Unit-I

Introduction

Object oriented Programming

Object oriented Programming is defined as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand. Writing object-oriented programs involves creating classes, creating objects from those classes, and creating applications, which are stand-alone executable programs that use those objects. After being created, classes can be reused over and over again to develop new programs. Thinking in an object-oriented manner involves envisioning program components as objects that belong to classes and are similar to concrete objects in the real world; then, you can manipulate the objects and have them interrelate with each other to achieve a desired result.

Basic Concepts of Object oriented Programming

1. Class

A class is a user defined data type. A class is a logical abstraction. It is a template that defines the form of an object. A class specifies both code and data. It is not until an object of that class has been created that a physical representation of that class exists in memory. When you define a class, you declare the data that it contains and the code that operates on that data. Data is contained in instance variables defined by the class known as data members, and code is contained in functions known as member functions. The code and data that constitute a class are called members of the class.

2. Object

An object is an identifiable entity with specific characteristics and behavior. An object is said to be an instance of a class. Defining an object is similar to defining a variable of any data type. Space is set aside for it in memory.

3. Encapsulation

Encapsulation is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse. C++'s basic unit of encapsulation is the class. Within a class, code or data or both may be private to that object or public. Private code or data is known to and accessible by only another part of the object. That is, private code or data cannot be accessed by a piece of the program that exists outside the object. When code or data is public, other parts of your program can access it even though it is defined within an object. Typically, the public parts of an object are used to provide a controlled interface to the private elements of the object. This insulation of the data from direct access by the program is called data hiding.

4. Data abstraction

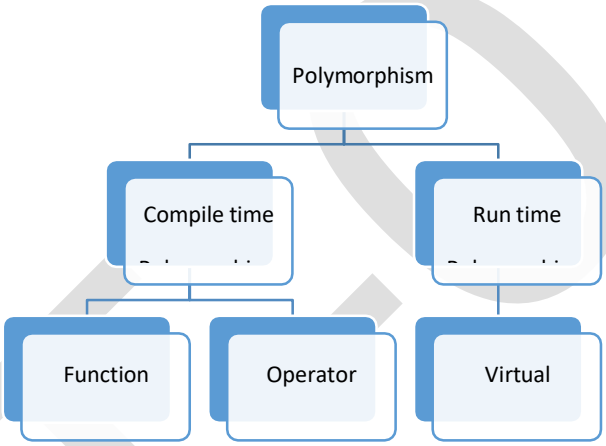
In object oriented programming, each object will have external interfaces through which it can be made use of. There is no need to look into its inner details. The object itself may be made of many smaller objects again with proper interfaces. The user needs to know the external interfaces only to make use of an object. The internal details of the objects are hidden which makes them abstract. The technique of hiding internal details in an object is called data abstraction.

5. Inheritance

Inheritance is the mechanism by which one class can inherit the properties of another. It allows a hierarchy of classes to be build, moving from the most general to the most specific. When one class is inherited by another, the class that is inherited is called the base class. The inheriting class is called the derived class. In general, the process of inheritance begins with the definition of a base class. The base class defines all qualities that will be common to any derived class. . In OOPs, the concept of inheritance provides the idea of reusability. In essence, the base class represent the most general description of a set of traits. The derived class inherits those general traits and adds properties that are specific to that class.

6. Polymorphism

Polymorphism (from the Greek, meaning "many forms") is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. The concept of polymorphism is often expressed by the phrase "one interface, multiple methods." This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler's job to select the specific action as it applies to each situation.



In compile time polymorphism, the compiler is able to select the appropriate function for a particular call at compile time. In C++, it is possible to use one function name for many different purposes. This type of polymorphism is called function overloading. Polymorphism can also be applied to operators. In that case, it is called operator overloading.

In run time polymorphism, the compiler selects the appropriate function for a particular call while the program is running. C++ supports a mechanism known as virtual functions to achieve run time polymorphism.

Need for Object oriented Programming

Object-oriented programming scales very well, from the most trivial of problems to the most complex tasks. It provides a form of abstraction that resonates with techniques people use to solve problems in their everyday life.

Object-oriented programming was developed because limitations were discovered in earlier approaches to programming. There were two related problems. First, functions have unrestricted access to global data. Second, unrelated functions and data, the basis of the procedural paradigm, provide a poor model of the real world.

Benefits of Object oriented Programming

1. Simplicity: Software objects model real world objects, so the complexity is reduced and the program structure is very clear.

2. Modularity: Each object forms a separate entity whose internal workings are decoupled from other parts of the system.

3. Modifiability: It is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.

4. Extensibility: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.

5. Maintainability: objects can be maintained separately, making locating and fixing problems easier.

6. Re-usability: objects can be reused in different programs.

C++

C++ is an object oriented programming language. It was developed by Bjarne Stroustrup in 1979 at Bell Laboratories in Murray Hill, New Jersey. He initially called the new language "C with Classes." However, in 1983 the name was changed to C++.

C++ is a superset of C. Stroustrup built C++ on the foundation of C, including all of C's features, attributes, and benefits. Most of the features that Stroustrup added to C were designed to support object-oriented programming .These features comprise of classes, inheritance, function overloading and operator overloading. C++ has many other new features as well, including an improved approach to input/output (I/O) and a new way to write comments.

C++ is used for developing applications such as editors, databases, personal file systems, networking utilities, and communication programs. Because C++ shares C's efficiency, much high-performance systems software is constructed using C++.

A Simple C++ Program

```
#include<iostream.h>

#include<conio.h>

int main()

{

 cout<< "Simple C++ program without using class";

return 0;

}
```

Lines beginning with a hash sign (#) are directives read and interpreted by what is known as the preprocessor. They are special lines interpreted before the compilation of the program itself begins. In this case, the directive #include <iostream.h>, instructs the preprocessor to include a section of standard C++ code, known as header iostream that allows to perform standard input and output operations, such as writing the output of this program to the screen.

The function named main is a special function in all C++ programs; it is the function called when the program is run. The execution of all C++ programs begins with the main function, regardless of where the function is actually located within the code.

The open brace ({) indicates the beginning of main's function definition, and the closing brace (}) indicates its end.

The statement :

cout<< “Simple C++ program without using class”;

causes the string in quotation marks to be displayed on the screen. The identifier cout (pronounced as c out) denotes an object. It points to the standard output device namely the console monitor. The operator << is called insertion operator. It directs the string on its right to the object on its left.

The program ends with this statement:

return 0;

This causes zero to be returned to the calling process (which is usually the operating system). Returning zero indicates that the program terminated normally. Abnormal program termination should be signaled by returning a nonzero value.

The general structure of C++ program with classes is shown as:

1. Documentation Section

2. Preprocessor Directives or Compiler Directives Section

(i) Link Section

(ii) Definition Section

3. Global Declaration Section

4. Class declaration or definition

5. Main C++ program function called main ( )

C++ keywords

When a language is defined, one has to design a set of instructions to be used for communicating with the computer to carry out specific operations. The set of instructions which are used in programming, are called keywords. These are also known as reserved words of the language. They have a specific meaning for the C++ compiler and should be used for giving specific instructions to the computer. These words cannot be used for any other purpose, such as naming a variable. C++ is a case-sensitive language, and it requires that all keywords be in lowercase. C++ keywords are:

asm	auto	bool	break
case	catch	char	class
const	const_cast	continue	default
delete	do	double	dynamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static
static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

Identifiers

An identifier is a name assigned to a function, variable, or any other user-defined item. Identifiers can be from one to several characters long.

Rules for naming identifiers:

- Variable names can start with any letter of the alphabet or an underscore. Next comes a letter, a digit, or an underscore.
- Uppercase and lowercase are distinct.
- C++ keywords cannot be used as identifier.

Data types

Data type defines size and type of values that a variable can store along with the set of operations that can be performed on that variable. C++ provides built-in data types that correspond to integers, characters, floating-point values, and Boolean values. There are the seven basic data types in C++ as shown below:

Type	**Meaning**
char(character)	holds 8-bit ASCII characters
wchar_t(Wide character)	holds characters that are part of large character sets
int(Integer)	represent integer numbers having no fractional part
float(floating point)	stores real numbers in the range of about 3.4x10–38to 3.4x10 38,with a precision of seven digits.

double(Double floating point)	Stores real numbers in the range from 1.7x10 –308 to1.7x10308 with a precision of 15 digits.
bool(Boolean)	can have only two possible values: true and false.
Void	Valueless

C++ allows certain of the basic types to have modifiers preceding them. A modifier alters the meaning of the base type so that it more precisely fits the needs of various situations. The data type modifiers are: signed, unsigned, long and short

Type	Minimal Range
char	–127 to 127
unsigned char	0 to 255
signed char	–127 to 127
int	–32,767 to 32,767
unsigned int	0 to 65,535
signed int	Same as **int**
short int	–32,767 to 32,767
unsigned short int	0 to 65,535
signed short int	Same as **short int**
long int	–2,147,483,647 to 2,147,483,647
signed long int	Same as **long int**
unsigned long int	0 to 4,294,967,295
float	1E–37 to 1E+37, with six digits of precision
double	1E–37 to 1E+37, with ten digits of precision
long double	1E–37 to 1E+37, with ten digits of precision

This figure shows all combinations of the basic data types and modifiers along with their size and range for a 16-bit word machine

Variable

A variable is a named area in memory used to store values during program execution. Variables are run time entities. A variable has a symbolic name and can be given a variety of values. When a variable is given a value, that value is actually placed in the memory space assigned to the variable. All variables must be declared before they can be used. The general form of a declaration is:

type variable_list;

Here, type must be a valid data type plus any modifiers, and variable_list may consist of one or more identifier names separated by commas. Here are some declarations:

int i,j,l;

short int si;

unsigned int ui;

double balance, profit, loss;

Constants

Constants refer to fixed values that the program cannot alter. Constants can be of any of the basic data types. The way each constant is represented depends upon its type. Constants are also called literals. We can use keyword const prefix to declare constants with a specific type as follows:

const type variableName = value;

e.g,

const int LENGTH = 10;

Enumerated Types

An enumerated type declares an optional type name and a set of zero or more identifiers that can be used as values of the type. Each enumerator is a constant whose type is the enumeration. Creating an enumeration requires the use of the keyword enum. The general form of an enumeration type is:

enum enum-name { list of names } var-list;

Here, the enum-name is the enumeration's type name. The list of names is comma separated.

For example, the following code defines an enumeration of colors called color and the variable c of type color. Finally, c is assigned the value "blue".

enum color { red, green, blue } =c;

By default, the value of the first name is 0, the second name has the value 1 and the third has the value 2, and so on. But you can give a name, a specific value by adding an initializer. For example, in the following enumeration, green will have the value 5.

enum color { red, green=5, blue };

Here, blue will have a value of 6 because each name will be one greater than the one that precedes it.

Operator

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators. Generally, there are six type of operators: Arithmetical operators, Relational operators, Logical operators, Assignment operators, Conditional operators, Comma operator.

Arithmetical operators

Arithmetical operators +, -, *, /, and % are used to performs an arithmetic (numeric) operation.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

You can use the operators +, -, *, and / with both integral and floating-point data types. Modulus or remainder % operator is used only with the integral data type.

Relational operators

The relational operators are used to test the relation between two values. All relational operators are binary operators and therefore require two operands. A relational expression returns zero when the relation is false and a non-zero when it is true. The following table shows the relational operators.

Relational Operators	Meaning
<	Less than
<=	Less than or equal to
==	Equal to
>	Greater than
>=	Greater than or equal to
! =	Not equal to

Logical operators

The logical operators are used to combine one or more relational expression. The logical operators are

Operators	Meaning
||	OR
&&	AND
!	NOT

Assignment operator

The assignment operator '=' is used for assigning a variable to a value. This operator takes the expression on its right-hand-side and places it into the variable on its left-hand-side. For example:

m = 5;

The operator takes the expression on the right, 5, and stores it in the variable on the left, m.

x = y = z = 32;

This code stores the value 32 in each of the three variables x, y, and z. In addition to standard assignment operator shown above, C++ also support compound assignment operators.

Compound Assignment Operators

Operator	Example	Equivalent to
+ =	A + = 2	A = A + 2
- =	A - = 2	A = A – 2
% =	A % = 2	A = A % 2
/=	A/ = 2	A = A / 2
*=	A * = 2	A = A * 2

Increment and Decrement Operators

C++ provides two special operators viz '++' and '--' for incrementing and decrementing the value of a variable by 1. The increment/decrement operator can be used with any type of variable but it cannot be used with any constant. Increment and decrement operators each have two forms, pre and post.

The syntax of the increment operator is:
Pre-increment: ++variable
Post-increment: variable++

The syntax of the decrement operator is:

Pre-decrement: —variable
Post-decrement: variable—

In Prefix form first variable is first incremented/decremented, then evaluated
In Postfix form first variable is first evaluated, then incremented / decremented.

Conditional operator

The conditional operator ?: is called ternary operator as it requires three operands. The format of the conditional operator is :

Conditional_ expression ? expression1 : expression2;
If the value of conditional expression is true then the expression1 is evaluated, otherwise expression2 is evaluated.

int a = 5, b = 6;

big = (a > b) ? a : b;
The condition evaluates to false, therefore big gets the value from b and it becomes 6.

The comma operator

The comma operator gives left to right evaluation of expressions. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

int a = 1, b = 2, c = 3, i; // comma acts as separator, not as an operator

i = (a, b); // stores b into i would first assign the value of a to i, and then assign value of b to variable i. So, at the end, variable i would contain the value 2.

The sizeof operator

The sizeof operator can be used to find how many bytes are required for an object to store in memory. For example

sizeof (char) returns 1

sizeof (float) returns 4

Typecasting

Typecasting is the concept of converting the value of one type into another type. For example, you might have a float that you need to use in a function that requires an integer.

Implicit conversion

Almost every compiler makes use of what is called automatic typecasting. It automatically converts one type into another type. If the compiler converts a type it will normally give a warning. For example this warning: conversion from ‘double’ to ‘int’, possible loss of data. The problem with this is, that you get a warning (normally you want to compile without warnings and errors) and you are not in control. With control we mean, you did not decide to convert to another type, the compiler did. Also the possible loss of data could be unwanted.

Explicit conversion

The C++ language have ways to give you back control. This can be done with what is called an explicit conversion.

Four typecast operators

The C++ language has four typecast operators:

- static_cast
- reinterpret_cast
- const_cast
- dynamic_cast

Type Conversion

The Type Conversion is that which automatically converts the one data type into another but remember we can store a large data type into the other. For example we can't store a float into int because a float is greater than int.

When a user can convert the one data type into then it is called as the **type casting.** The type Conversion is performed by the compiler but a casting is done by the user for example converting a float into int. When we use the Type Conversion then it is called the promotion. In the type casting when we convert a large data type into another then it is called as the demotion. When we use the type casting then we can loss some data.