



Lab Manual

**OBJECT ORIENTED PROGRAMMING**

**Semester : Fall 2023**

**Program : BS**

**Course Title and Name : CSC 213**

**Credits : 1**

**Faculty : Mehar Khan Niazi**

**Student Name :**

**Student ID :**

**Total Marks : 100**

**Obtained Marks :**

**Submitted Date :**

**Week 2B**

**Lab: Introduction to OOP,**

**Procedural Programming vs Object Oriented Programming**

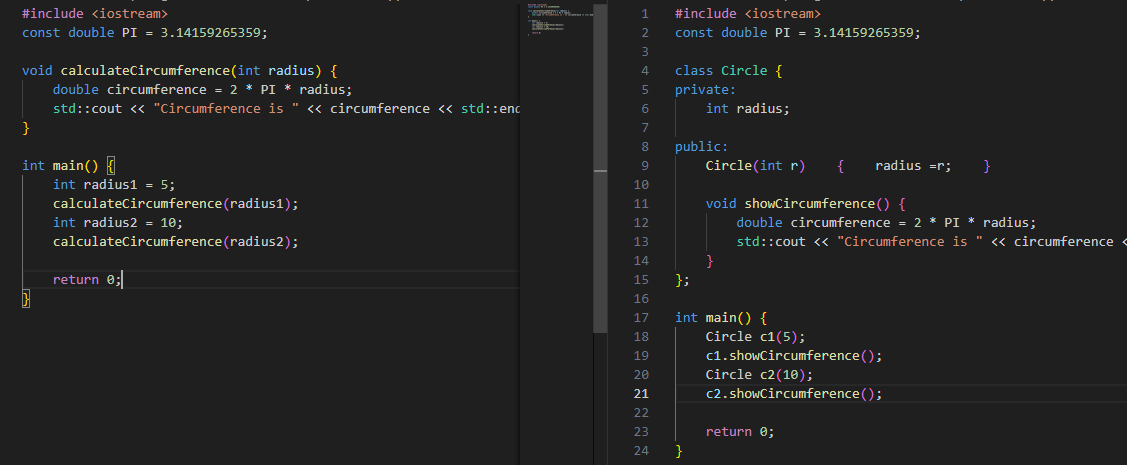
**Introduction**

Procedural programming uses a list of instructions to tell the computer what to do step by-step. Procedural programming relies on procedures, also known as routines or subroutines. A procedure contains a series of computational steps to be carried out. Procedural programming is intuitive in the sense that it is very similar to how you would expect a program to work. If you want a computer to do something, you should provide step-by-step instructions on how to do it. It is, therefore, no surprise that most of the early programming languages are all procedural. Examples of procedural languages include Fortran, COBOL and C, which have been around since the 1960s and 70s.

Object-oriented programming, or OOP, is an approach to problem-solving where all computations are carried out using objects. An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You would also expect a person to be able to do something, such as walking. This would be considered a method of the person. A method in object-oriented programming is like a procedure in procedural programming. The key difference here is that the method is part of an object. In object-oriented programming, you organize your code by creating objects, and then you can give those objects properties and you can make them do certain things.

One of the most important characteristics of procedural programming is that it relies on procedures that operate on data - these are two separate concepts. In object-oriented programming, these two concepts are bundled into objects. This makes it possible to create more complicated behavior with less code. The use of objects also makes it possible to reuse code. Once you have created an object with more complex behavior, you can use it anywhere in your code.

**Procedural programming vs Object oriented programming**

****

**Comparison:**

**Procedural Approach:**

* Functions are used to perform calculations.
* Data is passed as parameters to functions.
* Code is more procedural and less organized.

**Object-Oriented Approach:**

* Data (radius) and methods (showCircumference) are encapsulated within a class.
* Objects (c1 and c2) are created to represent individual circles.
* Code is more organized and follows the principles of encapsulation and abstraction.

**Lab Exercise 1: Basic Syntax and Structure of a Class in C++**

**Objective:**

* Understand the basic structure of a class in C++.
* Create a simple class with attributes and member functions.
* Implement class constructors and member functions.
* Demonstrate object initialization and method invocation.

**Basic Syntax and Structure of a Class:**

A class is a fundamental concept used for organizing and defining the blueprint for objects. It encapsulates data members and functions that operate on those members. The basic structure of a class in C++ typically consists of the following elements:

**1. Class Declaration:** You begin by declaring the class using the `class` keyword followed by the class name. Here's a simple example of a class declaration:

**class MyClass {**

**// Class members and functions go here**

**};**

**2. Data Members:** Data members are variables that represent the attributes or properties of objects created from the class. These variables are defined within the class and specify the characteristics of the objects. Data members can have different access specifiers:

* **Public:** These members are accessible from outside the class.
* **Private:** These members are only accessible from within the class.

**class MyClass {**

**public:**

**int publicVar; // Public data member**

**private:**

**double privateVar; // Private data member**

**};**

**3. Member Functions:** Member functions are functions defined within the class, and they operate on the data members of the class. Like data members, member functions can also have different access specifiers.

* **Public Functions:** These functions can be called from outside the class.
* **Private Functions:** These functions are only callable from within the class.

**class MyClass {**

**public:**

**int publicVar; // Public data member**

**void publicFunction() {**

**// Public member function**

**}**

**private:**

**double privateVar; // Private data member**

**void privateFunction() {**

**// Private member function**

**}**

**};**

**4. Constructor:** A constructor is a special member function that is called when an object of the class is created. It is used to initialize the data members of the object. Constructors have the same name as the class and no return type.

**class MyClass {**

**public:**

**MyClass() {**

**// Constructor code**

**}**

**};**

**5. Destructor:** A destructor is also a special member function, with the same name as the class but preceded by a tilde (`~`). It is called when an object is destroyed (e.g., when it goes out of scope). It is used to release any resources acquired by the object.

**class MyClass {**

**public:**

**MyClass() {**

**// Constructor code**

**}**

**~MyClass() {**

**// Destructor code**

**}**

**};**

**6.** **Member Access Specifiers:** As mentioned earlier, C++ provides access specifiers to control the visibility of class members. These access specifiers are `public`, `private`, and `protected`. They dictate whether a member can be accessed from outside the class or only from within.

**7. Member Variables:** These are variables used to store the state or attributes of objects created from the class.

**8. Member Functions:** These are functions defined within the class that operate on the class's data members.

Once you have defined a class, you can create objects (instances) of that class, and each object will have its own set of data members and can invoke the class's member functions. Here's an example of creating an object of the `MyClass` class:

**MyClass obj; // Creating an object of MyClass**

**obj.publicVar = 42; // Accessing and modifying public data member**

**obj.publicFunction(); // Calling a public member function**

This basic structure forms the foundation of object-oriented programming in C++, allowing you to model and manipulate data using classes and objects.

**Tools/Software Requirement:**

* Dev-C++

**LAB EXPERIMENT # 1**

**Example 1: Basic Syntax and Structure of a Class in C++**

#include <iostream>

using namespace std;

class Student {

public:

// Attributes

string name;

int age; };

int main() {

// Create an object of the 'Student' class

Student student1;

student1.name = "Azhar";

student1.age = 20;

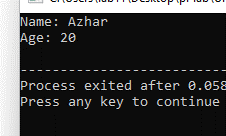
// Display student information

cout << "Name: " << student1.name << endl;

cout << "Age: " << student1.age << endl;

return 0; }

**Output & Explanation/Reason:**



**Example 2: Basic Syntax and Structure of a Class in C++**

#include <iostream>

using namespace std;

class Student {

public:

// Attributes

string name;

int age;

// Constructor

Student(string n, int a) {

name = n;

age = a; }

// Member function to display student information

void displayInfo() {

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

cout << "Age: " << age << endl; } };

int main() {

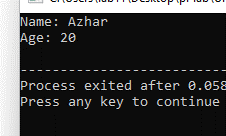
// Create an object of the 'Student' class

Student student1("Azhar", 21);

// Call the member function to display student information

student1.displayInfo(); return 0; }

**Output & Explanation/Reason:**



**Example 3:**

#include <iostream>

#include <string>

using namespace std;

class Student {

public:

// Attributes

string name;

int age;

double grade;

// Constructor

Student(string n, int a, double g) {

name = n;

age = a;

grade = g;

}

// Member function to display student information

void displayInfo() {

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

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

cout << "Grade: " << grade << endl; }

// Member function to check if the student passed

bool isPassing() {

return grade >= 50; }

// Member function to congratulate the student

void congratulate() {

if (isPassing()) {

cout << "Congratulations, " << name << "! You passed!" << endl;

} else {

cout << "Sorry, " << name << ". You need to work harder." << endl;

}

}

};

int main() {

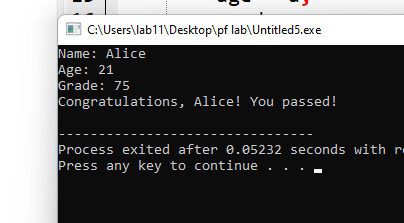
Student student1("Alice", 21, 75);

student1.displayInfo();

student1.congratulate();

return 0; }

**Output:**



**LAB EXPERIMENT # 2 Access Control and Encapsulation**

**Objective:**

* Understand access control in C++.
* Implement encapsulation by using private access specifiers.
* Use getter and setter methods to access private attributes.

**Example 1: Access Control and Encapsulation**

class Student {

private:

string name;

int age;

public:

// Constructor

Student(string n, int a) {

name = n;

age = a;

}

// Getter methods

string getName() {

return name;

}

int getAge() {

return age;

}

// Setter methods

void setName(string n) {

name = n;

}

void setAge(int a) {

age = a;

}

// Member function to display student information

void displayInfo() {

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

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

}

};

**Output:**



**Lab Assignments: Requirements (Code, Output and Reason)**

**Task 1:**

Create a class 'Student’, than create five student objects containing Student Name , Age , Gender (without member function) and display student details (without member) function)

#include<iostream>

using namespace std;

class Student {

public:

string name,gender;

int age;

};

int main(){

Student student1,student2,student3,student4,student5 ;

student1.name= "sonu";

student1.age= 20;

student1.gender="male";

student2.name="pakistan";

student2.age= 77;

student2.gender="male";

student3.name= "india";

student3.age= 77;

student3.gender="Female";

student4.name= "china";

student4.age= 100;

student4.gender="male";

student5.name= "sindh";

student5.age= 20000;

student5.gender="MALE";

cout<<"name:"<<student1.name<<endl;

cout<<"age :"<<student1.age<<endl;

cout<<"gender :"<<student1.gender<<endl;

cout<<"name:"<<student2.name<<endl;

cout<<"age :"<<student2.age<<endl;

cout<<"gender :"<<student2.gender<<endl;

cout<<"name:"<<student3.name<<endl;

cout<<"age :"<<student3.age<<endl;

cout<<"gender :"<<student3.gender<<endl;

cout<<"name :"<<student4.name<<endl;

cout<<"age :"<<student4.age<<endl;

cout<<"gender :"<<student4.gender<<endl;

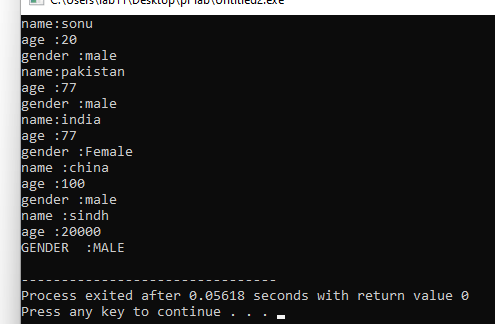
cout<<"name :"<<student5.name<<endl;

cout<<"age :"<<student5.age<<endl;

cout<<"GENDER :"<<student5.gender<<endl;

};

OUTPUT:



**Task 2:**

Create a class 'Student’, Create six student objects containing Student Name , Age , ID (with the help of member function) and display student details (with the help of member function)

#include<iostream>

using namespace std;

class Student {

public:

string name;

int age;

int id;

};

int main(){

Student student1,student2,student3,student4,student5 ;

student1.name= "sonu";

student1.age= 20;

student1.id=29990;

student2.name="pakistan";

student2.age= 77;

student2.id=29890;

student3.name= "india";

student3.age= 77;

student3.id=59990;

student4.name= "china";

student4.age= 100;

student4.id=89990;

student5.name= "sindh";

student5.age= 20000;

student5.id=280990;

cout<<"name:"<<student1.name<<endl;

cout<<"age :"<<student1.age<<endl;

cout<<"Id :"<<student1.id<<endl;

cout<<"name:"<<student2.name<<endl;

cout<<"age :"<<student2.age<<endl;

cout<<"Id :"<<student2.id<<endl;

cout<<"name:"<<student3.name<<endl;

cout<<"age :"<<student3.age<<endl;

cout<<"Id :"<<student3.id<<endl;

cout<<"name :"<<student4.name<<endl;

cout<<"age :"<<student4.age<<endl;

cout<<"Id :"<<student4.id<<endl;

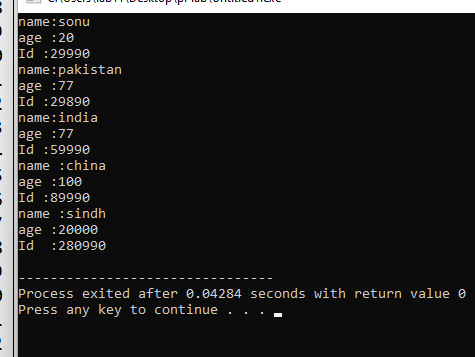
cout<<"name :"<<student5.name<<endl;

cout<<"age :"<<student5.age<<endl;

cout<<"Id :"<<student5.id<<endl;

};

Output



**Task 3:**

Create a 'BankAccount'

Implement the 'displayInfo()' function to display account information.

Implement the 'deposit()' function to deposit money into the account (with validation like deposit amount greater than zero).

Implement the 'withdraw()' function to withdraw money from the account (with validation like ammount is greater than zero and less than balance).

In the 'main()' function, create an object of your 'BankAccount' class, set its attributes, and call the 'displayInfo()', 'deposit()', and 'withdraw()' functions to perform transactions.

**Task 4:**

Create 'Student' class , make 'name' , ‘id’ and 'department' private attributes.

Implement getter and setter methods.

In the 'main()' function, create five object of your 'Student' class, set its attributes using setter methods, display one student details using getter and others ones using the 'displayInfo()' method.