Object-Oriented Programming (CS112)
Lab Task # 01
[CLO-2, Taxonomy Level-C3, GA3]

Instructions

Plagiarism, copy & paste material will lead to the cancellation of your assignment. The input values should be different, don't copy the solution from external source as the plagiarism will be tested through Turnitin and the plagiarized assignments will lead to marks' deduction.

Objectives:

✅ Classes and Objects
✅ Mutator (Setter) Functions
✅ Function Calling

Lab Activity

```
#include <iostream>

using namespace std;


// Define the class

class Book {

public:

   // Function to display the title

   void displayTitle() {

      cout << "The title of the book is 'C++ Programming Basics'" << endl;

   }

};

int main() {

   // Create an object of the class

   Book myBook;


   // Call the function using the object

   myBook.displayTitle();


   return 0;

}
```

Setter (Mutator) and Getter functions are essential concepts in Object-Oriented Programming (OOP), primarily used for data encapsulation and controlling access to an object's attributes (fields). Here's a breakdown of their uses:

1. Mutator (Setter) Functions

Purpose:

- **A setter (or mutator) function is used to modify the value of a private or protected attribute in a class.**
- **They provide a controlled way of setting or updating data values, ensuring validation and consistency if needed.**

Why Use Mutator Functions?

- **Encapsulation: Encapsulation is one of the core principles of OOP. By using setter functions, you ensure that the internal data of an object is protected from being directly accessed or changed. This helps in maintaining data integrity.**
- **Validation: You can add logic inside the setter to validate the value before assigning it. For example, you can check if an age value is within a certain range.**
- **Control over changes: Setters give you the ability to track and control when and how an attribute's value changes.**

```
void setAge(int a) {
   if (a >= 0 && a <= 150) {  // Validate age
      age = a;
   } else {
      cout << "Invalid age!" << endl;
   }
}
```

2. Getter Functions

Purpose:

- A **getter** function is used to retrieve (or "get") the value of a private or protected attribute.
- They allow external code to access an object's attributes in a controlled manner.

Why Use Getter Functions?

- **Encapsulation:** Just like setters, getters help to encapsulate the data. Without getters, an object's attributes would be directly accessible, which could lead to incorrect values being accessed or modified.
- **Read-only access:** By using getters, you allow **read-only access** to data fields without exposing them for direct modification.
- **Control over access:** You can add additional logic inside the getter (like formatting or calculations) when retrieving the value.

```
int getAge() {
   return age;  // Just returns the age value
}
```

- ```
  #include <iostream>
  using namespace std;

  // Class definition
  class Person {
  private:
      string name;  // Private attribute
      int age;      // Private attribute

  public:
      // Setter function for name
      void setName(string n) {
          name = n;
      }

      // Getter function for name
      string getName() {
          return name;
      }

      // Setter function for age
      void setAge(int a) {
          age = a;
      }

      // Getter function for age
      int getAge() {
          return age;
      }
  };

  int main() {
      // Create an object of the Person class
      Person p1;
  ```
  ```

```
    // Set values using setter functions
    p1.setName("John");
    p1.setAge(25);

    // Retrieve and display values using getter functions
    cout << "Name: " << p1.getName() << endl;
    cout << "Age: " << p1.getAge() << endl;

    return 0;
}
```

Create a class called Student with private attributes: name and age. Use setter functions to set the values, and getter functions to retrieve and display them.

```
#include <iostream>
using namespace std;
// Define the class
class Student {
private:
    string name; // private attribute
    int age;     // private attribute
public:
    // Setter (mutator) function to set the name
    void setName(string n) {
        name = n;  }
    // Getter function to get the name
    string getName() {
        return name;
    }
    // Setter (mutator) function to set the age
    void setAge(int a) {
        age = a;
    }
```

```
 // Getter function to get the age
    int getAge() {
        return age;
    }
};
int main() {
    // Create an object of the class
    Student s1;
    // Use setter functions to set values
    s1.setName("Alice");
    s1.setAge(20);
    // Use getter functions to retrieve and display values
    cout << "Student Name: " << s1.getName() << endl;
    cout << "Student Age: " << s1.getAge() << endl;


    return 0;
}
```

Lab Tasks

Q.1. Create a **Person** class with a **private attribute** name. Use a setter function setName() to modify it.

Q.2. Use a Setter and Display the Value

Problem:
Modify the Person class to **add a function that prints the name**.

Q.3. Create a Class with Multiple Setters

Problem:
Create a class **Student** with **name** and **age** as private attributes. Use two setter functions to modify them.

Q.4. Calling Setters Inside a Function

Problem:
Modify the Student class to include a **function that sets both attributes at once**.

Q.5. Create Multiple Objects

Problem:
Modify the previous Student class to **create multiple objects** and set their values separately.

Q.6. Creating a Bank Account Class

Problem:
Create a **BankAccount** class with accountNumber and balance as private attributes. Provide setter functions to modify them.

Q.7. Employee Class with Function Calling

Problem:
Create a class Employee with **setter functions for name and salary**, and a display() function to print values.

Q.8. Modify Class Using a Function

Problem:
Modify the BankAccount class to allow updating balance with a **deposit() function**.

Q.9. Vehicle Class with Multiple Setters and Function Calling

Problem:
Create a Vehicle class with **setter functions for brand, model, and price**. Use a function to display all values.