Lab 1

Instructions for Lab Submission:

1. File Naming and Submission:
    a. Submit your solution as a zip file named **Lab1_<ERP>.zip**, where *<ERP>* is your ERP number. For example, if your ERP number is *12345*, the file should be named *Lab1_12345.zip*.
    b. Ensure that the zip file:
        i. Contains the following directories:
            1. ***headers/***: All header files, named as ***<class/struct name>.hpp***
            2. ***src/***: All implementation files named as ***<class/struct name>.cpp***
            3. ***main/***: All files with main function for each task named as ***Task<number>.cpp***
        ii. Does not include unnecessary files like executables.
    c. Do not submit ***TEXT*** files.
2. Code Organization:
    a. Each class or struct used in a task must have its declaration in a *.hpp* file (in the ***headers/*** directory) and its implementation in a *.cpp* file (in the ***src/*** directory). The logic and function calls should be written in a separate main file (in the ***main/*** directory).
    b. Ensure that the code structure is modular and adheres to the principles of good software development (e.g., separation of concerns).
3. Code Neatness:
    a. Ensure your code is well-formatted and easy to read. Use proper indentation and meaningful variable names.
    b. Include appropriate comments to explain the logic where necessary (especially for functions and complex sections of the code).
    c. **Add clear explanations where required(or specified)** to make the logic of your program easy to follow.
4. Code Compilation and Functionality:
    a. Ensure your code compiles without errors or warnings. You should be able to compile and run your code on any system with a standard C++ compiler (e.g., GCC, Clang, MSVC).
    b. Double-check that the functionality of the program meets the requirements outlined in the task.
5. Additional Instructions:
    a. Do not use any libraries or features that have not been covered in the course material, unless specifically instructed.
    b. Make sure your code adheres to the principles of good software development (e.g., modular functions, minimal repetition, and no hardcoding).
    c. If you encounter any issues or require clarifications, reach out before the submission deadline.
6. Deadline:
    a. Ensure that you submit your lab on time, before the deadline. Late submissions will not be accepted
    b. Double-check that the zip file is correctly named and contains all necessary files before submitting.
7. Plagiarism Policy:
    a. The work submitted must be entirely your own. Any code or content copied from others (including online sources, classmates or AI) will result in an automatic zero for the task.

Object-Oriented Programming Basics in C++

Access Modifiers: Public, Private, and Protected

Access modifiers define how members (variables and functions) of a class can be accessed. They are fundamental to encapsulation, one of the key principles of Object-Oriented Programming.

Public

- Members declared as public can be accessed from anywhere in the program.
- Typically used for functions or variables that are intended to be accessed by objects of the class.

```cpp
class Example {
public:
    int x;  // Public member
};

int main() {
    Example obj;
    obj.x = 10;  // Accessible
    return 0;
}
```

Private

- Members declared as private can only be accessed within the class.
- These are typically used to protect sensitive data or to enforce abstraction.

```cpp
class Example {
private:
    int x;  // Private member

public:
    void setX(int value) { x = value; }  // Public method to modify private member
    int getX() { return x; }             // Public method to access private member
};

int main() {
    Example obj;
    obj.setX(10);  // Indirectly accessing private member
    return 0;
}
```

Protected

- Members declared as protected can be accessed within the class and by derived (child) classes.
- These are primarily used in inheritance scenarios.

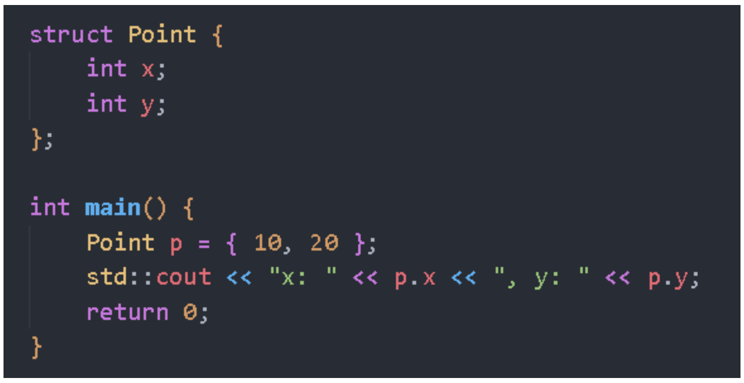
```cpp
class Parent {
protected:
    int x;  // Protected member
};

class Child : public Parent {
public:
    void setX(int value) { x = value; }  // Accessing protected member from base class
};
```

Classes and Structs

C++ supports both class and struct for defining custom data types. While they are similar, there are key differences:

Structs:

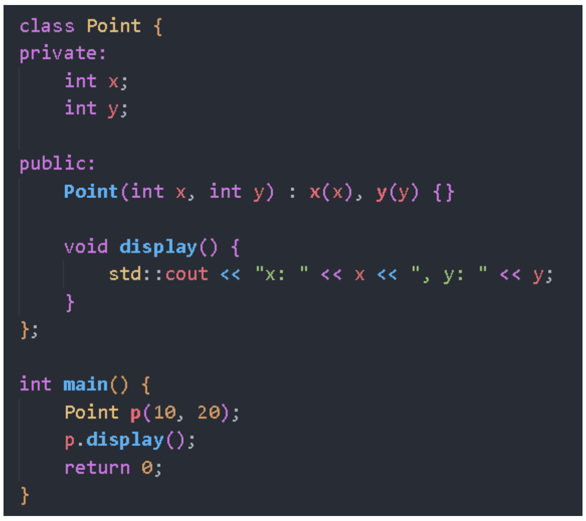
- Members of a struct are public by default.
- Typically used for grouping data in a simple way, without requiring strict encapsulation.



```
struct Point {
    int x;
    int y;
};

int main() {
    Point p = { 10, 20 };
    std::cout << "x: " << p.x << ", y: " << p.y;
    return 0;
}
```

Classes:

- Members of a class are private by default.
- Provides full support for encapsulation and abstraction.



```
class Point {
private:
    int x;
    int y;

public:
    Point(int x, int y) : x(x), y(y) {}

    void display() {
        std::cout << "x: " << x << ", y: " << y;
    }
};

int main() {
    Point p(10, 20);
    p.display();
    return 0;
}
```

Lab Tasks

Task 1:

Design and implement a class to store student data.

1. Define the necessary attributes for each student (id, name, CGPA). Define a public field called courses that stores each course name the student is enrolled in.
2. Define a custom default constructor that takes inputs from the user and set the attributes accordingly.
3. Define a parameterized constructor as well, that takes arguments to set the attributes of each student.
4. Display the 'courses' of a student by accessing the attribute directly in the main function. Then figure out how to display the other private attributes.

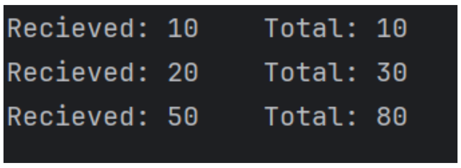
Note: You should maintain and submit a separate header file (.h) and a source file (.cpp) for this question.

Task 2:

Create a class to track your pocket money. It should have attributes like source, amount received etc.

Keep track of the total amount in your wallet by using a static variable. It should increase each time a new object is created (more money is received).

Display the updated money each time in the format given below.



```
Recieved: 10    Total: 10
Recieved: 20    Total: 30
Recieved: 50    Total: 80
```

Task 3(Mandatory):

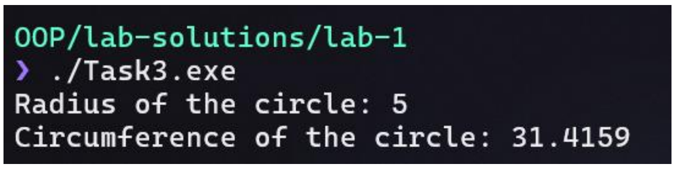
In a library, books need to be tracked with details such as title, author, and availability. Members of the library can borrow and return books, but they are limited to borrowing a maximum of three books at a time. The system should ensure that only available books can be borrowed and provide a structured way to track which books a member has taken. When a book is returned, it becomes available for other members. A librarian is responsible for managing the books by adding and removing them. When a member requests a book, the system checks if it is available. If available, the book is marked as borrowed, and it is added to the member's borrowing list. If a member wants to return a book, the system ensures that the book is returned in the same order it was borrowed, following a First-In-First-Out (FIFO) policy. If a book is not available, the system displays a message indicating that it is currently borrowed by another member.

The client program (main.cpp) acts as the interface through which users interact with the system. It initializes the library with some books and members. Librarians can then add or remove books as needed, and members can borrow or return books. The system provides a display function to show the current status of all books, indicating whether they are available or borrowed.

Task 4:

Create a class **Circle** with:

- A private member **radius** *(float)*.
- Public methods:
  - **setRadius(float r)** to set the value of radius.
  - **getRadius()** to return the value of radius.
  - **circumference()** to calculate and return the circumference.



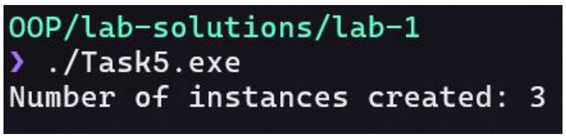
```
OOP/lab-solutions/lab-1
❯ ./Task3.exe
Radius of the circle: 5
Circumference of the circle: 31.4159
```

In main, create a **Circle** object, set the radius using **setRadius()**, and display its circumference.

Task 5:

Create a class **InstanceCounter** with:

1. A **private static** member variable count that tracks the number of instances of the class.
2. A **constructor** that increments count whenever a new object is created.
3. A **public static** method **getCount()** that returns the current value of count.



```
OOP/lab-solutions/lab-1
❯ ./Task5.exe
Number of instances created: 3
```

In the main function, create multiple objects of the class, and use **getCount()** method to print how many objects have been created.