Object Oriented Programming using C++

Overview of C++

- Object-oriented programming is a programming paradigm.

- It is used to build more reliable and reusable systems.

- There are two main programming paradigms:

1. Procedure-oriented programming
2. Object-oriented programming

Structured programming

- Focuses on logic rather than data.

- Emphasizes algorithms over data.

- Programs are divided into modules.

- Uses independent functions (procedures) for discrete tasks.

- Does not support inheritance or polymorphism.

- In procedural languages like FORTRAN, PASCAL, COBOL, and C, a program is a sequence of instructions.

- Each statement directs the computer to perform a specific action.

- Procedural approach has its limitations:

Complexity: Large programs become difficult to debug and maintain.

Data undervalued: Emphasizes actions rather than data, leading to less secure data management. Global variables can be accessed and modified by any function.

Object oriented programming

- To address the limitations of procedural programming, the concept of Object-Oriented Programming (OOP) was developed.

- OOP provides a new approach to solving problems with computers.

- OOP languages enable programmers to create class hierarchies.

- Programmers can develop modular and reusable code.

- Existing modules can be modified as needed.

- The core idea of OOP is to combine both data and functions into a single unit known as an object.

- Functions within objects are referred to as member functions, which facilitate data access.

- To read a data item from an object, you call the relevant member function.

- The member function retrieves and returns the data item; direct access to the data is restricted. The data is hidden.

- Data and its functions are encapsulated within a single entity called an object.

OOP Definition: Object-Oriented Programming (OOP) is a technique where a computer program is designed and written around objects.

Fundamental Features of OOP:

- Encapsulation
- Data Abstraction
- Inheritance
- Polymorphism
- Message Passing
- Extensibility
- Persistence
- Delegation
- Genericity
- Multiple Inheritance

Class

- Class is a template (format).
- The C++ class mechanism allows users to define their own data types.
- For this reason, classes are called user-defined types.
- A class definition has two parts:
  - class head - composed of the keyword class followed by the class name
  - class body - enclosed by a pair of curly braces.
- A class definition must be followed either by a semicolon or a list of declarations. For example:

  class Screen {..................

  class Screen { .... } myScreen, yourScreen;

Objects

A class represents a logical abstraction. An object has a physical existence. An object is an instance of a class. An object is a combination or collection of data and code designed to emulate a physical or abstract entity.

Example:

```
class Screen {
   // myScreen and yourScreen are objects of the class Screen
   // Screen is the class name
   // myScreen and yourScreen are instances of the Screen class
}
```

- Class Name: Screen
- Objects: myScreen and yourScreen

The general form for defining a class and its objects is:

```
class class-name {
  private:
    // data and functions with private access specifier
  public:
    // data and functions with public access specifier
} object-list;
```

Example for representation of class

```
class Account {
private:
   char name[20];     // Name of the account holder
   int acc_type;      // Type of account (e.g., savings, checking)
   int acc_no;        // Account number
   float balance;     // Current balance in the account
public:
   // Member functions
   void deposit();    // Function to deposit money into the account
   void withdraw();   // Function to withdraw money from the account
   void enquire();    // Function to enquire about the account balance or details
};
```

- Objects serve the following purposes:

 - Understanding the Real World: They provide a practical basis for designers by modeling real-world entities.

 - Decomposition of Problems: Objects help in breaking down problems into manageable pieces based on judgment and the nature of the problem.

 - Attributes and Behavior: Each object has attributes (data structures representing its properties) and behavior (operations or methods that define its actions).

Encapsulation

- It is a mechanism that associates code and data into a single unit, supported by a class.

- Encapsulation binds together methods and data, keeping both safe from outside interference and misuse.

- In object-oriented languages, encapsulation creates a "black box" by combining functions and data.

- The process of wrapping data and functions into a single unit is known as encapsulation.

Data Abstraction

- The technique of creating new data types that are well-suited to the specific needs of an application is known as data abstraction.

- The data types created through data abstraction are called **Abstract Data Types (ADTs).**

Inheritance

- It is the process by which an object of one class acquires the properties of objects from another class.

- The class that inherits properties from another class is referred to as the derived class (child class), while the class providing properties is known as the base class (parent class).

- Inheritance establishes a parent-child relationship between the base class and the derived class.

- It enables the extension and reuse of existing code without the need to rewrite it from scratch.

- The derived class inherits the members of the base class and can also introduce its own members.

- Example: When class `B` inherits from class `A`, class `B` is referred to as the derived class or subclass.

- Class `A` is known as the base class or superclass.

- Class `B` consists of two parts:

  - Derived Part: The components inherited from the base class `A`.

  - Incremental Part: The new code that is added to class `B` beyond what is inherited from class `A`.

- For example, Maruthi, sports cars and Benz are all types of cars.
- In the object oriented language, sports cars, Maruthi and Benz are subclasses of the class car.
- The class car is a "super class" (parent class or base class) of Maruthi, Benz, and sports cars.
- Every subclass will inherit data (state) and functions (properties) from the super class.
- The various types of cars such as Maruthi and Benz will share certain properties such as break, escalator, steering etc.
- The attribute once declared in the super class which are inherited by its subclasses, need not repeated. They can be accessed form any subclass unless they are private.
- Only the methods of a class can access its private attributes.
- The attributes which are declared as protected are accessible to subclasses.

- Single Inheritance: Involves deriving a class from a single base class.

- Multiple Inheritance: Involves deriving a class from more than one base class.

Polymorphism

- Derived from "poly" meaning many and "morph" meaning forms, polymorphism means "many forms."

- It allows a single name or operator to be associated with different operations based on the data provided.

- Polymorphism is the ability to use an operator or function in multiple ways, giving different meanings depending on the context.

- Essentially, polymorphism enables a single function or operator to behave differently based on how it is used.

- The two types of polymorphism are operator overloading and function overloading.

Function Overloading

- Example of Function Overloading: Function overloading occurs when multiple methods in a class have the same name but differ in their parameters, such as `Calculate(int a, float b)`, `Calculate(int a, int b)`, and `Calculate(float a, float b)`.

- In this case, the appropriate `Calculate` function is executed based on the type and number of arguments passed to it.

Message Passing

- In an object-oriented language, a message is sent to an object.

- This process involves invoking an operation on the object in response to a message, which triggers the execution of the corresponding method in the object.

- A message to an object is interpreted as a request to execute a function.

- When the object receives a message, the appropriate function is invoked, and the result is generated within the object.

- Example: `student.marks(rollNo)`

  - In this example, `marks()` is the message with `rollNo` as the parameter, and `student` is the object.

  - The message `marks()` requests the execution of the function with `rollNo` as the information passed to the object.

- Objects have a lifetime during which they can be created and destroyed. Communication between objects can occur as long as they are alive.

- Extensibility: Allows for the extension of the functionality of existing software components.

- Persistence: Refers to the phenomenon where an object outlives the program execution time, existing between different runs of the program.

- Genericity: Enables the declaration of data items without specifying their exact datatype.

Delegation

- The two most common techniques for reusing functionality in object-oriented systems are class inheritance and object composition.

- Class Inheritance: If class `B` is derived from class `A`, then `B` is considered a specialized kind of `A`.

- Object Composition: An object can be a collection of other objects, and this relationship is known as a "has-a" relationship or containership.

- Delegation: This technique makes object composition as powerful as inheritance for reuse. It involves two objects: a receiving object delegates operations to another object (similar to subclasses sending requests to parent classes).

- In some cases, inheritance and containership can serve similar purposes.

- A C++ program consists of multiple objects that communicate by calling each other's member functions, which are known as methods.

- Calling a member function of an object is referred to as sending a message to the object.

- The internal structure of an object is hidden from the user, a property known as data/information hiding or data encapsulation.

- Both attributes (data) and methods (functions) are members of a class.

- Members are declared as either private or public:

  - Public Members: Can be accessed by any function.

  - Private Members: Can only be accessed by methods of the same class.

- C++ has special functions:

  Constructor: Initializes new instances of a class.

  Destructor: Performs necessary cleanup when an object is destroyed.

- C++ provides three types of memory allocations for objects:

  Static: Pre-allocated by the compiler, typically for variables declared outside functions using the `static` keyword.

  Automatic: Allocated on the stack, used for local variables within functions.

  Dynamic: Allocated from the heap based on explicit requests from the programmer.

C++

C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C programming language. Initially named "C with Classes," it was renamed to "C++" in 1983.

The C++ program

- In C++, an action is referred to as an expression.

- An expression terminated by a semicolon is known as a statement.

- The smallest independent unit in a C++ program is a statement.

- Example: `int book_count = 0;`

- This is a declaration statement. `book_count` is referred to as an identifier, variable, symbolic variable, or object.

- Every C++ program must contain a function called `main`.

- A C++ program begins execution with the first statement of the `main()` function.

Functions

- A function in C++ consists of four parts:

- Return Type: Specifies the type of value the function will return.
- Function Name: The identifier used to call the function.
- Parameter List: A comma-separated list of parameters enclosed in parentheses, which may include zero or more parameters.
- Function Body: The definition of the function, enclosed in curly braces, containing a sequence of program statements.

- The first three parts (return type, function name, and parameter list) are collectively known as the function prototype.

- The parameter list is enclosed in parentheses and may be empty or contain multiple parameters.

- The function body is enclosed in curly braces `{}` and includes the executable code of the function.

Errors

- - One of the compiler's tasks is to analyze the program for correctness.
- - While the compiler can identify syntax and type errors, it cannot verify whether the meaning or logic of the program is correct.
- - Two common forms of program errors are:
- - Syntax Error: Errors in the code structure, such as missing semicolons or mismatched parentheses, that prevent the program from being compiled.
- - Type Error: Errors involving incorrect use of data types, such as assigning a string to an integer variable, which can lead to compilation issues.

- Syntax Error:

  - Occurs when the programmer makes a grammatical mistake in the C++ program. This could include issues like missing semicolons, incorrect use of keywords, or improper syntax that prevents the program from compiling.

- Type Error:

  - In C++, each data item has a specific type (e.g., integers, strings). For example, the value `10` is an integer, while the word `"hello"` is a string. A type error occurs if a function expects an integer argument but receives a string instead. The compiler will signal this mismatch as a type error.

Comment

- - The main purpose of comments is to assist human readers of the program.
- - Comments help both the programmer writing the code and anyone else who needs to read the source file or code by explaining what the code does and how it works.
- - The compiler ignores comments, so they do not affect the execution of the program.
- - Comments are useful for describing parts of the code and providing additional context or explanations.

In C++, there are two types of comment delimiters:

Single-Line Comments: Start with a double slash // and continue to the end of the line.

Used for brief comments on a single line.

Example: // This is a program to add two numbers

Multi-Line Comments: Enclosed between /* and */.

The compiler treats everything between /* and */ as part of the comment, which can span multiple lines.

Example:

```
/* This is a program
   to add two numbers */
```

Input / Output

- In C++, input and output operations are managed by the standard library known as the iostream library.

- Input from the terminal (standard input) is handled by the predefined iostream object `cin`.

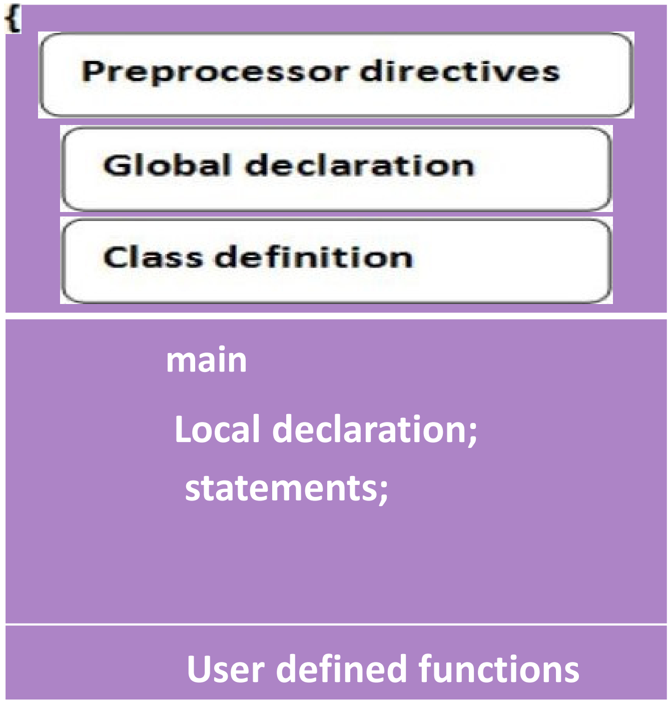
- Output to the terminal (standard output) is managed by the predefined iostream object `cout`.

- To use `cin` and `cout` in a program, you must include the following statement at the beginning of your code: `#include <iostream>`. Note that in modern C++, the correct header file is `<iostream>`, not `<iostream.h>`.

- C++ uses the bit-wise left shift operator (`<<`) for output operations.
  - Example: `cout << "Hello world";`
  - This symbol is called the insertion or put-to operator.
  - Multiple items can be displayed using a single `cout` object.
    - Example: `cout << "age = " << age;`

- C++ uses the bit-wise right shift operator (`>>`) for input operations.
  - Syntax: `cin >> variable;`
  - Example: `cin >> age;`
  - This symbol is called the extraction operator.
  - Multiple items can be read using a single `cin` object.
    - Example: `cin >> name >> age;`

Basic structure of C++ program



{

Preprocessor directives

Global declaration

Class definition

main

Local declaration;

statements;

User defined functions

A sample C++ program

```
// Program to display "Hello World"
#include <iostream>  // Preprocessor directive to include the iostream library

using namespace std;  // To use standard namespace

int main() {  // Function declaration
   cout << "Hello world, SIT, Valachil, Mangalore";  // Output statement
   return 0;  // End of the program
}
```

Key Points:

- `#include <iostream>`: Includes the iostream library for input and output operations.

- `using namespace std;`: Allows you to use standard library names without prefixing them with `std::`.

- `int main()`: The main function, where program execution starts.

- `cout << "Hello world, SIT, Valachil, Mangalore";`: Displays the message to the console.

- `return 0;`: Indicates that the program has ended successfully.

- The source code of the program is written in a text editor.

- The source code is then compiled to convert it into machine code.

- C++ programs use libraries that contain the object code of standard functions. The object code for all functions used in the program must be combined with the programmer's code.

- Startup code is also required to produce an executable version of the program.

- The process of combining the necessary object code and startup code to create an executable file is called linking. This process results in the production of executable code.

Scope resolution operator

In C++, when a local variable has the same name as a global variable, the scope resolution operator (::) is used to access the global variable from within the function. Here’s an example illustrating this:

```
#include <iostream>  // Preprocessor directive to include the iostream library

using namespace std;  // To use standard namespace

int num = 20;  // Global variable

int main() {

   int num = 30;  // Local variable with the same name as the global variable

   cout << "Local = " << num << endl;            // Outputs the local variable

   cout << "Global = " << ::num << endl;         // Accesses the global variable using the scope resolution operator

   cout << "Sum = " << ::num + num << endl;          // Calculates and outputs the sum of the global and local variables

   return 0;  // End of the program

}
```

Output:

```
Local = 30
Global = 20
Sum = 50
```

Manipulators

Manipulators in C++ are operators used to format the display of data.

Common manipulators include `endl` and `setw`:

- `endl`:
  - Indicates the end of a line, causing the next output to appear on a new line.
  - Example: `cout << "Hello" << endl;`

- `setw`:
  - Specifies the field width for the data, ensuring that the data is right-justified within the given width. If the data has fewer characters than the specified width, spaces are added to the right.
  - Usage: `setw(n)` where `n` is the width of the field.

- Example:

```cpp
#include <iostream>
#include <iomanip>  // Required for setw

using namespace std;

int main() {
    cout << setw(10) << 42 << endl;  // Output: "        42" (with 8 spaces before 42)
    return 0;
}
```