**Objective**

* To get accustomed to the idea of C++ class and object concepts.
* To implement initializer methods for automatically setting up objects.
* To use clean-up methods for releasing system resources and understanding lifetime of objects.
* To promote well-structured and reusable programs by employing principles of OOP.

**Tools and Header Files Used:**

* Language Used: C++
* Development Environment: G++
* Header Files: include <iostream>, include <string>

**Theory:**

C++ classes function as user-defined data types that encapsulate both data (properties) and methods (actions). A initializer method is a special method that runs automatically upon object initialization, used to set up its starting configuration. The clean-up method is called when the object goes out of scope or is destroyed, releasing system resources or performing cleanup tasks. This combination forms the foundation of OOP, enabling code reuse, hiding internal details, and simplified upkeep.

**Instance**

In OOP, a problem is divided into a set of such entities, each retaining its own data and functionality.

* Instances are the main elements that come into action during a program’s execution.
* Interaction among objects happens through messages.
* An object's methods can directly access its own data members.

**Blueprint**

A class acts as a model or outline for creating numerous objects with similar properties and methods. Each object instantiated from a class maintains its own separate data.

* Once a class is defined, you can create multiple instances of it.
* A class typically regulates member accessibility through public, protected, or private visibility.
* This forms the basis for data hiding and restricted access to internals.

Syntax:

1. class BlueprintName {
2. private:
3. // private data members
4. // private member functions
5. protected:
6. // protected members (optional)
7. public:
8. // public data members
9. // public member functions
10. };

Example:

1. #include <iostream>
2. using namespace std;
3. class Demo {
4. private:
5. int a;
6. protected:
7. int b;
8. public:
9. int c;
10. void setValues() {
11. a = 10;
12. b = 20;
13. c = 30;
14. }
15. void display() {
16. cout << "Private a = " << a << endl;
17. cout << "Protected b = " << b << endl;
18. cout << "Public c = " << c << endl;
19. }
20. };
21. int main() {
22. Demo obj;
23. obj.setValues();
24. obj.display();
25. // obj.a = 1; // Error: private member
26. // obj.b = 2; // Error: protected member
27. obj.c = 3; // OK: public member
28. return 0;
29. }

**Initialization Functions**

* A initializer method is a special member function in C++.
* It is invoked by default when a new object is created.
* The main role of a initializer method is to initialize its data members.

Purpose:

* Initializes the object’s properties when it’s created.
* Allows you to set default or custom values for its members.
* Helps streamline object instantiation when multiple copies are needed.

Characteristics:

* The initializer method’s name is the same as its class.
* It’s automatically called upon object instantiation.
* If you do not provide a custom initializer method, the compiler generates a default one.
* The initializer method doesn’t have any return type.

Syntax:

1. class BlueprintName {
2. public:
3. BlueprintName(); // Default initializer method
4. };

Types of Initialization Functions

1. **Default initializer method**
2. class Test {
3. // No initializer method defined
4. };
5. int main() {
6. Test t1; // Compiler provides a default initializer method
7. return 0;
8. }
9. };
10. **Parameterized initializer method**
11. class BlueprintName {
12. public:
13. BlueprintName(data\_type parameter1, data\_type parameter2, ...);
14. };
15. **Copy initializer method**
16. class BlueprintName {
17. public:
18. BlueprintName(const BlueprintName &obj);
19. };

**Finalization Function**

* The clean-up method is a special method in C++.
* It is automatically called when an object is destroyed.
* It performs termination procedures and frees resources, if needed.

Purpose:

* Allows proper release of resources such as closing files or releasing memory.
* Helps avoid unreleased memory issues in programs employing dynamic memory.

Characteristics:

* The clean-up method’s name is the class’s name with a tilde (~).
* It does not take any parameters and returns nothing.
* Each class can have at most one clean-up method.
* It’s invoked by default when the object goes out of scope or is destroyed.

Syntax:

1. ~BlueprintName() {
2. // code to release resources
3. }

Example:

1. #include <iostream>
2. using namespace std;
3. class Demo {
4. public:
5. Demo() {
6. cout << "Initialization Function called." << endl;
7. }
8. ~Demo() {
9. cout << "Finalization Function called." << endl;
10. }
11. void show() {
12. cout << "Inside show function." << endl;
13. }
14. };
15. int main() {
16. Demo obj; // Initialization Function is called
17. obj.show(); // Method is called
18. return 0; // Finalization Function is invoked by default
19. }