Chapter 8. Introduction to C++

8.1 Object oriented concepts, Features,

8.2 Advantages and Applications of OOPS

8.3 Data types, new operators and keywords, type conversion in C++

8.4 Classes & Objects

Introduction:

The C++ were first invented by **Bjarne Stroustrup in 1979** at Bell Laboratories in Murray Hill, New Jersey. Bjarne Stroustrup initially called the new language "C with Classes." However, in 1983 the name was changed to C++. C++ is a middle- level programming language. **C++** is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a compiled language. C++ is a middle-level language rendering it the advantage of programming low-level (drivers, kernels) and even higher-level applications (games, GUI, desktop apps etc.). The basic syntax and code structure of both C and C++ are the same.

8.1 Features of C++ / Object oriented Programming:

The prime purpose of C++ programming was to add object orientation to the C programming language, which is in itself one of the most powerful programming languages.

The core of the pure object-oriented programming is to create an object, in code, that has certain properties and methods. While designing C++ modules, we try to see whole world in the form of objects. For example a car is an object which has certain properties such as color, number of doors, and the like. It also has certain methods such as accelerate, brake, and so on.

There are a few principle concepts that form the foundation of object-oriented programming −

The important features of Object Oriented programming are:

- **Inheritance**
- **Polymorphism**
- **Abstraction**
- **Encapsulation**
- **Overloading**
- **Objects**
- **Classes**

Object

This is the basic unit of object oriented programming. That is both data and function that operate on data are bundled as a unit called as object.

Class

When you define a class, you define a blueprint for an object. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

Abstraction

Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

For example, a database system hides certain details of how data is stored and created and maintained. Similar way, C++ classes provides different methods to the outside world without giving internal detail about those methods and data.

Encapsulation

Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you framework to place the data and the relevant functions together in the same object.

Inheritance

One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.

This is a very important concept of object-oriented programming since this feature helps to reduce the code size.

Polymorphism

The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism. Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.

Overloading

The concept of overloading is also a branch of polymorphism. When the exiting operator or function is made to operate on new data type, it is said to be overloaded.

8.2 Advantages of OOP:

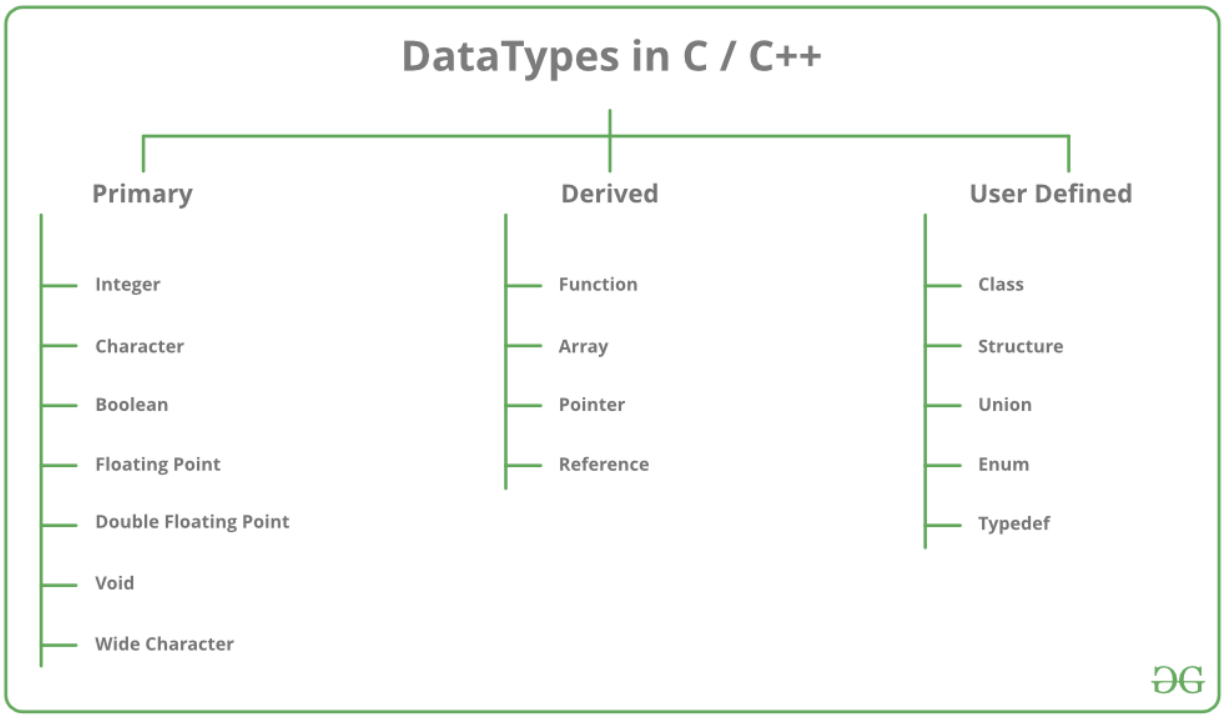
- It provides a clear ***modular structure*** for programs which makes it good for defining abstract data types in which implementation details are hidden
- Objects can also be ***reused*** within an across applications. The reuse of software also lowers the cost of development. More effort is put into the object-oriented analysis and design, which lowers the overall cost of development.
- It makes software ***easier to maintain.*** Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes
- Reuse also enables ***faster development***. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.
- It provides a good framework for code libraries where the supplied software components can be ***easily adapted and modified by the***

programmer. This is particularly useful for developing graphical user interfaces.

- ***Better Productivity*** **as** OOP techniques enforce rules on a programmer that, in the long run, help her get more work done; finished programs work better, have more features and are easier to read and maintain. OOP programmers take new and existing software objects and "stitch" them together to make new programs. Because object libraries contain many useful functions, software developers don't have to reinvent the wheel as often; more of their time goes into making the new program.

8.3 C++ Data Types

All **variables** use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared. Every data type requires a different amount of memory



Data types in C++ is mainly divided into three types:

1. **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:
   - Integer
   - Character
   - Boolean
   - Floating Point
   - Double Floating Point
   - Valueless or Void
   - Wide Character
2. **Derived Data Types:** The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:
   - Function
   - Array
   - Pointer
   - Reference
3. **Abstract or User-Defined Data Types**: These data types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:
   - Class
   - Structure
   - Union
   - Enumeration
   - Typedef defined DataType

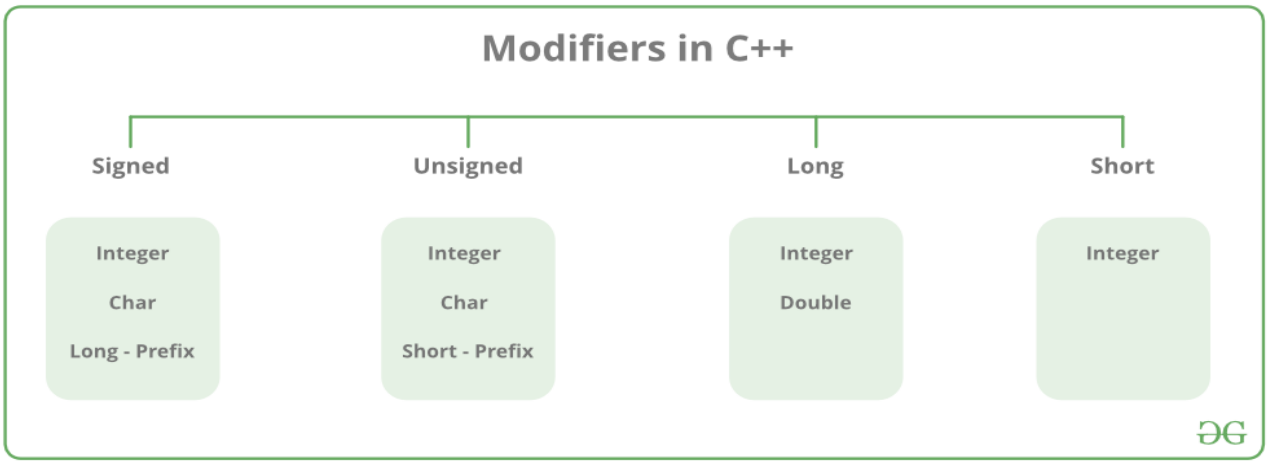
primitive data types available in C++.

- **Integer**: Keyword used for integer data types is **int**. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
- **Character**: Character data type is used for storing characters. Keyword used for character data type is **char**. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.

- **Boolean**: Boolean data type is used for storing boolean or logical values. A boolean variable can store either *true* or *false*. Keyword used for boolean data type is **bool**.
- **Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is **float**. Float variables typically requires 4 byte of memory space.
- **Double Floating Point**: Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is **double**. Double variables typically requires 8 byte of memory space.
- **void**: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.

➢ **Data type Modifiers**

- As the name implies, data type modifiers are used with the built-in data types to modify the length of data that a particular data type can hold.



Data type modifiers available in C++ are:

- **Signed**
- **Unsigned**
- **Short**
- **Long**

Below table summarizes the modified size and range of built-in datatypes when combined with the type modifiers:

DATA TYPE	SIZE (IN BYTES)	RANGE
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	

C++ program to sizes of data types

```
#include<iostream>
#include<conio.h>
int main()
{
   cout << "Size of char : " << sizeof(char)
     << " byte" << endl;
   cout << "Size of int : " << sizeof(int)
     << " bytes" << endl;
   cout << "Size of short int : " << sizeof(short int)
     << " bytes" << endl;
```

```
    cout << "Size of long int : " << sizeof(long int)
      << " bytes" << endl;
    cout << "Size of signed long int : " << sizeof(signed long int)
      << " bytes" << endl;
    cout << "Size of unsigned long int : " << sizeof(unsigned long int)
      << " bytes" << endl;
    cout << "Size of float : " << sizeof(float)
      << " bytes" <<endl;
    cout << "Size of double : " << sizeof(double)
      << " bytes" << endl;
    cout << "Size of wchar_t : " << sizeof(wchar_t)
      << " bytes" <<endl;

    return 0;
}
```

Output:

```
Size of char : 1 byte
Size of int : 4 bytes
Size of short int : 2 bytes
Size of long int : 8 bytes
Size of signed long int : 8 bytes
Size of unsigned long int : 8 bytes
Size of float : 4 bytes
Size of double : 8 bytes
Size of wchar_t : 4 bytes
```

8.4 New operators in C++:

1. **insertion operator:**

The insertion operator << is the one we usually use for output, as in:

```
cout << "This is output" << endl;
```

It gets its name from the idea of inserting data into the output stream.

2. extraction operator:

The extraction operator >> is the one we usually use for input, as in:

```
cin >> X;
```

It gets its name from the idea of extracting data from the input stream.

3. Scope resolution Operator:

The :: (scope resolution) operator is used to get hidden names due to variable scopes so that you can still use them. The scope resolution operator can be used as both unary and binary. You can use the unary scope operator if a namespace scope or global scope name is hidden by a particular declaration of an equivalent name during a block or class.

For example,

if you have a **global variable of name my_var** and a **local variable of name my_var,**

to access global my_var, you'll need to use the scope resolution operator.

example

```
#include <iostream>
#include<conio.h>
int my_var = 0;// above main funtion global variable
int main( )
{
  int my_var = 0;// within main function local variable
  ::my_var = 1;  // set global my_var to 1
```

```
   my_var = 2;    // set local my_var to 2
   cout << ::my_var << ", " << my_var;
   return 0;
getch();
}
```

Output

This will give the output −

```
1, 2
```

4. endl operator:

endl is the line feed operator in C++. It acts as a stream manipulator whose purpose is to feed the whole line and then point the cursor to the beginning of the next line. We can use \n (\n is an escape sequence) instead of endl for the same purpose.

For example:

```
1
2 #include <iostream>
3  main()
4 {
5     cout << "Hi!" << endl; // will cause the cursor to move to the next line of the
6     cout << "My name is Raj." << endl;
7  return 0;
8 }
9
```

This prints:

```
Hi!
My name is Raj.
```

8.5 Basic input and output Statements in C++:(cin and cout)

Using the C++ iostream library we will get the user's input from the keyboard and we will print messages onto the screen. The iostream library is part of the C++ standard library.In C++, I/O is performed by using streams. A stream is a "stream of data" in which character sequences are "flow into" or "flow out off." A stream is an object with properties that are defined by a class. Global objects are predefined for the standard I/O channels.The header file iostream must be included to make use of the input/output (cin/cout) operators.

➢ Standard output stream (cout)

The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream **insertion operator (<<)** to display the output on a console

```
#include <iostream>
 main( )
{
  char ary[] = "Welcome to Shahu";
  cout << "Value of ary is: " << ary << endl;
}
```

Output:

```
Value of ary is: Welcome to Shahu
```

➢ Standard input stream (cin) :

The **cin** is a predefined object of **istream** class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Let's see the simple example of standard input stream (cin):

```
#include <iostream>
 main( )
{
  int age;
   cout << "Enter your age: ";
   cin >> age;
   cout << "Your age is: " << age << endl;
}
```

Output:

```
Enter your age: 22
Your age is: 22
```

8.6 Class and object:

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called members of the class.

➢ C++ Class

A class definition starts with the keyword **class** followed by the class name; and the 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, we defined the Box data type using the keyword **class** as follows −

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

};

The keyword **public** : determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object.

The keyword **Private:** You can also specify the members of a class as **private** or **protected** which we will discuss in a sub-section.

➢ C++ Objects

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box −

```
Box Box1;        // Declare Box1 of type Box
Box Box2;        // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

Accessing the Data Members

The public data members of objects of a **class can be accessed using the direct member access operator (.)**

example −

```
#include <iostream>

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

 main()
{
  Box Box1;        // Declare Box1 of type Box
  Box Box2;        // Declare Box2 of type Box
  double volume = 0.0;     // Store the volume of a box here
```

```
  // box 1 specification
  Box1.height = 5.0;
  Box1.length = 6.0;
  Box1.breadth = 7.0;

  // box 2 specification
  Box2.height = 10.0;
  Box2.length = 12.0;
  Box2.breadth = 13.0;

  // volume of box 1
  volume = Box1.height * Box1.length * Box1.breadth;
  cout << "Volume of Box1 : " << volume <<endl;

  // volume of box 2
  volume = Box2.height * Box2.length * Box2.breadth;
  cout << "Volume of Box2 : " << volume <<endl;
  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

Volume of Box1 : 210
Volume of Box2 : 1560

It is important to note that private and protected members can not be accessed directly using direct member access operator (.)