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.

Preview from Notesale.co.uk
Page 2 of 55

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**.**

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

Preview from Notesale.co.uk
Page 7 of 55

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.

Operators

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.

Preview from Notesale.co.uk
Page 8 of 55

Unit –II

Control Structures

Control structures allows to control the flow of program's execution based on certain conditions C++ supports following basic control structures:

1) Selection Control structure

2) Loop Control structure

1) Selection Control structure:

Selection Control structures allows to control the flow of program's execution depending upon the state of a particular condition being true or false .C++ supports two types of selection statements :if and switch. Condition operator (?:) can also be used as an alternative to the if statement.

If Statement:

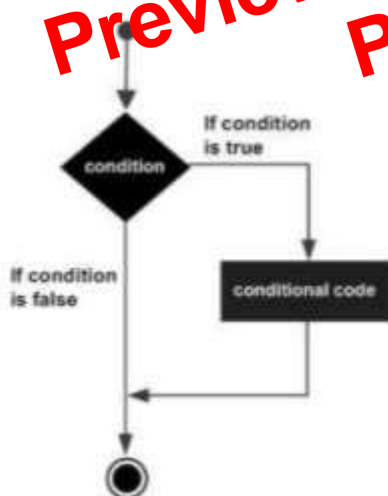
The syntax of an if statement in C++ is:

```
if(condition)

{

// statement(s) will execute if the condition is true

}
```

If the condition evaluates to true, then the block of code inside the if statement will be executed. If it evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Preview from Notesale.co.uk
Page 12 of 55

Flowchart showing working of if statement



```
int no, f;
cout<<"enter the positive number:-";
cin>>no;
f=factorial(no);                    //function call
cout<<"\nThe factorial of a number"<<no<<"is"<<f;
return 0;
}
int factorial(int n)         //function definition
{       int i , fact=1;
for(i=1;i<=n;i++){
        fact=fact*i;
}
return fact;
}
```

Inline Functions

An inline function is a function that is expanded inline at the point at which it is invoked, instead of actually being called. The reason that inline functions are an important addition to C++ is that they allow you to create very efficient code. Each time a normal function is called, a significant amount of overhead is generated by the calling and return mechanism. Typically, arguments are pushed onto the stack and various registers are saved when a function is called and then restored when the function returns. The trouble is that these instructions take time. However, when a function is expanded inline, none of those operations occur. Although expanding function calls in line can produce faster run times, it can also result in larger code size because of duplicated code. For this reason, it is best to inline only very small functions. inline is actually just a request, not a command, to the compiler. The compiler can choose to ignore it. Also, some compilers may not inline all types of functions. If a function cannot be inlined, it will simply be called as a normal function.

Preview from Notesale.co.uk
Page 20 of 55

A function can be defined as an inline function by prefixing the keyword inline to the function header as given below:

```
inline function header {
function body
}
```

```
// A program illustrating inline function
#include<iostream.h>
#include<conio.h>
inline int max(int x, int y){
if(x>y)
return x;
```

Unit-III

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. The general form of class declaration is:

```
class class-name {

access-specifier:

data and functions

access-specifier:

data and functions

// ...

access-specifier:

data and functions

} object-list;
```

The object-list is optional. If present, it declares objects of the class. Here, access-specifier is one of these three C++ keywords:

public   private   protected

Preview from Notesale.co.uk
Page 25 of 55

By default, functions and data declared within a class are private to that class and may be accessed only by other members of the class. The public access_specifier allows functions or data to be accessible to other parts of your program. The protected access_specifier is needed only when inheritance is involved.

Example:

```
#include<iostream.h>

#include<conio.h>

Class myclass {                // class declaration

// private members to myclass

int a;

public:

// public members to myclass

void set_a(intnum);

int get_a( );

};
```

```
int main(){
 A u;
 u.sum();
 getch();
 return(0);
}
```

Scope Resolution operator

Member functions can be defined within the class definition or separately using scope resolution operator (::). Defining a member function within the class definition declares the function inline, even if you do not use the inline specifier. Defining a member function using scope resolution operator uses following declaration

```
return-type class-name::func-name(parameter- list) {
// body of function
}
```

Here the class-name is the name of the class to which the function belongs. The scope resolution operator (::) tells the compiler that the function func-name belongs to the class class-name. That is, the scope of the function is restricted to the class-name specified.

```
Class myclass {
int a;
public:
void set_a(intnum); //member function declaration
int get_a( );                //member function declaration
};
//member function definition outside class using scope resolution operator
void myclass :: set_a(intnum)
{
a=num;
}
int myclass::get_a( ) {
return a;
}
```

Preview from Notesale.co.uk
Page 27 of 55

Another use of scope resolution operator is to allow access to the global version of a variable. In many situation, it happens that the name of global variable and the name of the local variable are same .In

```
}
```

Here, obj is a reference to an object that is being used to initialize another object. The keyword const is used because obj should not be changed.

Destructor

A destructor destroys an object after it is no longer in use. The destructor, like constructor, is a member function with the same name as the class name. But it will be preceded by the character Tilde (~).A destructor takes no arguments and has no return value. Each class has exactly one destructor. . If class definition does not explicitly include destructor, then the system automatically creates one by default. It will be invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer accessible.

```
// A Program showing working of constructor and destructor
#include<iostream.h>
#include<conio.h>
class Myclass{
public:
int x;
Myclass(){                    //Constructor
x=10; }
~Myclass(){                   //Destructor
cout<<"Destructing…." ;
}
int main(){
Myclass ob1, ob2;
cout<<ob1.x<<" "<<ob2.x;
return 0; }
```

Output:

10 10

Destructing……..

Destructing……..

Friend function

In general, only other members of a class have access to the private members of the class. However, it is possible to allow a nonmember function access to the private members of a class by declaring it as a friend of the class. To make a function a friend of a class, you include its prototype in the class declaration and precede it with the friend keyword. The function is declared with friend keyword. But while defining friend function, it does not use either keyword friend or :: operator. A function can be a friend of more than one class. Member function of one class can be friend functions of another class. In such cases they are defined using the scope resolution operator.

Preview from Notesale.co.uk
Page 31 of 55

```
 return 0;

}
```

It is critical to understand that when one class is a friend of another, it only has access to names defined within the other class. It does not inherit the other class. Specifically, the members of the first class do not become members of the friend class.

Preview from Notesale.co.uk
Page 43 of 55