Assignment 3

question number 1 ANS

```
class Book {
private:
    std::string title;
};


int main() {
    Book myBook;
    myBook.title = "C++ Programming"; // Error: 'title' is a private member of 'Book'
    std::cout << myBook.title << std::endl;
    return 0;
}
```

```
error: 'std::string Book::title' is private within this context
```

This error occurs because title is a private member, and private members cannot be accessed or modified directly from outside the class. This is a violation of encapsulation, a core principle of object-oriented programming.

Problem:

Direct access to private members from outside the class is not allowed.

Potential Solutions:

Use public setter and getter methods to access or modify private members.

Change the access specifier of title to public (not recommended for data integrity).

```
class Book {
public:
    std::string publisher;
};


int main() {
    Book myBook;
    myBook.publisher = "O'Reilly Media";
    std::cout << myBook.publisher << std::endl;

    return 0;

}
```

Why This Works:

Public members are accessible from anywhere in the program, so you can directly assign and print their values.

Is It Good to Always Use Public Specifier?

No, it is not good practice to always use the public specifier. Making all members public breaks encapsulation and data hiding, which can lead to unintended modifications and bugs. Use public only for members that are meant to be accessed from outside the class, and keep sensitive data private or protected.

```
class Book {

private:

    std::string author;

public:

    void displayInfo() {

        std::cout << "Author: " << author << std::endl;

    }

    void setAuthor(const std::string& a) {

        author = a;

    }

};


int main() {

    Book myBook;

    myBook.setAuthor("Bjarne Stroustrup");

    myBook.displayInfo(); // This works

    return 0;

}
```

Why This Works:

Public member functions can be called from outside the class. displayInfo is public, so it can be accessed from main.

What If Changed to Private?

If you change displayInfo to private, it cannot be called from outside the class, and the compiler will throw an error. Private functions are only accessible from within other member functions of the same class.

D. Access Specifiers/Modifiers in C++

Access specifiers/modifiers control the accessibility of class members:

Specifier	Accessibility	Typical Use Cases
public	Accessible from anywhere in the program	Interface functions, constants
private	Accessible only within the class itself	Internal data, helper methods
protected	Accessible within the class and its derived (child) classes, but not from outside	Base class data/functions for inheritance

Benefits:

Private: Enforces encapsulation and data hiding, protecting internal state.

Public: Allows controlled access to class functionality.

Protected: Supports inheritance by allowing derived classes to access base class members, but still hides them from outside code.

E. What Are Classes in C++ and Why Use Them?

A class in C++ is a user-defined data type that encapsulates data (attributes) and functions (methods) that operate on that data.

Why Use Classes?

They enable object-oriented programming (OOP), allowing you to model real-world entities.

Classes promote encapsulation, data abstraction, and code reuse.

They make complex systems manageable by logically grouping related code and data together.

Summary:

Classes are fundamental to C++ for organizing code, maintaining data integrity, and supporting OOP principles. Access specifiers help control how data and functions are exposed, ensuring robust and maintainable software.

A. Why the Program Can Be Compiled

A C++ program can be compiled successfully if you only access or modify class members that are marked as public. Public members are accessible from outside the class, including from the main function or other classes. For example, if you declare an attribute as public, you can set or get its value directly in your program without any compilation errors. Compilation errors typically occur when trying to access private members from outside the class, but with public members, such restrictions do not apply.

B. Good Methods to Set/Modify/Access Private Attributes

The recommended way to set, modify, or access private attributes in C++ is through public getter and setter methods. These methods provide controlled access to private data:

Getter: Returns the value of a private attribute.

Setter: Sets or modifies the value of a private attribute.

```
class MyClass {
private:
   int value;
public:
   void setValue(int v) { value = v; }
   int getValue() { return value; }
};
```

. Meaning of the Line:

group (string) = “2025 group”

This line is intended to assign the string value "2025 group" to a variable named group of type string. In correct C++ syntax, it should be written as:

cpp

```
std::string group = "2025 group";
```

Task	Explanation
Why program compiles	Only public members are accessed, which is allowed.
Good methods for private	Use public getter and setter methods for controlled access.
Meaning of assignment	Assigns a string value to a string variable; correct syntax is std::string group = ...

A. What is a Constructor? Why Do We Need to Use It?

A constructor in C++ is a special method in a class that is automatically called when an object of that class is created. Its main purpose is to initialize the object's data members and allocate any resources the object needs. Constructors have the same name as the class and do not have a return type. Using constructors ensures that objects start their life in a valid state, often with user-supplied or default values.

B. What is a Destructor? Why Do We Need to Use It?

A destructor is a special member function that is automatically called when an object goes out of scope or is explicitly deleted. Its purpose is to release resources acquired by the object, such as memory or file handles, and to perform any necessary cleanup. Destructors have the same name as the class, preceded by a tilde ~, and they do not take parameters or return a value. Using destructors prevents resource leaks and ensures proper cleanup.

```
class Car {
public:
    Car();      // Constructor
    ~Car();     // Destructor
};


Car::Car() {
    // Initialization code
}


Car::~Car() {
    // Cleanup code
}
```

Can Constructors or Destructors Return a Value?

No, constructors and destructors cannot return a value, not even void. They are designed this way because they are called automatically by the system, and their sole purpose is to initialize or clean up objects.

. Difference: Creating an Object with new vs. On the Stack

Method	Memory Location	Lifetime	Deletion Needed?
Car myCar;	Stack	Automatically destroyed when out of scope	No
Car* myCar = new Car;	Heap	Exists until delete myCar; is called	Yes, must use delete

What Happens If You Don’t Define a Constructor?

If you do not define any constructor, the compiler automatically provides a default constructor that initializes built-in types to indeterminate values and calls constructors for member objects. If your class needs special initialization, you must define your own constructor.

F. What Happens If You Don’t Define a Destructor?

If you do not define a destructor, the compiler generates a default destructor that performs no special action. This is sufficient unless your class manages resources (like dynamic memory, files, or network connections). In such cases, you must define a destructor to release those resources and prevent leaks.