Purpose of C++ Programming

The purpose of C++ was to introduce object-oriented programming (OOP) concepts to the C programming language, enhancing its capabilities while maintaining efficiency.

Principles of Object-Oriented Programming (OOP)

1. **Object**
    a. Basic unit of OOP.
    b. Bundles data and functions that operate on data into a single unit.
2. **Class**
    a. Blueprint for objects.
    b. Defines the structure and behavior of objects without specifying data.
    c. Combines data representation and methods into one package.
3. **Abstraction**
    a. Hides implementation details.
    b. Provides only essential information, making the program simpler.
4. **Encapsulation**
    a. Groups data and related functions in one unit (class).
    b. Protects data by restricting access to specific parts.
5. **Inheritance**
    a. Allows new classes (derived) to reuse code from existing classes (base).
    b. Reduces redundancy and improves maintainability.
6. **Polymorphism**
    a. Enables using operators or functions in multiple ways.
    b. Includes:
        i. **Overloading:** Extending functionality of existing operators or functions for new data types.

C++ Class and Object Essentials

Class Definition Syntax

```
class ClassName {
    access_specifier:
    // Body of the class
};
```

- **Access Specifiers:**
    - `public`: Accessible from outside the class.

- `private`: Accessible only within the class.
- `protected`: Accessible within the class and derived classes.

Example: Class Definition

```
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
```

Object Creation

- Objects are instances of a class.
- Syntax:

```
ClassName ObjectName;
```

- Example:

```
Box Box1;  // Declare Box1 of type Box
```

Accessing Data Members

- Use the `.` (dot) operator for public members.
- Example:

```
Box1.length = 5.0;
```

Example: Volume and Area of a Box

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

class Box {
   public:
      double length, breadth, height;
};

int main() {
   Box Box1, Box2;
```

```
    double volume, area;

    // Box1 specifications
    Box1.length = 6.0; Box1.breadth = 7.0; Box1.height = 5.0;

    // Box2 specifications
    Box2.length = 12.0; Box2.breadth = 13.0; Box2.height = 10.0;

    // Volume and Area calculations
    volume = Box1.length * Box1.breadth * Box1.height;
    area = Box1.length * Box1.breadth;
    cout << "Volume of Box1: " << volume << "\nArea of Box1: " << area << endl;

    volume = Box2.length * Box2.breadth * Box2.height;
    area = Box2.length * Box2.breadth;
    cout << "Volume of Box2: " << volume << "\nArea of Box2: " << area << endl;

    return 0;
}
```

Why Use namespace std?

- Avoids naming conflicts by providing a scope for standard library functions and variables.
- Without it, every standard function would require the `std::` prefix.

Advanced Concepts in C++

1. **Member Functions**
    a. Functions defined within or outside the class that operate on class members.
2. **Access Modifiers**
    a. Default is `private`.
    b. Used to control visibility and accessibility.
3. **Constructor & Destructor**
    a. **Constructor:** Special function called when an object is created.
    b. **Destructor:** Special function called when an object is destroyed.
4. **Copy Constructor**
    a. Initializes an object using another object of the same class.
5. **Friend Functions**
    a. Functions allowed to access private and protected members of a class.
6. **Inline Functions**
    a. Compiler replaces function call with function code to improve efficiency.

7. **`this` Pointer**
    a. Special pointer pointing to the current object.
8. **Static Members**
    a. Shared by all objects of the class.
    b. Retains value across function calls.

Defining Member Functions Outside Class

- Use the `::` (scope resolution) operator.
- Example:

```
class Geeks {
   public:
      string name;
      void printName();
};

void Geeks::printName() {
    cout << "Name: " << name;
}
```

Examples of Class Usage

Bank Account Example

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

class BankAccount {
public:
    string name;
    int ac_num;
    double balance;

    void displayDetails() {
        cout << "Name: " << name << "\nAccount number: " << ac_num
             << "\nBalance: " << balance << endl;
    }
};

int main() {
    BankAccount account;
```

```
    account.name = "Aswin";
    account.ac_num = 123123123;
    account.balance = 100000;
    account.displayDetails();
    return 0;
}
```

Accessing Data with Access Specifiers

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

class Geeks {
private:
    string name;
    int num;
    double balance;

public:
    void setDetails(string n, int id, double bal) {
        name = n; num = id; balance = bal;
    }
    void getDetails() {
        cout << "Name: " << name << "\nID: " << num << "\nBalance: " << balance
<< endl;
    }
};

int main() {
    Geeks obj;
    obj.setDetails("Aswin", 123123123, 100000);
    obj.getDetails();
    return 0;
}
```

Conclusion

C++ offers a rich set of features to support object-oriented programming, enabling developers to design robust, reusable, and efficient software solutions.