**Logo, company name

Description automatically generated**

**Department of (Computer Science)**

**Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology, Haripur, Pakistan**

**COMP-112L Object Oriented Programming Lab**

**Lab Journal**

**Class: BS Computer Science**

**Name: Ahmed Raza**

**Registration No.: B20F0436CS031**

**Semester: 3rd**

**Submission Date: 08 February, 2022**

**Submitted to: Engr. Rafi-Ullah**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Instructor Signature**

**Lab No. 04**

**Structures & Object Orientation in C++**

**Objectives:**

In this lab we will be discussing about Structures in detail. This is one of the most important concepts in C++ language A struct is a derived data type that consists of members that are each fundamental or derived data types. struct is used to declare a new data-type. Basically this means grouping variables together.

In C++ it is required to do some meaningful operations on structures they are:

* Structure Declaration
* Structure Definition
* Structure Variables
* Structure Membership
* Arrays of structures

**Tools/Software Required:**

* All the tasks are implemented on DEV C++.

**Introduction:**

**Arrays of Character**

**STRUCTURES**

Arrays require that all elements be of the same data type. Many times it is necessary to group information of different data types. An example is a materials list for a product. The list typically includes a name for each item, a part number, dimensions, weight, and cost. C and C++ support data structures that can store combinations of character, integer floating point and enumerated type data. They are called a **STRUCTS**.

**Streaming an Array of Characters**

Like any other variable, before using a string, you must first declare it, which is done by type the char keyword, followed by the name of the variable, followed by square brackets. When declaring the variable, if/since you do not know the number of characters needed for the string; you must still provide an estimate number. You can provide a value large enough to accommodate the maximum number of characters that would be necessary for the variable. For a person's name, this could be 20. For the title of a book or a web page, this could be longer.

**Structures Definition**

Before a structure is created, it is necessary to define its overall composition. The format of the a structure is provided in the shape of a template or pattern which is then used to creatstructure variables of the same composition. The template is composed of the names and attributes of the data items to be included in the structure. The definition begins with the keyword **struct** which is followed by a structure declaration consist of a set of user-defined data names and data types. These entries are separated by semicolons and enclosed within a pair of curly brackets. The definition ends with a semicolon. Thus, in general, the structure definition has the form

**struct tag-name**

**{**

**type var-1;**

**type-var-2; . . . . . . . . typevar-n;**

**}**;

Where **tag-name** is the user-supplied name that identified the structure template; **type** refers to any valid data type such as char, int, float, and so forth; and var-1, var-2, ….var-n are user-defined variables Page 42 . . . names, arrays or pointers. The components of a structure are commonly referred to as **members or** **field**.

**Object Orientation in C++**

**Data Types, Objects, Classes and Instances**

So far, we've learnt that C++ lets you create variables which can be any of a range of basic data types: int, long, double and so on. However, the variables of the basic types don't allow you to model realworld objects (or even imaginary objects) adequately. It's hard to model a box in terms of an int, for example; what we need is something that allows us to collect together the various attributes of an object. In C++, we can do this very easily using classes.

You could define variables, length, breadth and height to represent the dimensions of the box and bind them together as members of a Box class, as follows:

**class Box**

**{ public: double length;**

**double breadth;**

**double height;**

**};**

This code actually creates a new type, called Box. The keyword class defines Box as such, and the elements that make up this class are defined within the curly braces. Note that each line defining an element of the class is terminated by a semicolon, and that a semicolon also appears after the closing brace. The elements length, breadth and height are referred to as data members. At the top of the class definition, you can see we have put the keyword public - this just means that the data members are generally accessible from outside the class. You may, however, place restrictions on the accessibility of class members, and we'll see how this works a bit later in this chapter.

With this definition of a new data type called Box, you can go ahead and define variables of this type just as you did with variables of the basic types:

**Box myBox; //Declare a variable myBox**

Once we've defined the class Box, the declaration of variables of this type is quite standard. The variable myBox here is also referred to as an **object** or an **instance** of the class Box.

With this definition of a new data type called Box, you can go ahead and define variables of this type just as you did with variables of the basic types. You can then create, manipulate and destroy as many Box objects as you need to in your program. This means that you can model objects using classes and write your programs around them. So - that's object-oriented programming all wrapped up then?

Well, not quite. You see, object-oriented programming (OOP) is based on a number of foundations (famously encapsulation, polymorphism and inheritance) and what we have seen so far doesn't quite fit the bill. Don't worry about what these terms mean for the moment - we'll be exploring these ideas in more detail as we learn more about what you can do with classes.

**First Class**

The notion of class was invented by an Englishman to keep the general population happy. It derives from the theory that people who knew their place and function in society would be much more secure and comfortable in life than those who did not. The famous Dane, Bjarne Stroustrup, who invented C++, undoubtedly acquired a deep knowledge of class concepts while at Cambridge University in England, and appropriated the idea very successfully for use in his new language. The idea of a class in C++ is similar to the English concept, in that each class usually has a very precise role and a permitted set of actions. However, it differs from the English idea, because class in C++ has largely socialist overtones, concentrating on the importance of working classes. Indeed, in some ways it is the reverse of the English ideal, because, as we shall see, working classes in C++ often live on the backs of classes that do nothing at all.

**Operations on Classes**

In C++ you can create new data types as classes to represent whatever kinds of objects you like. As you'll come to see, classes aren't limited to just holding data; you can also define member functions that act on your objects, or even operations that act between objects of your classes using the standard C++ operators. You can define the class Box, for example, so that the following statements work and have the meanings you want them to have:

**Box Box1;**

**Box Box2;**

**If**

**(Box1 > Box2) // Fill the larger box**

**Box1.Fill();**

**else**

**Box2.Fill();**

You could also implement operations as part of the Box class for adding, subtracting or even multiplying boxes - in fact, almost any operation to which you could ascribe a sensible meaning in the context of boxes.

**Lab Tasks:**

**Task 1:**

Write a C++ program to create class name ‘student’ to get and print details of a student. Declare two functions in class name getDetail() and printDetail(). In getDetail() take detail of a student(Name, Roll Number, Total marks out of 500,) and calculate percentage. In printDetail() display the all data of a student. Output should be like this. Note: Name should be in two words e.g. (Shahid Khan).

**Code:**

**#include<iostream>**

**#include<string>**

**using namespace std;**

**class Student**

**{**

**public :**

**string name;**

**int Roll\_No, Total\_marks;**

**float percent;**

**get\_detail()**

**{**

**cout<<"Enter the name of the student : ";**

**cin>>name;**

**cout<<"Enter the Roll\_No of the student : ";**

**cin>>Roll\_No;**

**cout<<"Enter the Total marks of the student(OUT OF 500) : ";**

**cin>>Total\_marks;**

**}**

**print\_detail()**

**{**

**cout<<"Name of the student is : "<<name<<endl;**

**cout<<"Roll\_No of the student is : "<<Roll\_No<<endl;**

**cout<<"Total marks of the student(OUT OF 500) is : "<<Total\_marks<<endl;**

**percent=Total\_marks;**

**cout<<"Percentage is "<<(percent/500)\*100<<endl;**

**}**

**};**

**int main()**

**{**

**Student stud;**

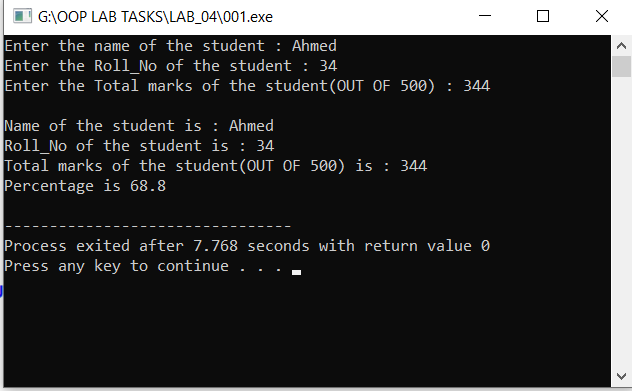
**stud.get\_detail();**

**stud.print\_detail();**

**return 0;**

**}**

**Output:**

****

**Task # 02:**

Write a C++ program to create a class ‘Distance’ to read and add two distance. Make 4 functions 1) getDist() →→ Take two distance in feet and inches form. 2) addDist() →→ Add two distance. 3) subDist() →→Subtract two distance. 4) showDist() →→ Display original distances. In main show addition and subtraction of distances. Note: If we enter inches >=12 then feet++ and inches-=12

**Code:**

**#include <iostream>**

**using namespace std;**

**class Distance**

**{**

**private:**

**int feet;**

**int inch;**

**public:**

**Distance ()**

**{**

**feet = 0; inch = 0;**

**}**

**void getDist()**

**{**

**cout << "Enter Value of feets : ";**

**cin >> feet;**

**cout << "Enter value of inches : ";**

**cin >> inch;**

**inch = (inch >= 12) ? 12 : inch;**

**}**

**void showDist()**

**{**

**cout << endl << "\tFeets : " << feet;**

**cout << endl << "\tInches: " << inch;**

**}**

**Distance addDist( Distance d2 )**

**{**

**Distance temp;**

**temp.feet = feet + d2.feet;**

**temp.inch = inch + d2.inch;**

**if( temp.inch >= 12)**

**{**

**temp.feet++;**

**temp.inch -= 12;**

**}**

**return temp;**

**}**

**Distance subDist( Distance d2 )**

**{**

**Distance temp;**

**temp.feet = feet - d2.feet;**

**temp.inch = inch - d2.inch;**

**if( temp.inch < 0 )**

**{**

**temp.feet--;**

**temp.inch = 12 + temp.inch;**

**}**

**return temp;**

**}**

**};**

**int main()**

**{**

**Distance d1;**

**Distance d2;**

**Distance d3;**

**Distance d4;**

**cout << "Enter Distance1 : " << endl;**

**d1.getDist();**

**cout << "Enter Distance2 : " << endl;**

**d2.getDist();**

**d3 = d1.addDist(d2);**

**d4 = d1.subDist(d2);**

**cout << endl << "Distance1 : " ;**

**d1.showDist();**

**cout << endl << "Distance2 : " ;**

**d2.showDist();**

**cout << endl << "Addition of the distance : " ;**

**d3.showDist();**

**cout << endl << "Subtraction of the distance : " ;**

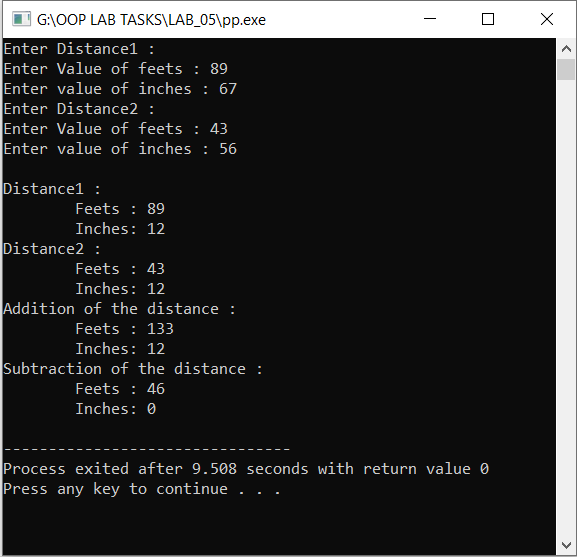
**d4.showDist();**

**cout << endl;**

**return 0;**

**}**

**Output:**

****

**Results & Observations:**

In this Lab I’ve learned about the concept of Structures like: Structure Declaration, Structure Definition, Structure Variables, Structure Membership, Arrays of structures. And also, I understand the concept of classes and its objects. In the first task, I’ve used a Class of student in which I’ve used two functions like getdetail() for taking data from the user of the student and printdetail() for printing it and in the second task I’ve used constructers that’s having the same name as the Class then I’ve used 4 function like getdist() for getting data , showdist() for printing output, adddist() for adding the two distances and subdist() for subtracting two distance.