3. Exercise care when working with or near hydraulically- or pneumatically-driven equipment. Sudden or unexpected motion can inflict serious injury. ****

**Additional Safety Guidelines**

1. Never do unauthorized experiments.
2. Never work alone in laboratory.
3. Keep your lab space clean and organized.
4. Do not leave an on-going experiment unattended.
5. Always inform your instructor if you break a thermometer. Do not clean mercury yourself!!
6. Never taste anything. Never pipette by mouth; use a bulb.
7. Never use open flames in laboratory unless instructed by TA.
8. Check your glassware for cracks and chips each time you use it. Cracks could cause the glassware to fail during use and cause serious injury to you or lab mates.
9. Maintain unobstructed access to all exits, fire extinguishers, electrical panels, emergency showers, and eye washes.
10. Do not use corridors for storage or work areas.
11. Do not store heavy items above table height. Any overhead storage of supplies on top of cabinets should be limited to lightweight items only. Also, remember that a 36" diameter area around all fire sprinkler heads must be kept clear at all times.
12. Areas containing lasers, biohazards, radioisotopes, and carcinogens should be posted accordingly. However, do not post areas unnecessarily and be sure that the labels are removed when the hazards are no longer present.
13. Be careful when lifting heavy objects. Only shop staff may operate forklifts or cranes.
14. Clean your lab bench and equipment, and lock the door before you leave the laboratory.

**Clothing**

1. Dress properly during a laboratory activity.
2. Long hair, dangling jewelry, and loose or baggy clothing are a hazard in the laboratory.
3. Long hair must be tied back, and dangling jewelry and baggy clothing must be secured.
4. Shoes must completely cover the foot.
5. No sandals allowed on lab days.
6. A lab coat or smock should be worn during laboratory experiments.

**Accidents and Injuries**

1. Do not panic.
2. Report any accident (spill, breakage, etc.) or injury (cut, burn, etc.) to the teacher immediately, no matter how trivial it seems.
3. If you or your lab partner is hurt, immediately (and loudly) yell out the teacher's name to get the teacher's attention.

**General Warning Signs**



**Lab - 01**

**To get familiar with Programming Environment and Fundamentals of Programming Language**

**Theory:**

**The Development Environment - Integrated Development Environment (IDE):**

The C compiler has its own built-in text editor. You may also use a commercial text editor or word processor that can produce text files. The important thing is that whatever you write your program in, it must save simple, plain-text files, with no word processing commands embedded in the text. The files you create with your editor are called source files, and for

C++ they typically are named with the extension *.CPP*, *.CP*, or *.C*.

The C Developing Environment, also called as Programmers’ Platform, is a screen display with windows and pull-down menus. Code of the program, error messages and other information are displayed in separate windows. The menus may be used to invoke the operations necessary to develop the program, debug and execute the program.

**Invoking the IDE**

To invoke the IDE from the windows you need to double click the Dev C++ icon.

This makes you enter the IDE interface, which initially displays only a menu bar at the top of the screen and a status line below will appear. The menu bar displays the menu names and the status line tells what various function keys will do.

**Opening New Window**

To type a program, you need to open an Edit Window. For this, open file menu and click “new” or press [CTRL +N]. A window will appear on the screen where the program may be typed.

**Writing a Program**

When the Edit window is active, the program may be typed. Use the certain key combinations to perform specific edit functions.

**Saving a Program**

To save the program, select **save** command from the **file** menu. This function can also be performed by pressing the [CTRL+S]. A dialog box will appear asking for the path and name of the file. Provide an appropriate and unique file name. You can save the program after compiling too but saving it before compilation is more appropriate.

**Making an Executable File**

The source file is required to be turned into an executable file. This is called “Making” of the .*exe* file. The steps required to create an executable file are:

**1.** Create a source file, with a .*cpp* extension.

**2.** Compile the source code into a file with the .*obj* extension.

**3.** Link your .*obj* file with any needed libraries to produce an executable program.

**Compiling the Source Code**

Although the source code in your file is somewhat cryptic, and anyone who doesn't know C will struggle to understand what it is for, it is still in what we call human-readable form. But, for the computer to understand this source code, it must be converted into machine-readable form. This is done by using a compiler. Hence, compiling is the process in which source code is translated into machine understandable language.

**Creating an Executable File with the Linker**

After your source code is compiled, an object file is produced. This file is often named with the extension .*OBJ*. This is still not an executable program, however. To turn this into an executable program, you must run your linker. C programs are typically created by linking together one or more OBJ files with one or more libraries. A library is a collection of linkable files that were supplied with your compiler.

**Project/Make**

Before compiling and linking a file, a part of the IDE called Project/Make checks the time and date on the file you are going to compile.

**Compiling and linking in the IDE**

In the Dev C++, compiling and linking can be performed together in one step. There are two ways to do this: you can select Make *EXE* from the compile menu, or you can press the F9 key.

**Executing a Program**

If the program is compiled and linked without errors, the program is executed by selecting

Run from the Run Menu or by pressing the F10 key.

**The Development Cycle**

If every program worked the first time you tried it that would be the complete development cycle: Write the program, compile the source code, link the program, and run it.

Unfortunately, almost every program, no matter how trivial, can and will have errors, or bugs, in the program. Some bugs will cause the compile to fail, some will cause the link to fail, and some will only show up when you run the program. Whatever type of bug you find, you must fix it, and that involves editing your source code, recompiling and relinking, and then rerunning the program.

**Correcting Errors**

If the compiler recognizes some error, it will let you know through the Compiler window.

You’ll see that the number of errors is not listed as 0, and the word “Error” appears instead of the word “Success” at the bottom of the window. The errors are to be removed by returning to the edit window. Usually these errors are a result of a typing mistake. The compiler will not only tell you what you did wrong; they’ll point you to the exact place in your code where you made the mistake.

**Exiting IDE**

An Edit window may be closed in a number of different ways. You can click on the small square in the upper left corner, you can select **close** from the **window** menu, or you can press the [ALT+F4] combination.

**Building Blocks of Programming Language:**

In any language there are certain building blocks:

* Constants
* Variables
* Operators
* Methods to get input from user (cin , getch( ) etc.)
* Methods to display output (Escape Sequences etc.) and so on.

**Variables and Constants**

If the value of an item can be changed in the program then it is a variable. If it will not change then that item is a constant. The various variable types (also called *data type*) in C are: *int*, *float*, *char*, *long* etc. For constants, the keyword ***const*** is added before declaration.

**Operators**

There are various types of operators that may be placed in three categories:

***Basic*: + - \* / %**

***Assignment*: = += - = \*= /= %=**

(++, - - may also be considered as assignment operators)

***Relational*: < > <= >= == !=**

**Escape Sequences**

Escape Sequence causes the program to **escape** from the normal interpretation of a string, so that the next character is recognized as having a special meaning. The back slash “\” character is called the **Escape Character”**. The escape sequence includes the following:

\n => new line

\b => back space

\r => carriage return

\” => double quotations

**Getting Input From the User**

The input from the user can be taken by the following techniques: scanf( ), getch( ), getche(),getchar( ) etc.

**Examples:**

1. **Implementing a Simple C++ Program**

#include<iostream>

using namespace std;

int main()

{

cout<<"\n Hello World";

return 0;

}



1. **Demonstrating the fundamentals of C++ Language**

#include<iostream>

using namespace std;

int main()

{

int num1,num2,sum,product;

cout<<" The program takes two numbers as input and prints their sum and product"<<endl;

cout<<" Enter first number:";

cin>>num1;

cout<<" Enter second number:";

cin>>num2;

sum=num1+num2;

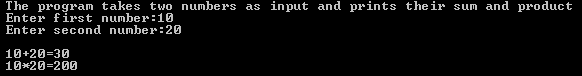
product=num1\*num2;

cout<<"\n "<<num1<<"+"<<num2<<"="<<sum;

cout<<"\n "<<num1<<"\*"<<num2<<"="<<product;

return 0;

}



**Lab Tasks**

1. Write a program to calculate the Area ( A= r2 ) and circumference of a circle (*C=r*), where r = radius is taken as input and is declared as a constant. The precision of should be the number of characters in your name. Display the result to 4 decimal places.

#include <iostream>

#include <iomanip> // For setprecision

using namespace std;

int main() {

const double PI = 3.141592653589;

double radius;

cout << "Enter radius of the circle: ";

cin >> radius;

double area = PI \* radius \* radius;

double circumference = 2 \* PI \* radius;

cout << fixed << setprecision(4);

cout << "Area = " << area << endl;

cout << "Circumference = " << circumference << endl;

return 0;

}

1. Write single C ++ statement to output the following on the screen:

My name is “*Your Name*”

And roll number is “*Your\_roll\_no*”

I am a student of “*Your Department*”

#include <iostream>

using namespace std;

int main() {

cout << "My name is \"Alex Johnson\"\nAnd roll number is \"12345\"\nI am a student of "software engineering"" << endl;

return 0;

}

1. Write a program that perform basic arithmetic operations by getting values from User.

#include <iostream>

using namespace std;

int main() {

double num1, num2;

cout << "Enter first number: ";

cin >> num1;

cout << "Enter second number: ";

cin >> num2;

cout << "Addition: " << (num1 + num2) << endl;

cout << "Subtraction: " << (num1 - num2) << endl;

cout << "Multiplication: " << (num1 \* num2) << endl;

if (num2 != 0)

cout << "Division: " << (num1 / num2) << endl;

else

cout << "Division by zero is not allowed." << endl;

return 0;

}

1. Write a program that accept value from user and display its square and cube without using predefined functions?

#include <iostream>

using namespace std;

int main() {

int number;

cout << "Enter a number: ";

cin >> number;

int square = number \* number;

int cube = number \* number \* number;

cout << "Square: " << square << endl;

cout << "Cube: " << cube << endl;

return 0;

}

**Lab -02**

**To get familiar with Operators and Math library**

## Theory:

In the C/C++ programming language, the C/C++ Standard Library is a collection of classes and functions, which are written in the core language. The library includes several header files. Most frequently used header files are iostream, conio.h followed by preprocessor directives #include.

**#include<iostream>for C++ / #include<stdio.h> for C abbreviated for input/ output stream**

**#include<conio.h> abbreviated for console input/ output**

**#include<cmath> for C++ and #include<math.h> for C is used as mathematical functionalities like power, square root, cube root etc.**

**Note:** .h extension is used with those header files that are included in both C and C++.

Now, C++ has some collection of names referred to as the namespace. This includes various names which are not used for other purposes. For example **cin, cout**, basic input-output functions define in **std** namespace. After including header files **using namespace std** is added in global space in order to access various function in any program.

In different programs input is required to be entered from user at rum time and output is generate don the monitor screen so for this two functions are defined in **namespace std** that in Console input (**cin**), Console output (**cout**)

After declaring all header files and standard namespace, program will declare main function that is the gateway or the starting point of the program. In C++ all functions carried ( ) parenthesis.

**int** **main ()** In C++ main function has return type integer that indicates an integer will return to end up the functionality of main function and return a value to operating system.

**Syntax of Program:**

**#include <iostream>**

**#include<conio.h>**

**using namespace std;**

**int main(){**

**return 0; }**

Curly braces {} indicates the starting and ending of the program, return 0; statement returns the 0 value that indicates normal termination of program.

In main() function all the problem solving codes are written that involve different data. For example if a program is taking personal details from user to this program will includes name, mobile number, address, gender etc. that indicates some data are of only characters like name or some have numeric digits and alphabets like address.

**Data Types :**

Data types indicates type of data as string, character, integer, float, double, bool etc.

* **declaring a boolean variable which can only holds true or false value**

bool b1 = true;

* **declaring a string value**

string g= “Ali”;

string f= “6758”;

string y= “57.6”;

string d=“true”;

* **declaring a character value**

char z= ‘b’;

char j= ‘4’;

* **declaring a decimal value**

float h=25.2;

double u=612345.56;

* **declaring a integer value**

int u=3;

**Keywords:**

Like C and other languages C++ also have some keywords that cannot be used for variable names.

**Operators:**

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provide the following types of operators −

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operators

## Arithmetic Operators

There are following arithmetic operators supported by C++ language

*Assume variable A holds 10 and variable B holds 20, then*

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Adds two operands | A + B will give 30 |
| - | Subtracts second operand from the first | A - B will give -10 |
| \* | Multiplies both operands | A \* B will give 200 |
| / | Divides numerator by de-numerator | B / A will give 2 |
| % | Modulus Operator and remainder of after an **integer** division | B % A will give 0 |
| ++ | [Increment operator](https://www.tutorialspoint.com/cplusplus/cpp_increment_decrement_operators.htm), increases integer value by one | A++ will give 11 |
| -- | [Decrement operator](https://www.tutorialspoint.com/cplusplus/cpp_increment_decrement_operators.htm), decreases integer value by one | A-- will give 9 |

## Relational Operators:

There are following relational operators supported by C++ language

*Assume variable A holds 10 and variable B holds 20, then –*

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = = | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

## Logical Operators:

There are following logical operators supported by C++ language.

Assume variable A holds 1 and variable B holds 0, then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Called Logical AND operator. If both the operands are non-zero, then condition becomes true. | (A && B) is false. |
| || | Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true. | (A || B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false. | ! (A && B) is true. |

## Bitwise Operators:

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P** | **q** | **p & q** | **p | q** | **p ^ q** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Assume if A = 60; and B = 13; now in binary format they will be as follows −

A = 0011 1100

B = 0000 1101

-----------------------------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A  = 1100 0011

The Bitwise operators supported by C++ language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12 which is 0000 1100 |
| | | Binary OR Operator copies a bit if it exists in either operand. | (A | B) will give 61 which is 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49 which is 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -61 which is 1100 0011 |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 0000 1111 |

## Assignment Operators:

There are following assignment operators supported by C++ language −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Simple assignment operator, Assigns values from right side operands to left side operand. | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand. | C -= A is equivalent to C = C – A |

**Example: Write a program that takes two integer input from user and apply assignment operators?**

#include <iostream>

#include<conio.h>

using namespace std;

int main()

{int a,b;

cout<< "Enter first number=";

cin>>a;

cout<<"Enter second number=";

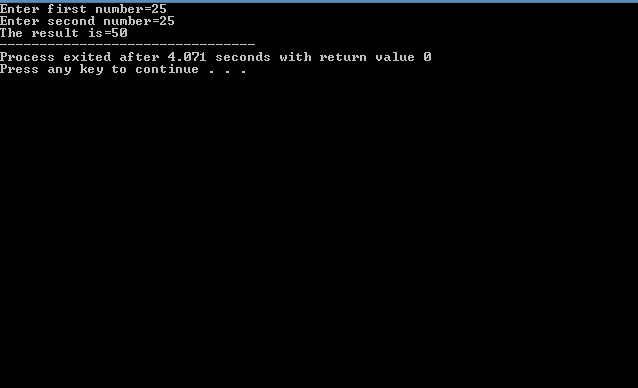
cin>>b;

a+=b;

cout<<"The result is="<<a;

return 0;

}

****

The <cmath>**header file** declares a set of functions to perform mathematical operations such as: sqrt() to calculate square root, log() to find natural logarithm of a number etc. Some of the frequently used functions of this library are mentioned as under;

* **Square root :** used to calculate the square root of the given argument

**double sqrt(double x)**

* **Cube root:** used to calculate the cube root of the given argument

**double cbrt(double x)**

* **Power:** determines the power for given exponent

**double pow(double base, double exponent);**

* **Absolute:** returns the absolute value of integer value

**int x = -5; int a = abs(x);**

* **Sine angle:** Use to calculate the sine value of given angle

**double sin (double x); //same as cos angle tangent angle**

* **Round off**: The round() function in C++ returns the integral value that is nearest to the argument, with halfway cases rounded away from zero

**double round(double x);**

* **Logarithm** returns logarithmic value of given argument

**double log10 (double x);**

* The **floor()** function in C++ returns the largest possible integer value which is less than or equal to the given argument.

**double floor(double x);**

* The **ceil()** function in C++ returns the smallest possible integer value which is greater than or equal to the given argument.

**double ceil(double x);**

**Lab Tasks**

Q1: Design a program to take integer input from the user, store it in a variable named num1, and apply post-increment (num1 ++), pre-increment (++num1), post-decrement (num1 --), and pre-decrement (num1--). Display each result in a new line with a track of the previous value before applying each increment or decrement.

#include <iostream>

using namespace std;

int main() {

int num1;

cout << "Enter an integer: ";

cin >> num1;

cout << "Before post-increment: " << num1 << endl;

cout << "After post-increment (output): " << num1++ << endl;

cout << "Value now: " << num1 << endl;

cout << "Before pre-increment: " << num1 << endl;

cout << "After pre-increment: " << ++num1 << endl;

cout << "Before post-decrement: " << num1 << endl;

cout << "After post-decrement (output): " << num1-- << endl;

cout << "Value now: " << num1 << endl;

cout << "Before pre-decrement: " << num1 << endl;

cout << "After pre-decrement: " << --num1 << endl;

return 0;

}

Q2. Write a program that takes float or double input from the user and returns the entered input's ceiling and floor value.

#include <iostream>

#include <cmath>

using namespace std;

int main() {

double input;

cout << "Enter a floating-point number: ";

cin >> input;

cout << "Ceiling value: " << ceil(input) << endl;

cout << "Floor value: " << floor(input) << endl;

return 0;

}

Q3. Write a program to calculate the cube root of every integer input value entered at run time.

#include <iostream>

#include <cmath>

using namespace std;

int main() {

int num;

cout << "Enter an integer to find its cube root: ";

cin >> num;

double cubeRoot = cbrt(num);

cout << "Cube root of " << num << " is " << cubeRoot << endl;

return 0;

}

Q4. Design a program to execute the expression a+b-c and print the final result where a,b,c is the console integer input from the user.

#include <iostream>

using namespace std;

int main() {

int a, b, c;

cout << "Enter value for a: ";

cin >> a;

cout << "Enter value for b: ";

cin >> b;

cout << "Enter value for c: ";

cin >> c;

int result = a + b - c;

cout << "Result of a + b - c is: " << result << endl;

return 0;

}

Q5. Design a program that does not take input from the user and calculates 65^3+pi. Where pi is a constant value of 3.142 should be declared as constant

#include <iostream>

#include <cmath>

using namespace std;

int main() {

const double pi = 3.142;

int base = 65;

double result = pow(base, 3) + pi;

cout << "Result of 65^3 + pi is: " << result << endl;

return 0;

}

**Lab- 03**

**To get familiar with the use of basic control structures in C++: if-else and else-if statements.**

**Theory:**

The ability to control the flow of your program, letting it make decisions i.e Control Structures on what code to execute, is valuable to the programmer. “if” statement allows you to control if a program enters a section of code or not based on whether a given condition is true or false, the meaning of TRUE and FALSE in computer terminology. A true statement is one that evaluates to a non-zero number. A false statement evaluates to zero, when you perform comparison with the relational operators.

1. **The IF Statement:**

This is the simplest form of the branching statements. It takes an expression in parenthesis and a statement or block of statements. If the expression is true, then the statement or block of statements gets executed otherwise these statements are skipped.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Checks if the value of two operands is equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the value of two operands is equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |
| && | Called Logical AND operator. If both the operands are non-zero, then then condition becomes true. | (A && B) is true. |
| || | Called Logical OR Operator. If any of the two operands is non zero, then then condition becomes true. | (A || B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logic. | !(A && B) is false. |

**Syntax:**

**If (TRUE ) {**

**/\* between the braces is the body of the if statement \*/**

**Execute all statements inside the body**

**}**

**Example:**

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

char a;

cout<<"Enter any character\n";

a=getche();

if(a=='y')

{

cout<<"\n IQRA UNIVERSITY";

}

getch();

return 0;

}**The IF-ELSE Statement:**

Sometimes when the condition in if statement evaluates to false, it would be nice to execute some code instead of the code executed when the statement evaluates to true. The "else" statement effectively says that whatever code after it (whether a single line or code between brackets) is executed after if statement is FALSE.   
**Syntax:**

**if ( TRUE ) {**

**/\* Execute these statements if TRUE \*/**

**}**

**else {**

**/\* Execute these statements if FALSE \*/**

**}**

**Example:**

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

int x,y;

cout<<" Enter value for x :";

cin>>x;

cout<<" Enter value for y :";

cin>>y;

if ( x > y ){

cout<<" X is large number="<<x;

}

else{

cout<<" Y is large number="<<y;

}

getch();

}

1. **The ELSE-IF Statement:**

Another use of else is when there are multiple conditional statements that may all evaluate to true, yet you want only one if statement's body to execute. You can use an "else if" statement following an if statement and its body; that way, if the first statement is true, the "else if" will be ignored, but if the if statement is false, it will then check the condition for the else if statement. If the if statement was true, the else statement will not be checked. It is possible to use numerous else if statements to ensure that only one block of code is executed.

**Syntax:**

**if(condition)  
statement 1;  
else if (condition)  
          statement 2;  
          .....................  
          .....................  
else if(condition)  
         statement ;**

**Example:**

#include<iostream>

#include<conio.h>

using namespace std;

int main() /\* Most important part of the program! \*/

{ int age; /\* Need a variable... \*/

cout<< "\n Please enter your age: " ; /\* Asks for age \*/

cin>>age; /\* The input is put in age \*/

if ( age < 50) { /\* If the age is less than 50 \*/

cout<<" You are young!\n" ; /\* Just to show you it works... \*/

}

else if ( age == 50 ) { /\* I use else just to show an example \*/

cout<<" You are old\n" ;

}

else {

cout<<" You are really old\n" ; /\* Executed if no other statement is \*/

}

getch();

}

**Lab Tasks**

1. Write a program that display whether a number is even or odd?

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter an integer: ";

cin >> num;

if (num % 2 == 0)

cout << num << " is Even." << endl;

else

cout << num << " is Odd." << endl;

return 0;

}

1. Write a program that accept three values from a user and print the highest value from them?

#include <iostream>

using namespace std;

int main() {

int a, b, c;

cout << "Enter three numbers: ";

cin >> a >> b >> c;

if (a >= b && a >= c)

cout << "Highest value is: " << a << endl;

else if (b >= a && b >= c)

cout << "Highest value is: " << b << endl;

else

cout << "Highest value is: " << c << endl;

return 0;

}

1. Write a program that design a program which shows at the top Transport Route decision chart through else-if statement that takes integer distance in kilometers from residence to university and checks the condition in the following format:

* If distance is less than and equals to 1km than print “Come university by foot”
* If distance is above 1 km but below and equals to 3km than print “Come University by public transport”
* If distance is above 3km less than 5 km than print “Come university by university point”
* else change the university.
* Also, apply condition that indicates negative value of distance as invalid input.

#include <iostream>

using namespace std;

int main() {

float distance;

cout << "Enter distance in kilometers: ";

cin >> distance;

if (distance < 0)

cout << "Invalid input: Distance can't be negative." << endl;

else if (distance <= 1)

cout << "Come university by foot." << endl;

else if (distance <= 3)

cout << "Come University by public transport." << endl;

else if (distance < 5)

cout << "Come university by university point." << endl;

else

cout << "Change the university." << endl;

return 0;

}

1. Take integer input as the date and character as month from user and decide the zodiac sign

#include <iostream>

#include <string>

using namespace std;

int main() {

int day;

string month;

cout << "Enter date (1-31): ";

cin >> day;

cout << "Enter month (e.g., March): ";

cin >> month;

string zodiac;

if ((month == "March" && day >= 21) || (month == "April" && day <= 19))

zodiac = "Aries";

else if ((month == "April" && day >= 20) || (month == "May" && day <= 20))

zodiac = "Taurus";

else if ((month == "May" && day >= 21) || (month == "June" && day <= 20))

zodiac = "Gemini";

else if ((month == "June" && day >= 21) || (month == "July" && day <= 22))

zodiac = "Cancer";

else if ((month == "July" && day >= 23) || (month == "August" && day <= 22))

zodiac = "Leo";

else if ((month == "August" && day >= 23) || (month == "September" && day <= 22))

zodiac = "Virgo";

else if ((month == "September" && day >= 23) || (month == "October" && day <= 22))

zodiac = "Libra";

else if ((month == "October" && day >= 23) || (month == "November" && day <= 21))

zodiac = "Scorpio";

else if ((month == "November" && day >= 22) || (month == "December" && day <= 21))

zodiac = "Sagittarius";

else if ((month == "December" && day >= 22) || (month == "January" && day <= 19))

zodiac = "Capricorn";

else if ((month == "January" && day >= 20) || (month == "February" && day <= 18))

zodiac = "Aquarius";

else if ((month == "February" && day >= 19) || (month == "March" && day <= 20))

zodiac = "Pisces";

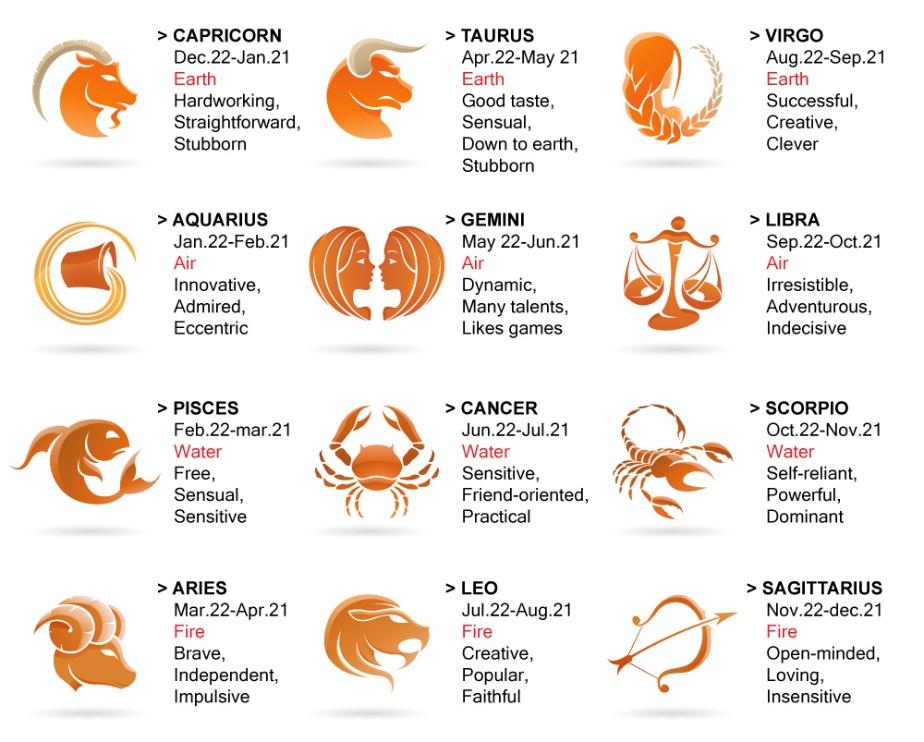
else

zodiac = "Invalid date or month";

cout << "Zodiac sign is: " << zodiac << endl;

return 0;

}

****

**Lab- 04**

**To get familiar with the use of nested if statements for more complex decision-making scenarios.**

### **Theory**

A **nested if statement** is an if statement placed inside another if or else statement. It allows you to test a condition within another condition, creating a more complex decision-making structure.

In C++, nested if statements are used when there are multiple conditions that need to be checked in a hierarchical manner. It is helpful when you want to test one condition first, and then, based on that condition, check further conditions

### **Syntax of Nested If:**

if (condition1) {

// First condition is true, check further conditions inside this block

if (condition2) {

// Second condition is true

// Code to execute if both condition1 and condition2 are true

} else {

// Code to execute if condition2 is false

}

} else {

// Code to execute if condition1 is false

if (condition3) {

// Third condition is checked if condition1 is false

}

}

### **Example:**

Here’s an example of a **nested if** in C++:

#include <iostream>

using namespace std;

int main() {

int age;

cout << "Enter your age: ";

cin >> age;

if (age >= 18) { // Check if age is greater than or equal to 18

if (age >= 65) { // Nested condition to check if age is greater than or equal to 65

cout << "You are a senior citizen." << endl;

} else {

cout << "You are an adult." << endl;

}

} else {

cout << "You are a minor." << endl; // Code executed if the first condition is false

}

return 0;

}

**Explanation:**

* The program first checks if the age is **greater than or equal to 18** (the first condition).
* If true, it checks a **nested condition** to see if the age is **greater than or equal to 65**.
* If the second condition is true, the user is identified as a **senior citizen**.
* If the first condition fails (age is less than 18), the program will output that the user is a **minor**.
* If the first condition is true but the second condition fails (age is between 18 and 64), the program will output that the user is an **adult**.

**Lab Tasks**

**Q1: Write a C++ program** to create a simple login system for a computer with three types of user accounts: **Admin**, **Faculty**, and **Guest**. The login system should check both the **username** and **password**, and grant access based on the following conditions:

1. **Admin**:
   * Username: admin
   * Password: admin123
   * **Access Rights**: Full access to the system (view/edit files, install software, and change settings).
   * If the password is incorrect, output "Invalid password for Admin. Access denied."
2. **Faculty**:
   * Username: faculty
   * Password: faculty123
   * **Access Rights**: Limited access (view and edit course-related files, but cannot install software or change system settings).
   * If the password is incorrect, output "Invalid password for Faculty. Access denied."
3. **Guest**:
   * Username: guest
   * Password: guest123
   * **Access Rights**: Very limited access (only view public files, cannot make changes to the system).
   * If the password is incorrect, output "Invalid password for Guest. Access denied."
4. **Invalid Username**:
   * If the username is not one of the three valid options (admin, faculty, or guest), output: "Invalid username. Please try again."

#include <iostream>

#include <string>

using namespace std;

int main() {

string username, password;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

if (username == "admin") {

if (password == "admin123") {

cout << "\nWelcome Admin!" << endl;

cout << "Access Rights: Full access (view/edit files, install software, and change settings)." << endl;

} else {

cout << "Invalid password for Admin. Access denied." << endl;

}

}

else if (username == "faculty") {

if (password == "faculty123") {

cout << "\nWelcome Faculty!" << endl;

cout << "Access Rights: Limited access (view and edit course-related files, no software installation or settings changes)." << endl;

} else {

cout << "Invalid password for Faculty. Access denied." << endl;

}

}

else if (username == "guest") {

if (password == "guest123") {

cout << "\nWelcome Guest!" << endl;

cout << "Access Rights: Very limited access (only view public files)." << endl;

} else {

cout << "Invalid password for Guest. Access denied." << endl;

}

}

else {

cout << "Invalid username. Please try again." << endl;

}

return 0;

}

**Lab - 05**

**To understand the structure and procedure of switch cases in C++**

## Theory

**Switch Statement in C++**

Switch case statements are a substitute for long if statements that compare a variable to several integral values. The switch statement is a multi-way branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. Switch is a control statement that allows a value to change control of execution.

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

**Syntax for integer input:**

//here n is integer

switch (n)

{

case 1: // code to be executed if n = 1;

break;

case 2: // code to be executed if n = 2;

break;

default: // code to be executed if n doesn't match any cases

}

**Syntax for character input:**

//here ch is character

switch (ch)

{

case ‘a’: // code to be executed if n = 1;

break;

case ‘b’: // code to be executed if n = 2;

break;

default: // code to be executed if n doesn't match any cases

}

***Note: character is written in single quotes ‘’ where integer, float values are written without ‘’.***

**EXAMPLE # 1: Write a program to make a Calculator using switch case**

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

char c, op;

double operand1,operand2;

cout<<"Enter first number,operator,second number:";

cin>>operand1>>op>>operand2;

switch(op)

{

case '+':

cout<<"Answer="<<operand1+operand2;

break;

case '-':

cout<<"Answer="<<operand1-operand2;

break;

case '\*':

cout<<"Answer="<<operand1\*operand2;

break;

case '/':

cout<<"Answer="<<operand1/operand2;

break;

default:

cout<<"invalid operator";

}

return 0;

}

**EXAMPLE #2: Write a program to determine percentage from the grade.**

#include <iostream>

#include<conio.h>

using namespace std;

int main()

{

char grade;

cout<<"\n Enter your Grade: ";

grade=getche();

switch(grade)

{

case 'a':

cout<<"\n Your percentage is 80 or above 80 ";

break;

case 'b':

cout<<"\n Your percentage is in 79-70 ";

break;

case 'c':

cout<<"\n Your percentage is in 69-60 ";

break;

case 'd':

cout<<"\n Your percentage is in 59-50 ";

break;

default:

cout<<"\n Your percentage is below 50 ";

}

return 0;

}

**Lab Tasks**

1. **Write a program that display a Menu, and perform that operation**

*Menu*

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

1. *Even or Odd*
2. *Addition*
3. *Subtraction*
4. *Division*
5. *Multiplication*
6. *Quit*

*#include <iostream>*

*using namespace std;*

*int main() {*

*int choice;*

*do {*

*cout << "\nMenu\n\*\*\*\*\*\*" << endl;*

*cout << "1. Even or Odd" << endl;*

*cout << "2. Addition" << endl;*

*cout << "3. Subtraction" << endl;*

*cout << "4. Division" << endl;*

*cout << "5. Multiplication" << endl;*

*cout << "6. Quit" << endl;*

*cout << "Enter your choice: ";*

*cin >> choice;*

*int a, b;*

*switch (choice) {*

*case 1:*

*cout << "Enter a number: ";*

*cin >> a;*

*if (a % 2 == 0)*

*cout << a << " is Even." << endl;*

*else*

*cout << a << " is Odd." << endl;*

*break;*

*case 2:*

*cout << "Enter two numbers: ";*

*cin >> a >> b;*

*cout << "Result: " << a + b << endl;*

*break;*

*case 3:*

*cout << "Enter two numbers: ";*

*cin >> a >> b;*

*cout << "Result: " << a - b << endl;*

*break;*

*case 4:*

*cout << "Enter two numbers: ";*

*cin >> a >> b;*

*if (b != 0)*

*cout << "Result: " << (float)a / b << endl;*

*else*

*cout << "Division by zero is not allowed." << endl;*

*break;*

*case 5:*

*cout << "Enter two numbers: ";*

*cin >> a >> b;*

*cout << "Result: " << a \* b << endl;*

*break;*

*case 6:*

*cout << "Exiting program..." << endl;*

*break;*

*default:*

*cout << "Invalid choice. Try again!" << endl;*

*}*

*} while (choice != 6);*

*return 0;*

*}*

1. **Write a program whose display heading on the top as “Admissions in Iqra University” that offers admission in Computer Science as choice 1, Software Engineering as choice 2, Computer Engineering as choice 3, Business management as choice 4, Pharmacy as choice 5.**

*The program takes integer input as choice number. And their percentage in float.*

*On the basis of their choice decide whether a student is eligible for this department or not.*

*If eligible display message “Your admission is confirmed”. Else display message “change your field”.*

*Selection criteria*

* *>=85 Computer Engineering*
* *Less than 85 and greater and equals to 82 Computer Science*
* *Less than 82 and greater and equals to 79 Software Engineering*
* *Less than 86 and greater and equals to 76 Business management*
* *Less than 88 and greater and equals to 70 Pharmacy*

*#include <iostream>*

*using namespace std;*

*int main() {*

*int choice;*

*float percentage;*

*cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Admissions in Iqra University \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";*

*cout << "1. Computer Science\n";*

*cout << "2. Software Engineering\n";*

*cout << "3. Computer Engineering\n";*

*cout << "4. Business Management\n";*

*cout << "5. Pharmacy\n";*

*cout << "Enter your program choice (1-5): ";*

*cin >> choice;*

*cout << "Enter your percentage: ";*

*cin >> percentage;*

*bool eligible = false;*

*switch (choice) {*

*case 1:*

*if (percentage >= 82 && percentage < 85)*

*eligible = true;*

*break;*

*case 2:*

*if (percentage >= 79 && percentage < 82)*

*eligible = true;*

*break;*

*case 3:*

*if (percentage >= 85)*

*eligible = true;*

*break;*

*case 4:*

*if (percentage >= 76 && percentage < 86)*

*eligible = true;*

*break;*

*case 5:*

*if (percentage >= 70 && percentage < 88)*

*eligible = true;*

*break;*

*default:*

*cout << "Invalid choice!" << endl;*

*return 1;*

*}*

*if (eligible)*

*cout << "Your admission is confirmed." << endl;*

*else*

*cout << "Change your field." << endl;*

*return 0;*

*}*

1. **Write a program whose display heading on the top as “Grading Scheme in Iqra University” that takes grades as input as display their remarks as per university criteria.**

*A= You have shown Excellent performance*

*B= Your performance is average.*

*C= You are below average. Needs improvement!!*

*D= You need serious hard work!!*

*E= Your are failed*

*#include <iostream>*

*using namespace std;*

*int main() {*

*char grade;*

*cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Grading Scheme in Iqra University \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";*

*cout << "Enter your grade (A/B/C/D/E): ";*

*cin >> grade;*

*switch (toupper(grade)) {*

*case 'A':*

*cout << "You have shown Excellent performance." << endl;*

*break;*

*case 'B':*

*cout << "Your performance is average." << endl;*

*break;*

*case 'C':*

*cout << "You are below average. Needs improvement!!" << endl;*

*break;*

*case 'D':*

*cout << "You need serious hard work!!" << endl;*

*break;*

*case 'E':*

*cout << "You are failed." << endl;*

*break;*

*default:*

*cout << "Invalid grade entered." << endl;*

*}*

*return 0;*

*}*

**Lab - 06**

**To understand the structure and procedure of Loops in C++**

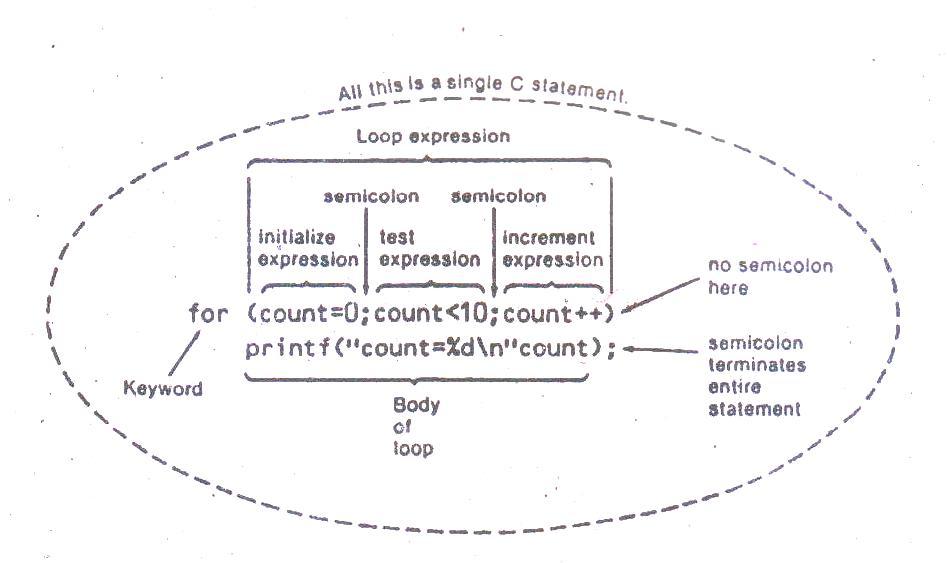
**Theory:**

In computer programming, a loop is a sequence of instruction s that is continually repeated until a certain condition is reached. Typically, a certain process 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 or not.

1. **The For Loop:**

The FOR loop is found in many procedural languages which repeatedly executes some instructions until given condition is true. In C++, the FOR loop is written in the form of expression in which initialization, testing of condition and increment or decrement value is include.

* **Structure of the For Loop:**

****

* **Initalization Expression:**

In initializing of expression, the given variables initialize the starting value. It always executed as soon as the loop is entered. We can start initialization at any number however in the given example variable count initializing or count from 0.

**Example:**

int var;

( var = 5; )

* **Test Expression:**

In secondpart of the expression variable testing the given condition. In given example count is less than 10, it test each time through the loop. If the test expression is true i.e. count<10 the body of the loop will be go to prntf statement and if the expression is false i.e. count is equal to 10 or more, the loop will be terminated.

**Example:**

int var;

( var >= 1;)

* **Increment / Decrement Expression:**

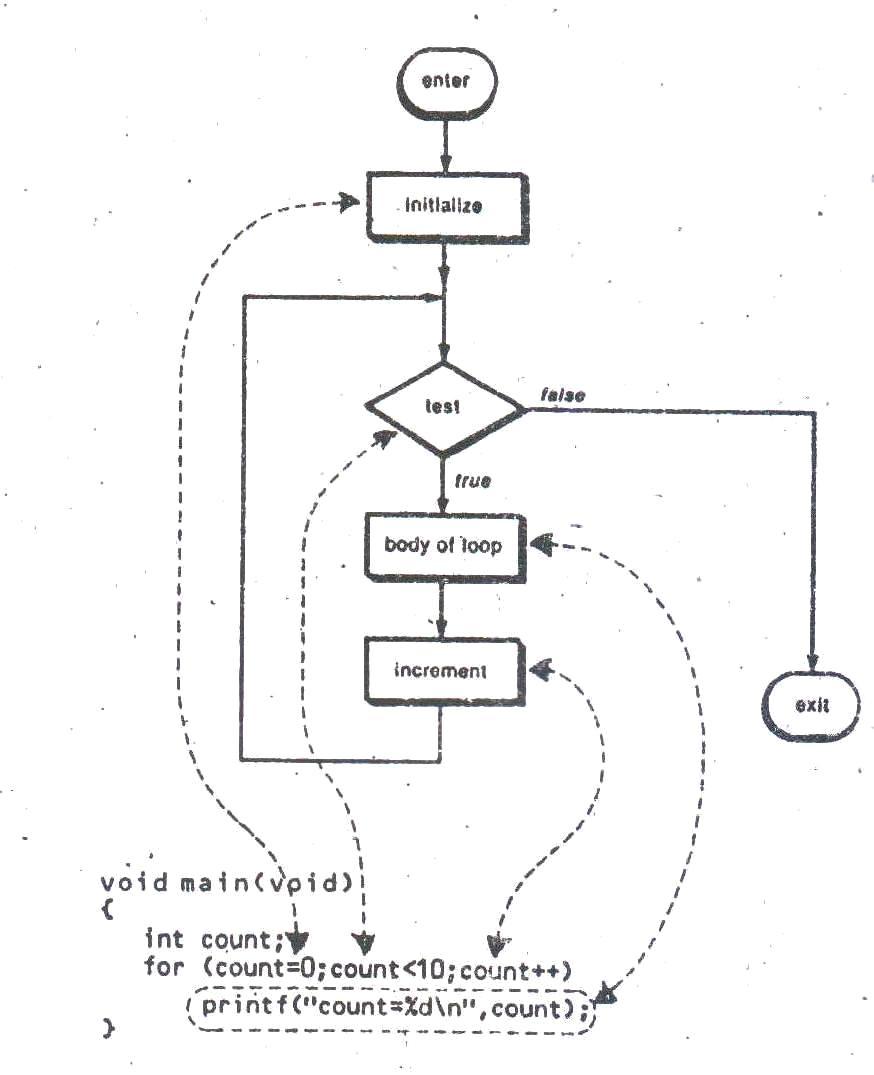
In third part of the expression we use increment or decrement operators to make continuous counter. In this example we use count ++ i.e. increment operator for increment in each step, the variable count each time the loop and then execute.

**Example:**

int var;

( var++ ) / ( var - - )

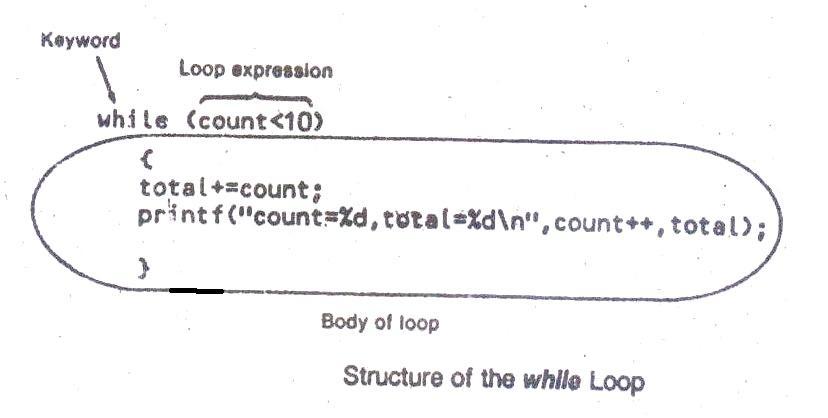
* **Operational Flow Chart of For Loop:**

****

1. **The While Loop:**

The while loop can be thought of as a repeating [if-statement](http://en.wikipedia.org/wiki/Conditional_%28programming%29). The while construct consists of a block of code and a condition. The condition is evaluated, and if the condition is true, the code within the block is executed. This repeats until the condition becomes false. Because while loop checks the condition before the block is executed, the control structure is often also known as a pre-test loop. The following flow chart illustrates the while loop in C.

* **Structure of While Loop:**

****

* **Test Expression:**

The loop executes as long as the given logical expression between parentheses after *while* is true. When expression is false, execution continues with the statement following the loop block.

**while (expression)**

**{**

**// statements**

**}**

**Example:**

#include <iostream>

#include<conio.h>

using namespace std;

int main()

{

int x = 10;

int i = 0; // using while loop statement

while(i < x){

i++;

cout<<i<<endl;

} // when number 5 found, escape loop body

int numberFound= 5;

int j = 1;

while(j < x){

if(j == numberFound){

cout<<"number found"<<endl;

}

cout<<j<< "..keep finding" <<endl;

j++;

}

getch();

}

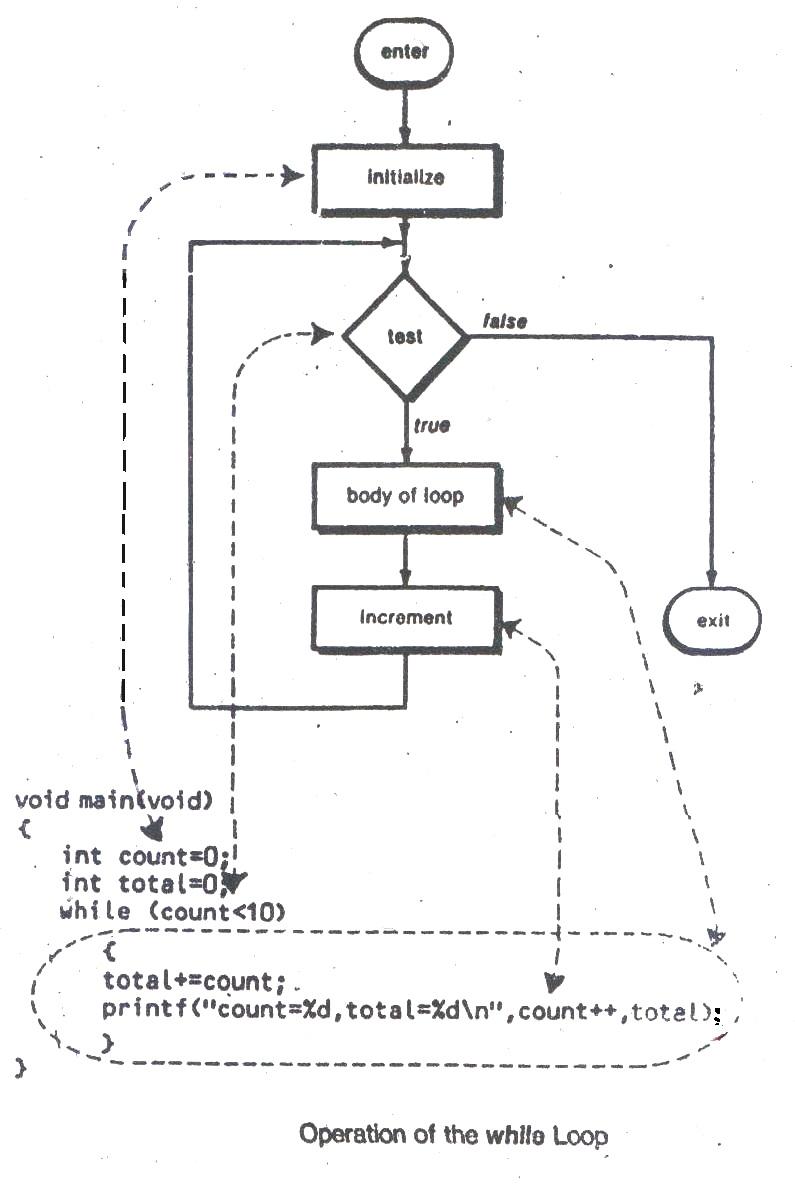
**Output:**

1  
2  
3  
4

5  
6

7  
8  
9  
10  
1...keep finding  
2...keep finding  
3...keep finding  
4...keep finding  
number found.

* **Operational Flow Chart:**

****

**Lab Tasks**

1. Write a C++ program that asks the user to enter the number of subjects they have. Use a `for` loop to iterate through each subject, allowing the user to input the subject name and corresponding marks. At the end of the program, display the total marks for all subjects.

#include <iostream>

using namespace std;

int main() {

int numSubjects;

string subjectName;

int marks, totalMarks = 0;

cout << "Enter the number of subjects: ";

cin >> numSubjects;

for (int i = 1; i <= numSubjects; i++) {

cout << "Enter the name of subject " << i << ": ";

cin >> subjectName;

cout << "Enter marks for " << subjectName << ": ";

cin >> marks;

totalMarks += marks;

}

cout << "Total marks: " << totalMarks << endl;

return 0;

}

2. Create a number guessing game using a `while` loop. The program should select a random number within a specified range (e.g., 1 to 100). Allow the user to guess the number and provide feedback on whether their guess is too high, too low, or correct. Count the number of turns taken to guess the correct number and display this count when the game ends.

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int main() {

srand(time(0)); // Seed the random number generator

int target = rand() % 100 + 1;

int guess, attempts = 0;

cout << "Guess the number (1 to 100):\n";

while (true) {

cout << "Enter your guess: ";

cin >> guess;

attempts++;

if (guess < target) {

cout << "Too low!\n";

} else if (guess > target) {

cout << "Too high!\n";

} else {

cout << "Correct! You guessed it in " << attempts << " attempts.\n";

break;

}

}

return 0;

}

1. Write a program that generates a multiplication table using a `for` loop. Prompt the user to enter the number for which they want the table and display it up to a specified multiple (e.g., up to 10).

#include <iostream>

using namespace std;

int main() {

int number, limit;

cout << "Enter the number: ";

cin >> number;

cout << "Enter the multiple limit: ";

cin >> limit;

for (int i = 1; i <= limit; i++) {

cout << number << " x " << i << " = " << number \* i << endl;

}

return 0;

}

1. Write a program that calculates the factorial of a given number using a `while` loop. Prompt the user to enter a non-negative integer and display the factorial of that number.

#include <iostream>

using namespace std;

int main() {

int num;

unsigned long long factorial = 1;

cout << "Enter a non-negative integer: ";

cin >> num;

if (num < 0) {

cout << "Factorial is not defined for negative numbers.\n";

return 1;

}

int i = 1;

while (i <= num) {

factorial \*= i;

i++;

}

cout << "Factorial of " << num << " is " << factorial << endl;

return 0;

}

1. Write a program using a `while` loop to manage fruit stock. Set a stock threshold level (e.g., 5 units). For each fruit type, check the current stock level. If the stock level is below the threshold, prompt the user to refill it until it meets or exceeds the threshold. Display the stock level of each fruit after refilling.

#include <iostream>

using namespace std;

int main() {

const int threshold = 5;

int numFruits;

cout << "Enter the number of fruit types: ";

cin >> numFruits;

for (int i = 0; i < numFruits; i++) {

string fruit;

int stock;

cout << "Enter fruit name: ";

cin >> fruit;

cout << "Enter current stock for " << fruit << ": ";

cin >> stock;

while (stock < threshold) {

cout << "Stock for " << fruit << " is below threshold (" << stock << "). Please refill: ";

int refill;

cin >> refill;

stock += refill;

}

cout << "Final stock for " << fruit << ": " << stock << endl;

}

return 0;

}

1. Write a program that implements a number guessing game. Use a `for` loop to give the player 6 attempts to guess a randomly chosen number within a specified range (e.g., 1 to 100). After each guess, provide feedback on whether the guess is too high, too low, or correct. If the player runs out of attempts, end the game with a message indicating the correct answer.

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int main() {

srand(time(0));

int target = rand() % 100 + 1;

int guess;

cout << "You have 6 attempts to guess the number (1 to 100):\n";

for (int i = 1; i <= 6; i++) {

cout << "Attempt " << i << ": ";

cin >> guess;

if (guess < target) {

cout << "Too low!\n";

} else if (guess > target) {

cout << "Too high!\n";

} else {

cout << "Correct! You guessed it in " << i << " attempts.\n";

return 0;

}

}

cout << "Sorry, you ran out of attempts. The correct number was " << target << ".\n";

return 0;

}

1. Explain the difference between using a `for` loop (with a limited number of attempts) and a `while` loop (without a fixed attempt limit) in the context of this game.

| **Feature** | **for Loop (Fixed Attempts)** | **while Loop (Unlimited Attempts)** |
| --- | --- | --- |
| **Use Case** | When attempts should be limited (e.g., 6 tries max). | When the game should go on until the correct guess. |
| **Structure** | Loop control defined at the top (for (int i = 0; ...)) | Runs based on a condition (while (guess != target)) |
| **Player Control** | Less player control; forced to stop after fixed tries. | More control; continues until player succeeds. |
| **Game Duration** | Predictable and fixed. | Unpredictable; depends on user input. |
| **Best For** | Timed or competitive challenges. | Casual or educational games with no pressure. |

**Lab-07**

**To develop understanding of nested for loop phenomenon*.***

**Theory:**

Nested loop uses for loop structure within a for loop structure.

The syntax for a **nested for loop** statement in C++ is as follows −

**for ( init; condition; increment )**

**{**

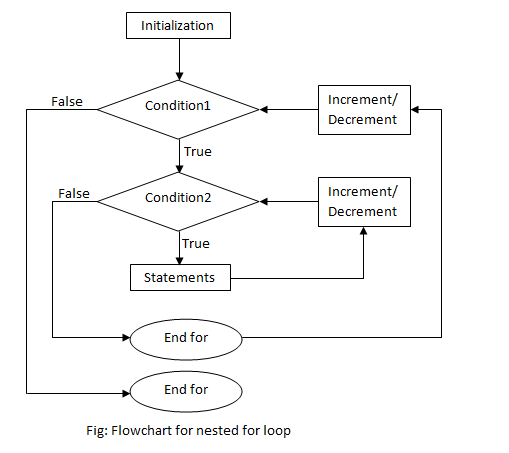
**for ( init; condition; increment )  
 {**

**statement(s);**

**}**

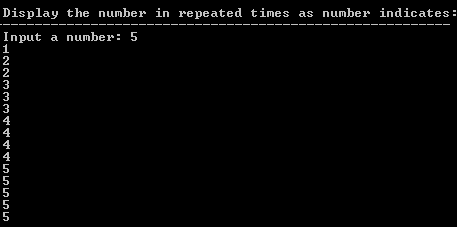
**statement(s); // you can put more statements.**

**}**



**Example 1: To Display the number in repeated times as number entered**

**Output:**



**Code:**

#include <iostream>

using namespace std;

int main() {

int num, r, sum = 0, i;

cout << "\n\n Display the number in repeated times as number indicates:\n";

cout << "---------------------------------------------------------\n";

cout << " Input a number: ";

cin >> num;

for (i = 1; i<=num; i++)

{ for (int j=1; j<=i; j++)

{ cout <<" "<< i <<endl;

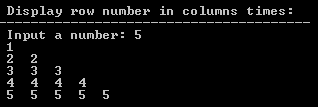
}

}

**}**

**Example 2: To print row number in columns times**

**Output:**



**Code:**

#include <iostream>

using namespace std;

int main()

{ int num, r, sum = 0, i;

cout << "\n\n Display row number in columns times:\n";

cout << "-----------------------------------------\n";

cout << " Input a number: ";

cin >> num;

for (i = 1; i<=num; i++)

{ for (int j=1; j<=i; j++)

{ cout << i <<" ";

}

cout <<endl ;

}

}

**Example 3: The following program uses a nested for loop to find the prime numbers from 2 to 100:**

**Code:**

#include <stdio.h>

int main ()

{

/\* local variable definition \*/

int i, j;

for(i=2; i<100; i++) {

for(j=2; j <= (i/j); j++)

if(!(i%j)) break; // if factor found, not prime

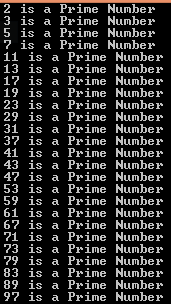
if(j > (i/j)) printf("%d is prime\n", i);

}

getch();

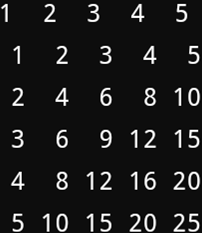
}

**Output:**



**Lab Tasks**

1.      Design a C++ program using nested loops to generate a multiplication table where the table is readable both horizontally and vertically. For example, if the table range is from 1 to 5, the output should be as follows:



#include <iostream>

#include <iomanip>

using namespace std;

int main() {

int range;

cout << "Enter the range for the multiplication table: ";

cin >> range;

cout << setw(4) << " ";

for (int i = 1; i <= range; i++) {

cout << setw(4) << i;

}

cout << endl;

// Print line

cout << " ";

for (int i = 1; i <= range; i++) {

cout << "----";

}

cout << endl;

// Print table body

for (int i = 1; i <= range; i++) {

cout << setw(2) << i << " |";

for (int j = 1; j <= range; j++) {

cout << setw(4) << i \* j;

}

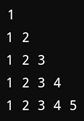
cout << endl;

}

return 0;

}

2.    Write a C++ program that takes the number of rows as input from the user and prints a triangle pyramid of numbers. The output should look like the following for a user input of 5:



#include <iostream>

using namespace std;

int main() {

int rows;

cout << "Enter number of rows: ";

cin >> rows;

for (int i = 1; i <= rows; i++) {

for (int j = 1; j <= i; j++) {

cout << j << " ";

}

cout << endl;

}

return 0;

}

3.    Create a C++ program that takes the number of rows as input from the user and displays a centered pyramid of asterisks (\*).



#include <iostream>

using namespace std;

int main() {

int rows;

cout << "Enter number of rows: ";

cin >> rows;

for (int i = 1; i <= rows; i++) {

for (int space = 1; space <= rows - i; space++) {

cout << " ";

}

for (int star = 1; star <= (2 \* i - 1); star++) {

cout << "\*";

}

cout << endl;

}

return 0;

}

**Lab-08**

**The purpose of this lab is to get familiar with the concept of numbers and arrays**

## Theory:

* **Numbers:**

Normally, when we work with Numbers, we use primitive data types such as int, short, long, float and double, etc. The number data types, their possible values and number ranges have been explained while discussing C++ Data Types.

**Defining Numbers in C++**

You have already defined numbers in various examples given in previous chapters. Here is another consolidated example to define various types of numbers in C++ −

**Code:**

#include <iostream>

using namespace std;

int main () {

// number definition:

short s;

int i;

long l;

float f;

double d;

// number assignments;

s = 10;

i = 1000;

l = 1000000;

f = 230.47;

d = 30949.374;

// number printing;

cout << "short s :" << s << endl;

cout << "int i :" << i << endl;

cout << "long l :" << l << endl;

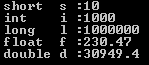
cout << "float f :" << f << endl;

cout << "double d :" << d << endl;

return 0;

**}**

**Output:**



**Math Operations in C++**

In addition to the various functions you can create, C++ also includes some useful functions you can use. These functions are available in standard C and C++ libraries and called **built-in** functions. These are functions that can be included in your program and then use.

|  |  |
| --- | --- |
| **Sr.No** | **Function & Purpose** |
| 1 | **double cos(double);**  This function takes an angle (as a double) and returns the cosine. |
| 2 | **double sin(double);**  This function takes an angle (as a double) and returns the sine. |
| 3 | **double tan(double);**  This function takes an angle (as a double) and returns the tangent. |
| 4 | **double log(double);**  This function takes a number and returns the natural log of that number. |
| 5 | **double pow(double, double);**  The first is a number you wish to raise and the second is the power you wish to raise it t |
| 6 | **double hypot(double, double);**  If you pass this function the length of two sides of a right triangle, it will return you the length of the hypotenuse. |
| 7 | **double sqrt(double);**  You pass this function a number and it gives you the square root. |
| 8 | **int abs(int);**  This function returns the absolute value of an integer that is passed to it. |
| 9 | **double fabs(double);**  This function returns the absolute value of any decimal number passed to it. |
| 10 | **double floor(double);**  Finds the integer which is less than or equal to the argument passed to it. |

C++ has a rich set of mathematical operations, which can be performed on various numbers. Following table lists down some useful built-in mathematical functions available in C++.

To utilize these functions you need to include the math header file **<cmath>**.

Following is a simple example to show few of the mathematical operations –

**Code:**

#include <iostream>

#include <cmath>

using namespace std;

int main () {

// number definition:

short s = 10;

int i = -1000;

long l = 100000;

float f = 230.47;

double d = 200.374;

// mathematical operations;

cout << "sin(d) :" << sin(d) << endl;

cout << "abs(i) :" << abs(i) << endl;

cout << "floor(d) :" << floor(d) << endl;

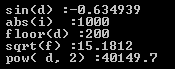
cout << "sqrt(f) :" << sqrt(f) << endl;

cout << "pow( d, 2) :" << pow(d, 2) << endl;

return 0;

}

**Output:**



**Random Numbers in C++**

There are many cases where you will wish to generate a random number. There are actually two functions you will need to know about random number generation. The first is **rand()**, this function will only return a pseudo random number. The way to fix this is to first call the **srand()** function.

Following is a simple example to generate few random numbers. This example makes use of **time()** function to get the number of seconds on your system time, to randomly seed the rand() function –

**Code:**

#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;

int main () {

int i,j;

// set the seed

srand( (unsigned)time( NULL ) );

/\* generate 10 random numbers. \*/

for( i = 0; i < 10; i++ ) {

// generate actual random number

j = rand();

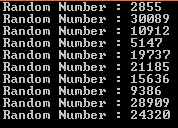
cout <<" Random Number : " << j << endl;

}

return 0;

}

**Output:**



* **Arrays**

C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

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

**Declaring Arrays**

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows −

**type arrayName [ arraySize ];**

This is called a single-dimension array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C++ data type. For example, to declare a 10-element array called balance of type double, use this statement −

**double balance[10];**

**Initializing Arrays**

You can initialize C++ array elements either one by one or using a single statement as follows −

**double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};**

The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array −

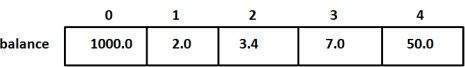
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

**double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};**

You will create exactly the same array as you did in the previous example.

**balance[4] = 50.0;**

The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th, i.e., last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representaion of the same array we discussed above −



**Accessing Array Elements**

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

**double salary = balance[9];**

The above statement will take 10th element from the array and assign the value to salary variable.

**Code:**

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

double temp[7],avg=0,sum=0;

for (int a=0;a<=6;a++)

{

cin>>temp[a];

sum = sum + temp[a];

}

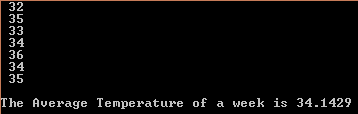
avg= sum/7;

cout<<"The Average Temperature of a week is "<<avg;

return 0;

}

**Output:**



**Lab Tasks**

* + - 1. Write a program in C++ to accept n integers from the user, store them in an array and display the sorted array in ascending order.

#include <iostream>

#include <algorithm> // for sort

using namespace std;

int main() {

int n;

cout << "Enter number of integers: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " integers:" << endl;

for (int i = 0; i < n; i++) {

cin >> arr[i];

}

sort(arr, arr + n); // sort in ascending order

cout << "Sorted array in ascending order:" << endl;

for (int i = 0; i < n; i++) {

cout << arr[i] << " ";

}

return 0;

}

2. Write a program in C++ to input n integers into an array and find the maximum value.

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter number of elements: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " integers:" << endl;

for (int i = 0; i < n; i++) {

cin >> arr[i];

}

int maxVal = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > maxVal) {

maxVal = arr[i];

}

}

cout << "Maximum value is: " << maxVal << endl;

return 0;

}

* + - 1. Write a program in C++ to input the marks of 10 students, store them in an array, and calculate the average marks.

#include <iostream>

using namespace std;

int main() {

const int SIZE = 10;

int marks[SIZE];

int sum = 0;

cout << "Enter marks for 10 students:" << endl;

for (int i = 0; i < SIZE; i++) {

cin >> marks[i];

sum += marks[i];

}

double average = static\_cast<double>(sum) / SIZE;

cout << "Average marks = " << average << endl;

return 0;

}

* + - 1. The program stores temperatures of 7 days in an array and calculates the highest and lowest temperature.

#include <iostream>

using namespace std;

int main() {

const int DAYS = 7;

float temp[DAYS];

cout << "Enter temperatures for 7 days:" << endl;

for (int i = 0; i < DAYS; i++) {

cout << "Day " << i + 1 << ": ";

cin >> temp[i];

}

float maxTemp = temp[0], minTemp = temp[0];

for (int i = 1; i < DAYS; i++) {

if (temp[i] > maxTemp) maxTemp = temp[i];

if (temp[i] < minTemp) minTemp = temp[i];

}

cout << "Highest Temperature: " << maxTemp << "°" << endl;

cout << "Lowest Temperature: " << minTemp << "°" << endl;

return 0;

}

**Lab-09**

**The purpose of this lab is to get a familiar with the concept and description multidimensional arrays**

## Theory:

Arrays are important to C++ and should need lots of more detail. There are following few important concepts, which should be clear to a C++ programmer

C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration −

**type name[size1][size2]...[sizeN];**

For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array :

**int threedim[5][10][4];**

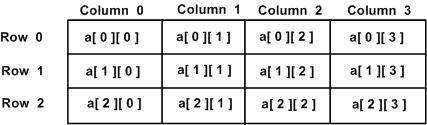
## Two-Dimensional Arrays

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows −

**type arrayName [ x ][ y ];**

Where **type** can be any valid C++ data type and **arrayName** will be a valid C++ identifier.

A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array **a**, which contains three rows and four columns can be shown as below −



Thus, every element in array a is identified by an element name of the form **a[ i ][ j ]**, where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

## Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

**int a[3][4] = {**

**{0, 1, 2, 3} , /\* initializers for row indexed by 0 \*/**

**{4, 5, 6, 7} , /\* initializers for row indexed by 1 \*/**

**{8, 9, 10, 11} /\* initializers for row indexed by 2 \*/**

**};**

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example −

**int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};**

## Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example −

**int val = a[2][3];**

The above statement will take 4th element from the 3rd row of the array. You can verify it in the above diagram.

**Code:**

**#include <iostream>**

**using namespace std;**

**int main () {**

**// an array with 5 rows and 2 columns.**

**int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};**

**// output each array element's value**

**for ( int i = 0; i < 5; i++ )**

**for ( int j = 0; j < 2; j++ ) {**

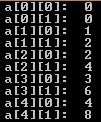
**cout << " a[" << i << "][" << j << "]: ";**

**cout << " "<<a[i][j]<< endl;**

**}**

**return 0; }**

**Output:**



**Lab Tasks**

Q1. Write a C++ program to create a **Marksheet System** using arrays that performs the following tasks:

1. Prompt the user to input the **number of students** and **number of subjects**.
2. Use a **1D array** to store the names of the students.
3. Use a **2D array** to store the marks of the students for each subject.
4. Calculate the **total marks** for each student using the 2D array.
5. Display each student's **name**, their marks for each subject, and their **total marks** in a formatted output.

#include <iostream>

#include <string>

using namespace std;

int main() {

int numStudents, numSubjects;

cout << "Enter number of students: ";

cin >> numStudents;

cout << "Enter number of subjects: ";

cin >> numSubjects;

string students[100];

int marks[100][100];

int totalMarks[100];

for (int i = 0; i < numStudents; i++) {

cout << "Enter name of student " << i + 1 << ": ";

cin >> students[i];

}

for (int i = 0; i < numStudents; i++) {

cout << "Enter marks for " << students[i] << ":\n";

totalMarks[i] = 0;

for (int j = 0; j < numSubjects; j++) {

cout << "Subject " << j + 1 << ": ";

cin >> marks[i][j];

totalMarks[i] += marks[i][j];

}

}

cout << "\n--- Marksheet ---\n";

for (int i = 0; i < numStudents; i++) {

cout << "Student: " << students[i] << "\n";

cout << "Marks: ";

for (int j = 0; j < numSubjects; j++) {

cout << marks[i][j] << " ";

}

cout << "\nTotal Marks: " << totalMarks[i] << "\n\n";

}

return 0;

}

Q2: Write a C++ program**to store the temperatures for 3 weeks (each week having 7 days). For each week, calculate and display the highest temperature of that week.**

**Instructions**:

1. Use a **2D array** to store the temperatures for 3 weeks, with each week containing 7 days of temperatures.
2. For each week, calculate the highest temperature and display it.
3. Ensure that the program displays the highest temperature for each week separately.

#include <iostream>

using namespace std;

int main() {

const int weeks = 3;

const int days = 7;

int temperature[weeks][days];

for (int i = 0; i < weeks; i++) {

cout << "Enter temperatures for week " << i + 1 << ":\n";

for (int j = 0; j < days; j++) {

cout << "Day " << j + 1 << ": ";

cin >> temperature[i][j];

}

}

for (int i = 0; i < weeks; i++) {

int highest = temperature[i][0];

for (int j = 1; j < days; j++) {

if (temperature[i][j] > highest) {

highest = temperature[i][j];

}

}

cout << "Highest temperature in Week " << i + 1 << ": " << highest << endl;

}

return 0;

}

**Lab-10**

**To be familiarized with Introduction of User-defined Function (PDF)**

#### ****Theory:****

A **function** in C++ is a self-contained block of code that performs a specific task. Functions allow us to break down a program into smaller, more manageable pieces, which improves the readability and maintainability of code. Functions can be **called** multiple times within a program, making them a powerful tool for code reusability.

### **Why Use Functions?**

1. **Code Reusability**: Once a function is written, it can be called multiple times without needing to rewrite the same code.
2. **Modularity**: Functions allow you to divide a complex task into simpler, smaller tasks, making it easier to manage and debug the program.
3. **Maintainability**: When a change needs to be made to a specific task in the program, you only need to modify the function. This makes the code easier to maintain.
4. **Abstraction**: Functions allow you to abstract away the implementation details. You only need to know what the function does, not how it does it.

**User-Defined Function:**

A **User-Defined Function** is a function that the programmer defines in order to perform a specific task. These functions allow for greater code modularity, reusability, and organization. They are used to avoid code repetition, especially when a task needs to be repeated multiple times within a program.

**Syntax of a User-Defined Function:**

return\_type function\_name(parameters) {

// Code to perform the task

}

1. **return\_type**: This specifies the data type of the value the function returns (e.g., int, float, void for no return value).
2. **function\_name**: The name by which the function will be called.
3. **parameters**: These are the values that are passed to the function to be processed. A function can have no parameters as well.
4. **Function Body**: This contains the code that gets executed when the function is called.

**Types of User-Defined Functions**:

**1. Function with no parameters and no return value**:

Used when a function needs to perform an action but doesn't require any input or produce a result.

void printMessage() {

cout << "Hello, World!" << endl;

}

**2.Function with parameters but no return value**:

Used when the function requires inputs to perform a task but does not need to return any output.

void printSum(int a, int b) {

cout << "The sum is: " << a + b << endl;

}

**3.Function with parameters and a return value**:

Used when the function needs both inputs and produces a result.

int add(int a, int b) {

return a + b;

}

1. **Function with no parameters but a return value**:

Used when no inputs are required, but the function needs to return a value.

int getNumber() {

return 42;

}

#### ****Example Code for User-Defined Functions:****

#include <iostream>

using namespace std;

// Function to calculate sum

int sum(int a, int b) {

return a + b;

}

// Function to calculate difference

int difference(int a, int b) {

return a - b;

}

// Function to check if a number is even or odd

void checkEvenOdd(int a) {

if (a % 2 == 0) {

cout << a << " is even." << endl;

} else {

cout << a << " is odd." << endl;

}

}

int main() {

int num1, num2;

// Input

cout << "Enter two numbers: ";

cin >> num1 >> num2;

// Call functions

cout << "Sum of " << num1 << " and " << num2 << " is: " << sum(num1, num2) << endl;

cout << "Difference between " << num1 << " and " << num2 << " is: " << difference(num1, num2) << endl;

checkEvenOdd(num1);

checkEvenOdd(num2);

return 0;

}

#### ****Expected Output:****

Enter two numbers: 10 5

Sum of 10 and 5 is: 15

Difference between 10 and 5 is: 5

10 is even.

5 is odd.

**LAB TASK**

**1.** Write a program that asks the user how they are feeling (happy, sad, or angry) and displays an appropriate response using a user-defined function.

#include <iostream>

#include <string>

using namespace std;

void respondToFeeling(string feeling) {

if (feeling == "happy")

cout << "That's great! Keep smiling ??" << endl;

else if (feeling == "sad")

cout << "I'm sorry to hear that. Hope things get better soon ??" << endl;

else if (feeling == "angry")

cout << "Take a deep breath. It'll pass. Try to relax ??" << endl;

else

cout << "I couldn't understand that feeling. Try again!" << endl;

}

int main() {

string feeling;

cout << "How are you feeling today? (happy, sad, or angry): ";

cin >> feeling;

respondToFeeling(feeling);

return 0;

}

* 1. Write a program that converts temperatures between Celsius and Fahrenheit based on the user’s choice, using separate user-defined functions for each conversion.

#include <iostream>

using namespace std;

float celsiusToFahrenheit(float celsius) {

return (celsius \* 9 / 5) + 32;

}

float fahrenheitToCelsius(float fahrenheit) {

return (fahrenheit - 32) \* 5 / 9;

}

int main() {

int choice;

float temp;

cout << "Temperature Converter\n";

cout << "1. Celsius to Fahrenheit\n";

cout << "2. Fahrenheit to Celsius\n";

cout << "Enter your choice (1 or 2): ";

cin >> choice;

if (choice == 1) {

cout << "Enter temperature in Celsius: ";

cin >> temp;

cout << "Temperature in Fahrenheit: " << celsiusToFahrenheit(temp) << "°F\n";

} else if (choice == 2) {

cout << "Enter temperature in Fahrenheit: ";

cin >> temp;

cout << "Temperature in Celsius: " << fahrenheitToCelsius(temp) << "°C\n";

} else {

cout << "Invalid choice!\n";

}

return 0;

}

1. Write a program to play Rock, Paper, Scissors with the computer. Use a user-defined function to determine the winner and another to randomly select the computer’s choice.

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

string getComputerChoice() {

int randNum = rand() % 3;

if (randNum == 0) return "rock";

else if (randNum == 1) return "paper";

else return "scissors";

}

string determineWinner(string user, string computer) {

if (user == computer)

return "It's a tie!";

else if ((user == "rock" && computer == "scissors") ||

(user == "paper" && computer == "rock") ||

(user == "scissors" && computer == "paper"))

return "You win!";

else

return "Computer wins!";

}

int main() {

srand(time(0)); // Seed random number generator

string userChoice, computerChoice;

cout << "Rock, Paper, Scissors Game\n";

cout << "Enter your choice (rock, paper, or scissors): ";

cin >> userChoice;

computerChoice = getComputerChoice();

cout << "Computer chose: " << computerChoice << endl;

cout << determineWinner(userChoice, computerChoice) << endl;

return 0;

}

**Pre-define Function (PDF)**

#### ****Theory:****

In C++, predefined functions are functions that are already defined in the C++ Standard Library. These functions are readily available for use and perform common tasks such as input/output, mathematical operations, and string manipulations.

Predefined functions are categorized based on the type of task they perform. Some common categories include:

* **Input/Output Functions**: Functions such as cin and cout for reading input and displaying output.
* **Mathematical Functions**: Functions like sqrt(), abs(), and pow() for performing mathematical operations.
* **String Manipulation Functions**: Functions like strlen(), strcpy(), and strcat() for working with strings.
* **Character Functions**: Functions like toupper(), tolower(), isdigit() for handling characters.

**Common Predefined Functions in C++**:

* **cin and cout**: For reading input from the user and displaying output to the console.
* **sqrt()**: Computes the square root of a number.
* **abs()**: Returns the absolute value of a number.
* **strlen()**: Returns the length of a string.
* **toupper()**: Converts a character to uppercase.
* **exit()**: Exits the program immediately.

These functions are already defined in C++ and can be used directly in your program by including the relevant header files (<cmath>, <cstring>, etc.).

#### ****Example Code for Predefined Functions:****

#include <iostream>

#include <cmath> // For sqrt(), abs()

#include <cstring> // For strlen()

using namespace std;

int main() {

int num1;

double num2;

// 1. Using cin and cout

cout << "Enter an integer: ";

cin >> num1;

// 2. Using sqrt() to calculate square root

cout << "Square root of " << num1 << " is: " << sqrt(num1) << endl;

// 3. Using abs() to find absolute value

cout << "Absolute value of " << num1 << " is: " << abs(num1) << endl;

// 4. Using strlen() to calculate the length of a string

char str[100];

cout << "Enter a string: ";

cin.ignore(); // To ignore the newline character left in the input buffer

cin.getline(str, 100);

cout << "Length of the entered string: " << strlen(str) << endl;

return 0;

}

#### ****Expected Output:****

Enter an integer: -25

Square root of -25 is: 5

Absolute value of -25 is: 25

Enter a string: Hello, World!

Length of the entered string: 13

**LAB TASK**

Write a C++ program to calculate the Body Mass Index (BMI) using the formula:



**Use only predefined functions** calculation.

#include <iostream>

#include <cmath>

using namespace std;

int main() {

float weight, height, bmi;

cout << "Enter your weight in kilograms: ";

cin >> weight;

cout << "Enter your height in meters: ";

cin >> height;

bmi = weight / pow(height, 2);

cout << "Your BMI is: " << bmi << endl;

if (bmi < 18.5)

cout << "Category: Underweight" << endl;

else if (bmi >= 18.5 && bmi < 24.9)

cout << "Category: Normal weight" << endl;

else if (bmi >= 25 && bmi < 29.9)

cout << "Category: Overweight" << endl;

else

cout << "Category: Obese" << endl;

return 0;

}

**Lab-11**

**The purpose of this lab is to get understanding about structures in C++**

## Theory:

C/C++ arrays allow you to define variables that combine several data items of the same kind, but **structure** is another user defined data type which allows you to combine data items of different kinds.

Structures are used to represent a record, suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

* Title
* Author
* Subject
* Book ID

## Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member, for your program. The format of the struct statement is this −

**struct [structure tag] {**

**member definition;**

**member definition;**

**...**

**member definition;**

**} [one or more structure variables];**

The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure −

**struct Books {**

**char title[50];**

**char author[50];**

**char subject[100];**

**int book\_id;**

**} book;**

## Accessing Structure Members

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use **struct** keyword to define variables of structure type. Following is the example to explain usage of structure −

**Code:**

#include <iostream>

#include <cstring>

using namespace std;

struct Books {

char title[50];

char author[50];

char subject[100];

int book\_id;};

int main() {

struct Books Book1; // Declare Book1 of type Book

struct Books Book2; // Declare Book2 of type Book

// book 1 specification

strcpy( Book1.title, "Learn C++ Programming");

strcpy( Book1.author, "Chand Miyan");

strcpy( Book1.subject, "C++ Programming");

Book1.book\_id = 6495407;

// book 2 specification

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Yakit Singha");

strcpy( Book2.subject, "Telecom");

Book2.book\_id = 6495700;

// Print Book1 info

cout << "Book 1 title : " << Book1.title <<endl;

cout << "Book 1 author : " << Book1.author <<endl;

cout << "Book 1 subject : " << Book1.subject <<endl;

cout << "Book 1 id : " << Book1.book\_id <<endl;

// Print Book2 info

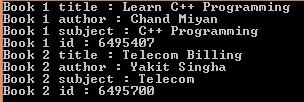
cout << "Book 2 title : " << Book2.title <<endl;

cout << "Book 2 author : " << Book2.author <<endl;

cout << "Book 2 subject : " << Book2.subject <<endl;

cout << "Book 2 id : " << Book2.book\_id <<endl;

return 0;}

**Output:**

## Structures as Function Arguments

You can pass a structure as a function argument in very similar way as you pass any other variable or pointer. You would access structure variables in the similar way as you have accessed in the above example –

**Code:**

#include <iostream>

#include <cstring>

using namespace std;

void printBook( struct Books book );

struct Books {

char title[50];

char author[50];

char subject[100];

int book\_id; };

int main() {

struct Books Book1; // Declare Book1 of type Book

struct Books Book2; // Declare Book2 of type Book

// book 1 specification

strcpy( Book1.title, "Learn C++ Programming");

strcpy( Book1.author, "Chand Miyan");

strcpy( Book1.subject, "C++ Programming");

Book1.book\_id = 6495407;

// book 2 specification

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Yakit Singha");

strcpy( Book2.subject, "Telecom");

Book2.book\_id = 6495700;

// Print Book1 info

printBook( Book1 );

// Print Book2 info

printBook( Book2 );

return 0;}

void printBook( struct Books book ) {

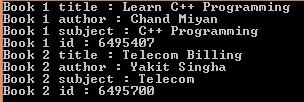
cout << "Book title : " << book.title <<endl;

cout << "Book author : " << book.author <<endl;

cout << "Book subject : " << book.subject <<endl;

cout << "Book id : " << book.book\_id <<endl;}

**Output:**



## Example: C++ Structure

C++ Program to assign data to members of a structure variable and display it.

**#include <iostream>**

**using namespace std;**

**struct Person**

**{**

char name[50];

int age;

float salary;

};

int main()

{

Person p1;

cout << "Enter Full name: ";

cin.get(p1.name, 50);

cout << "Enter age: ";

cin >> p1.age;

cout << "Enter salary: ";

cin >> p1.salary;

cout << "\nDisplaying Information." << endl;

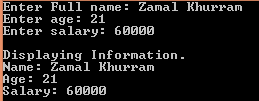
cout << "Name: " << p1.name << endl;

cout <<"Age: " << p1.age << endl;

cout << "Salary: " << p1.salary;

return 0;

}

**Output:**

**Lab Tasks**

1. **Write a C++ program to store and display information for a student using a structure. The program should:**

* Define a Student structure that contains the following fields:
  1. name: A string to store the name of the student.
  2. rollNumber: An integer to store the roll number.
  3. marks: An array of 3 floating-point numbers to store the marks of 3 subjects.
* Create an instance of the Student structure, initialize it with the following data:
  1. Name: "Alice"
  2. Roll Number: 101
  3. Marks: 85.5, 90.0, 78.5
* Output the student's name, roll number, and marks in all three subjects.

1. **Write a C++ program that demonstrates the use of nested structures. The program should:**

* Define an Address structure with the following fields:
  1. city: A string to store the name of the city.
  2. state: A string to store the name of the state.
  3. zipCode: An integer to store the zip code.
* Define an Employee structure that contains:
  1. name: A string to store the employee's name.
  2. id: An integer to store the employee's ID.
  3. address: A nested structure of type Address to store the employee’s address.
* Create an instance of the Employee structure with the following details:
  1. Name: "John Doe"
  2. ID: 101
  3. City: "New York"
  4. State: "NY"
  5. Zip Code: 10001
* Output the employee's name, ID, and full address (city, state, and zip code).

**3. Write a C++ program that stores the details of multiple people using an array of structures. The program should**:

* Define a Person structure with the following fields:
  1. first\_name: A string to store the first name of the person.
  2. last\_name: A string to store the last name of the person.
  3. age: An integer to store the age of the person.
  4. salary: A floating-point number to store the salary of the person.
* Implement a **member function** display\_info() inside the Person structure that displays the person's details.
* In the main() function:
  1. Ask the user to enter the number of people (n).
  2. Accept the details of n people (first name, last name, age, and salary).
  3. Store the details of each person in an array of structures.
  4. Display the information for all n people using the display\_info() function.

**Lab -12**

**To familiarize with the usage of the Filling in C++**

**Theory**

File handling is used for store a data permanently in computer. Using file handling we can store our data in secondary memory (Hard disk).

How to achieve the File Handling

For achieving file handling we need to follow the following steps:-

STEP 1-Naming a file

STEP 2-Opening a file

STEP 3-Writing data into the file

STEP 4-Reading data from the file

STEP 5-Closing a file.

**Streams in C++** :

We give input to the executing program and the execution program gives back the output. The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing program are called stream. In other words, streams are nothing but the flow of data in a sequence.

The input and output operation between the executing program and the devices like keyboard and monitor are known as “console I/O operation”. The input and output operation between the executing program and files are known as “disk I/O operation”.

**Classes for File stream operations** :

The I/O system of C++ contains a set of classes which define the file handling methods. These include ifstream, ofstream and fstream classes. These classes area derived from fstream and from the corresponding iostream class. These classes, designed to manage the disk files, are declared in fstream and therefore we must include this file in any program that uses files.

In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream available in fstream headerfile.

ofstream: Stream class to write on files

ifstream: Stream class to read from files

fstream: Stream class to both read and write from/to files.

The default mode for opening a file with ofstream's constructor is to create it if it does not exist, or delete everything in it if something does exist in it. If necessary, you can give a second argument that specifies how the file should be handled. They are listed below:  
ios::app -- Opens the file, and allows additions at the end  
ios::ate -- Opens the file, but allows additions anywhere  
ios::trunc -- Deletes everything in the file  
ios::nocreate -- Does not open if the file must be created  
ios::noreplace -- Does not open if the file already exists

**Example: To Create a file and write until “!” is encountered and display the result**

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

ofstream fout; // Creation of ofstream class object

string line;

// by default ios::out mode, automatically deletes

// the content of file. To append the content, open in ios:app

// fout.open("sample.txt", ios::app)

fout.open("sample.txt");

while (fout) // Execute a loop If file successfully opened

{ getline(cin, line); // Read a Line from standard input

if (line == "!") // Press ! to exit

break;

fout << line << endl; // Write line in file

}

fout.close(); // Close the File

ifstream fin; // Creation of ifstream class object to read the file

fin.open("sample.txt"); // by default open mode = ios::in mode

while (fin) { // Execute a loop until EOF (End of File)

getline(fin, line); // Read a Line from File

cout << line << endl; // Print line in Console

}

fin.close(); // Close the file

return 0;

}

**Lab Task**

Q1: Create a C++ program that performs the following operations:

1. Creates a file library.txt and writes the following data:
   * "Book Title: C++ Programming"
   * "Author: ABC"
   * "Year of Publication: 1985"
2. Reads and displays the content of library.txt.
3. Appends the following data to library.txt:
   * "Publisher: Addison-Wesley"
   * "ISBN: 123-0321563842"
4. Reads and displays the updated content of library.txt.

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

string filename = "library.txt";

ofstream outFile(filename); // Open for writing (overwrite mode)

if (outFile.is\_open()) {

outFile << "Book Title: C++ Programming\n";

outFile << "Author: ABC\n";

outFile << "Year of Publication: 1985\n";

outFile.close();

cout << "Initial data written to '" << filename << "'.\n\n";

} else {

cerr << "Error: Unable to open file for writing.\n";

return 1;

}

ifstream inFile(filename);

if (inFile.is\_open()) {

string line;

cout << "Reading current contents of '" << filename << "':\n";

while (getline(inFile, line)) {

cout << line << endl;

}

inFile.close();

cout << endl;

} else {

cerr << "Error: Unable to open file for reading.\n";

return 1;

}

ofstream appendFile(filename, ios::app); // Open in a

**Lab -13**

**PROBLEM BASED LEARNING**

**ATM System Design Challenge**

You are the developer tasked with building an **ATM system** that simulates user transactions, but the system must handle various scenarios that can occur in a real-world environment. Your goal is to design a system that performs multiple banking operations while handling edge cases, input validation, and user interactions.

### **The Problem: Imagine a customer approaches an ATM to access their bank account. They need to be able to:**

* **Check their account balance.**
* **Deposit money into their account.**
* **Withdraw money without exceeding the available balance.**
* **Transfer funds to another account.**
* **View a history of transactions they have made.**

However, there are several challenges that your ATM system needs to overcome:

### **Key Challenges to Solve:**

* **Balance Management**: How can you ensure that the user cannot withdraw more than their current balance?
* **Transaction Tracking**: How will you store and display transaction history effectively?
* **Input and Output Handling**: How do you ensure the system handles incorrect inputs like negative deposits or invalid menu selections?
* **User Experience**: How will you design the ATM menu flow so that it is intuitive and user-friendly while ensuring robust error handling?



**FACULTY OF ENGINEERING, SCIENCES AND TECHNOLOGY**

|  |  |
| --- | --- |
| **(Open-Ended Lab )** | |
| **Blooms Taxonomy** | **GAs** |
| C3  A3 | GA-4  GA-6 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Marks Allocated:** | 10 Marks | **Time Allocated:** | 1 Week |

**1. Problem Statement**

**2. Objective**

**3. Features and Functionalities**

**4. Code and output SC**