FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

OBJECT ORIENTED PROGRAMMING THROUGH C++

I B.TECH II SEMESTER

(R16- Regulation)

Academic Year: 2018-2019

(COURSE MATERIAL)

Prepared By

Mrs. M R S Lavanya devi, M.Tech.

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

OBJECT-ORIENTED PROGRAMMING THROUGH C++

OBJECTIVES:

- This course is designed to provide a comprehensive study of the C programming language. It stresses the strengths of C, which provide students with the means of writing efficient, maintainable and portable code. The nature of C language is emphasized in the wide variety of examples and applications. To learn and acquire art of computer programming. To know about some popular programming languages and how to choose
- Programming language for solving a problem.

UNIT-I: Introduction to C++

Difference between C and C++- Evolution of C++- The Object Oriented Technology- Disadvantage of Conventional Programming- Key Concepts of Object Oriented Programming- Advantage of OOP- Object Oriented Language.

UNIT-II: Classes and Objects &Constructors and Destructor

Classes in C++-Declaring Objects- Access Specifiers and their Scope- Defining Member Function-Overloading Member Function- Nested class, Constructors and Destructors, Introduction- Constructors and Destructor- Characteristics of Constructor and Destructor- Application with Constructor- Constructor with Arguments (parameterized Constructor- Destructors- Anonymous Objects.

UNIT-III: Operator Overloading and Type Conversion & Inheritance

The Keyword Operator- Overloading Unary Operator- Operator Return Type- Overloading Assignment Operator (=)- Rules for Overloading Operators, Inheritance, Reusability- Types of Inheritance- Virtual Base Classes- Object as a Class Member- Abstract Classes- Advantages of Inheritance-Disadvantages of Inheritance,

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

UNIT-IV: Pointers & Binding Polymorphisms and Virtual Functions

Pointer, Features of Pointers- Pointer Declaration- Pointer to Class- Pointer Object- The this Pointer- Pointer to Derived Classes and Base Class, Binding Polymorphisms and Virtual Functions, Introduction- Binding in C++- Virtual Functions- Rules for Virtual Function- Virtual Destructor.

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

UNIT-V: Generic Programming with Templates & Exception Handling

Generic Programming with Templates, Need for Templates- Definition of class Templates- Normal Function Templates- Over Loading of Template Function- Bubble Sort Using Function Templates- Difference Between Templates and Macros- Linked Lists with Templates, Exception Handling- Principles of Exception Handling- The Keywords try throw and catch- Multiple Catch Statements –Specifying Exceptions.

UNIT-VI: Overview of Standard Template Library

Overview of Standard Template Library- STL Programming Model- Containers- Sequence Containers- Associative Containers- Algorithms- Iterators- Vectors- Lists- Maps.

OUTCOMES:

- Understand the basic terminology used in computer programming
- Write, compile and debug programs in C language. Use different data types in a computer program.
- Design programs involving decision structures, loops and functions.
- Explain the difference between call by value and call by reference

Text Books:

1. A First Book of C++, Gary Bronson, Cengage Learing.
2. The Complete Reference C++, Herbert Schildt, TMH.
3. Programming in C++, Ashok N Kamathane, Pearson 2nd Edition.

Reference Books:

1. Object Oriented Programming C++, Joyce Farrell, Cengage.
2. C++ Programming: from problem analysis to program design, DS Malik, Cengage Learing.

www.FirstRanker.com

Unit-1: Introduction to C++

Introduction to C++:

The C++ programming was developed by Bjarne StrousStrup in the year 1979 at AT &

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

T Bell laboratories in Murray Hill, New Jersey (USA) as part of his PhD thesis. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. He added new features taking from the language called "SIMULA 67" to the c language to enhance its performance, and originally named it as "C with classes", but later he added some features from the language called "ALGOL 68", it was renamed by Rick Mascitti as "C++" in the year 1983. The name C++, is thought as the C post increment operator ++. C++ is a superset of c, hence any legal c program can be executed in the C++.

SIMULA 67 C

Difference between C and C++:

- C is a Procedural Oriented Language C++ is Object Oriented language
- Data is not Protected in C. Data is Secured in C++
- It uses Top Down approach It uses Bottom Up Approach
- In C, same name cannot be given to two function With function overloading it is possible to give same name to two or more functions
- scanf() and printf() are used for reading and writing data respectively cin and cout are used for reading and writing data respectively
- C uses "stdio.h" header file for input & output operations C++ uses, "iostream.h" for the same purpose.
- C has no constructor & destructor C++ provides constructors and destructors 8 Inline functions can be used as macros Inline functions are supported by C++

The Object Oriented Technology

The nature is composed of various objects such as rocks, plants, trees, animals, and people etc. These objects can be further divided into two groups: living things and Non-living things. The rocks, chairs are the best example for Non-Living things which nothing but physical objects. The animals, trees, and people are the best example for Living things, which are again physical objects.

To understand the object oriented technology, let us take an analogy of the education institute which has two different working sections: Teaching and Non-Teaching. The institute will be having different departments such as CSE, ECE, MEC, EEE, CIVIL and LIBRARY. Each Department contains Teaching and Non-Teaching staff. Each department is considered as Object and is working for specific goals and objectives. Each department carries its own activities, and serves other departments when even needed. Departments can "communicate" with each other to carry out inter-departmental activities such as organizing the "workshops" and "conferences" though they are performing their own activities. The departments interact with other departments by sending messages. All these things can be represented in the C++. The departments can be represented as "class" from which different objects can be derived. The CSE Department is called an instance of the "department" which is called as Object. These objects contain their own data and methods to carry out their task and to communicate with other department whenever needed.

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

Disadvantages of Conventional programming

• Traditional programming languages such as C, COBOL, FORTRAN are called Procedure Oriented (POP) language and also called as Conventional Languages.

• The program written using these languages contains sequence of instructions that tell the compiler or interpreter to perform the task.

• When the program size is large, it is a bit difficult to manage and debug. To overcome this problem it is divided into smaller functions that carryout some specific task.

• Each function can call other function during the execution.

• As the functions are executing they may access same data, and may modify the data which in turn affects the entire task.

• Most of the functions are allowed to access the global variables.

• Large program is divided into smaller functions. These functions can call one another. Hence security is not provided.

• No importance is given to the data security.

• Data passes globally from one function to other function.

• Most function access global data.

Programming Paradigms

The change in the development of the software had taken place because of the 4 reasons:

1. The complexity of the problem domain
2. The difficulty of managing development process
3. The flexibility possible through the software
4. The problems of characterizing the behavior of discrete systems

To address different problems of the above, 4 programming paradigms have been designed.

1. Monolithic Programming
2. Procedural Programming
3. Structural Programming
4. Object Oriented Programming

www.FirstRanker.com

Monolithic Programming

• These programs contain global data and sequential code.

• Program flow control is achieved through Jump statements.

• There is no support of subroutine concept.

• It is suitable for small and simple applications.

• There is no support for data abstraction

• Examples: Assembly language and BASIC

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

• The programs are organized in the form of subroutines and all the data items are global

• These programs are very easy to understand and modify.

• These programs follow Top Down approach.

• These programs focus on the functions apart from data.

• Data is globally accessed to functions.

• Data is transferred by means of functions.

• It is difficult to implement simultaneous processes/parallelism

Examples: FORTRAN and COBOL

Structured Programming

• Programs are divided into individual procedures that perform specific task.

• Each procedure is independent of other procedure

• Procedures have their own local data and processing logic

• The projects can be broken up into modules; each module consists of different set of functions.

• Data access is controlled across the modules

• Examples: Pascal and C

• One of the main drawbacks of the procedure oriented programming is, it cannot provide data abstraction. It provides functional abstraction.

• To address the increasing complexity with software for the data abstraction Object oriented programming came into existence.

• Depending on the object features the languages are classified into two categories: o Object – Based Programming (OBP), example: JavaScript o Object-Oriented Programming(OOP), examples: Java and C++

• The OBP languages support Encapsulation, and object Identity, but do not support Inheritance and polymorphism.

• The OOP languages incorporate all the features of the OOP.

www.FirstRanker.com

Object Oriented Programming

There are several fundamental concepts in object oriented programming. They are shown as follow:

Definition: The process of binding Data and functions into a single unit is called Encapsulation.

• C++ supports the feature of Encapsulation using classes.

• The data in the class is not accessed by outside functions.

• The functions that are defined along with the data within the same class are allowed to access the data.

• Class defines the structure of data and member functions. The variables declared inside the class are called instance variables, and functions are called member functions.

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

• Objects are created from the class. An Object is specimen of a class.

• Class is logical structure, and object is physical structure.

• Each member in the class may be public or private.

• Only the member functions can access the private data of the class. However the public members can be accessed by the outside class.

Class declarations

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. A Class is used to pack the data and member functions together. For example:

class class_name {

type variable1; type variable2; type variable3;

type name_of_function(parameter_list); type name_of_function(parameter_list); };

Data abstraction Definition: It the process of providing essential features without providing the background or implementation details. Example:

It is not important for the user how TV is working internally, and different components are interconnected. The essential features required to the user are how to start, how to control volume, and change channels.

Inheritance Definition: It the process by which an object of one class acquires the properties of another class. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class. The actual power of the inheritance is that it permits the programmer to reuse the existing class to create new class.

Polymorphism Definition: polymorphism is a technique in which various forms of a single function can be defined and shared by different objects to perform an operation.

A function "display()" can display the shape of line, rectangle and square. Here the function "display()" has three different forms to perform the task of displaying the different things.

In OOP, two classes can be joined in two ways: (a) Inheritance (b) Delegation. In inheritance a new class can be derived from the existing class. The relationship between these two classes is called "Kind of relationship". For example class Y has been derived from class X, then class Y is called kind of X.

Definition:The second type of relations ship is " has a relationship". When object of one class is used as data member in another class, such composition of objects is known as "Delegation"

shape

display()

line

display()

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

class X//Base class

class Y//Derived class

rectangle

square

display()

display()

class A class B

class C { A a; //object of A B b; //Object of B }

Genericity

We can write a program that contains more than one version depending on the data types of the arguments provided at run time. For example we can write a program that can perform addition on integers, and the same program can also perform addition on floating point numbers. This feature allows declaration of variables without specifying exact data types. The compiler identifies the data at run time. The "template" feature of C++ allows this genericity.

Advantages of OOP OOP provides many advantages to the programmer and use. This solves many problems related to the software development, and provides improved quality and low cost.

1. These programs can be easily upgraded.
2. Using Inheritance, we can avoid writing the redundant code, and reuse already existing code.
3. It allows designing and developing safe programs using the data hiding.
4. Using the encapsulation feature of OOP, we can define new class with many functions and only few functions can be exposed to the user.
5. All the OOP language allows crating extended and reusable parts of the programs.
6. It changes the thinking of the programmer and results in rapid development of the software in a short time.
7. Objects communicate with each other and pass messages.

www.FirstRanker.com

The structure of C++ Program

C++ program consists of objects, classes, variables, functions and other program statements. It contains 4 main parts.

(1) The preprocessor directives

(2) class declaration or definition

(3) class function definition

(4) The main() function.

Preprocessing directive Class declaration or definition Class function definition The "main()"

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

function Fig 14. The parts of the C++ program

(1) Preprocessor Directives(Include header file section):

The preprocessor directive must be included at the beginning of program. It begins with symbol #(hash). It can be placed anywhere, but quite often it is placed at the beginning before the main() function. In traditional C, # must begin at the first column. Example:

#include<iostream>

(2) Declaration of the class:

Declaration of the class is done in this section. The class declaration starts with keyword "class" and contains some name followed by a pair of curly ({ }) braces with instance variable and member functions. The class ends with semicolon (;); Example:

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

#include<iostream> using namespace std;

class Box {

public: double l; double w; double b; void volume() { //body of the function } }; (3) Class functions declaration:

This part contains the definition of the functions. The definition can be also written outside the function with class name and scope operator before the function name. Example:

#include<iostream> using namespace std;

class Box {

public: double l; double w; double b; void volume(void); //function declaration }; //end of the class void Box:: volume() //definition outside the class {

// body of the function }

(4) The main() function:

Like C, C++ also starts with main function. The execution of every program starts with "main()" functions. The programming rules for C++ are same as C language.

Example:

#include<iostream> using namespace std;

class Box {

public: double l; double w; double b; void volume(void); //function declaration }; //end of the class

void Box:: volume() //definition outside the class {// body of the function

cout<< "The volume of Box is:"<<(l*w*b); } int main() {

Box b; b.l=12.3; b.w=13.4; b.b=14.5; b.volume(); return 0; }

Simple C++ program:

hello.cpp

```
int main()
{
//simple c++ program
cout<<"\n Hello! Welcome to C++ programming:";
return 0;
}
```

Compiling:

$>g++ hello.cpp ENTER //compiling $>./a.out // seeing the output

Explanation of above program:

• Program must be saved as ".cpp" as extension.

• Every program executes from main() function.

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

• A Comment can be placed in anywhere of the program. We can use single line comments (Begins with //), and multiple comment line start with /*Comment here.....*/

• Header files are needed at the beginning of the program.

• The statement cout<<"\n Hello! Welcome to C++ program:"; displays the string enclosed between the quotations. The "\n" is called escape sequence used to print the string in the next line.

• The << operator is called "insertion" operator.

Header files and Libraries

The library functions perform operations such as managing memory, reading and writing to disk files, input/output, mathematical operations, etc.

Introduction to Input and Output in C++

Generally computer applications use large amount of Data to be read from input devices and writing them to the output devices. To facilitate reading and writing operations C++ support number of inbuilt functions. These functions are stored in the library. The library is a set of .OBJ file that are linked during the execution of the program. This library is also called "iostream" library.

What is a stream? : stream is an intermediator between I/O devices and user. It is a flow of data, measured in bytes, in sequence. Input Stream: If the data is received from the input devices, it is called "source stream" and also called input stream. Examples for the input stream are Keyboard and disk. Output Stream: If the Data is written or passed to the output devices, then that stream is called "destination stream" or "output stream". Examples for the output device are Monitor and disk.

Predefined Streams

C++ has number of predefined streams. These streams are also called as Standard I/O Objects. These streams are automatically activated when the program execution starts.

stream Description

Cin standard input stream, usually the keyboard

Cout standard output stream, usually the monitor Cerr standard error (output) stream, usually the monitor Clog standard logging (output) stream, usually the monitor

Write an Example program to display a message using predefined objects

```
#include<iostream>
int main()
 {
cout<<"\\Hello displayed on monitor";
 cerr<<"Errors also printed to the monitor":
```


www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
clog<<"Displayed to the monitor";
return 0;
 }
```

Stream Classes C++ streams are based on classes and object theory. C++ has number of classes that work together with console and file operations. These classes are known as stream classes. All these classes are declared inside the header file "iostream.h". The istream and ostream are the derived classes of the "ios" base class. The ios contains member variable streambuf. The iostream is derived from istream and ostream classes using the multiple inheritance.

Formatted and Unformatted Data

Formatting means representation of data with different settings according to the user requirements. Various settings that can be done are number format, field width, decimal points, etc. The data accepted or printed with default setting of I/O functions of the language is known as "Unformatted Data". For example, when cin is executed is asks for the number and if the number is entered, then the cout will display it on the screen. By default the I/O functions represent the data in decimal format.

If the user wants to accept the data in the HEXADECIMAL format, manipulator with I/O functions should be used. The obtained data with these formats is called "Formatted Data". For example, hex is the manipulator.

cout<<hex<<15;

The above statement is converted to decimal to hexadecimal format.

Unformatted Console I/O Operations

Input and Output Streams

• The input stream uses "cin" object to read the data from the keyboard.

• The output stream uses "cout" object to write data to the screen.

• The "cin" and "cout" are called predefined streams for input and output data.

• The operator "<<" is used after the cout. It is called "insertion" operator.

• The operator ">>" is used before the variable name. It is called "extraction" operator.

• These two means "<<" and ">>" are called overloading operators.

Working of cin and cout statements Input stream The input stream reads the data using the cin object. The cin statement uses >> (extraction operator) before the variable name. The Syntax is as follows:

cin>>variable;

Example:

int v1; float v2;

cin >> v1>>v2; Where v1 and v2 are variables. If the user enters data 4 and 5, then 4 is stored in v1 and 5 is stored in v2. More than one variable can be used in cin statement then it is called "cascading input operations"

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

Output Streams The output streams manage the output of the streams. The cout object is used as output stream. It uses the << operator before the variable or string to display it on the screen. The Syntax is as follows:

cout<< variable; Example:

cout<<v1<<v2; or cout<<"Hello";

Write a program to accept the string from the keyboard and display it on the screen. Use cin and cout statements.

exp.cpp

```
#include<iostream.h>
using namespace std;
int main()
{
char name[20];
cout<<"Enter your name:";
cin>>name;
cout<<name;
return 0;
}
```

Write a program to read int, float, char and string using cout to display them on the screen.

```
#include<iostream.h>
using namespace std;
int main() {
char c; int a; float b; char city[10];
cout<<"Enter Character:\n";
cin>>c;
cout<<"Enter integer a\n";
cin>>a;
cout<<"Enter String :";
cin>>city;
cout<<" Char is:\n"<<c<<"The integer is :\n"<<a<<"The float is \n:"<<b<<"\n the city name is :"<<city;
return 0;
}
```

Write a C++ program to carry out all the arithmetic operations on integers.

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
#include<iostream.h>
using namespace std;
int main()
{
int a,b,c;
cout<<"Enter integer a and b\n";
 cin>>a>>b; c=a+b;
 cout<<"The sum is :"<<c;
 c=a*b;
cout<<"The Multiplications is :"<<c; c=a/b; cout<<"The Division is :"<<c;
 return 0;
 }
```

Type casting with cout statement

The type casting refers to the conversion of one basic type into another by applying external use data type keywords. C++ provides the type casting operator in the following format:

cout<<(type)variable; // here the type refers to the destination type and variable refers to the data of one basic type. Example:

float f=2.34; cout<<(int)<<f; Output: displays 2

Write a c++ program to use different formats of the type casting and display the converted value.

```
#include<iostream.h>
 using namespace std;
 int main()
{
int a=66; float f=2.34; double d=85.45; char c='A';
cout<<"\n int in char format:"<<(char)a;
cout<<"\n float in in t format:"<<(int)f;
cout<<"\n double in char format:"<<(char)d;
cout<<"\n char in int format:"<<(int)c;
return 0;
 }
```

Write a c++ program to display A to Z alphabets using the ASCII values

#include<iostream.h>

using namespace std;

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
int main()
{
int j;
 cout<<"\n the alphabets from A to Z are :\n";
 for (j=65;j<91;j++)
cout<<(char)j<<"\t";
 return 0;
 }
```

Note 1: The & operator is used to print the address of the operator. For example, int x=12; the variable is stored at some address of the memory. That address can be printed using the & operator before the variable name.

Example:

```
cout<<"The address of the variable is:"<<&x;
```

output :0x887ffff4, it is the hexadecimal format of the address.

Note 2: The & and * operators are used with string to display string. Example: char *name=" Ashok Kamthen";

```
cout<<name<<"\n"; cout<<&name[0]<<"\n";
```

Output statements will display the Same output. If we write only the &name, then it displays the address, if we use & along with the base address of the variable then it displays the string present at that address.

Member functions of the istream class

The istream class contains the following functions using cin object.

Function name Description cin.get() Used to read a character from the keyboard cin.peek() It returns the succeeding character without extraction cin.ignore(number, character) Used to ignore the maximum number of characters cin.putback() Replaces the given character into the input

stream cin.gcount() Return the number of characters extracted from the stream to a variable cin.getline() Contains the two arguments: variable and size , used to get the text from the variable with specified number of character in 2nd argument

Example programs: Write a c++ program to demonstrate the use of get(), peek() and ignore() functions.

```
#include<iostream.h>
using namespace std;
int main()
{
char c;
 cout<<"Enter the chars : And PRESS F6 to end";
```


www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
while(cin.get( c ))
{
cout<<c;
while(cin.peek()=='#')
{
        cin.ignore(1,'#');
 }
 }
return 0;
}
```

Write a c++ program to demonstrate the use of putback() function #include<iostream.h> #include<conio.h>

```
 int main()
{
 char c;
clrscr();
 cout<<"Enter the text and F6 to end:";
while(cin.get(c)) {
if(c=='s') cin.putback('S'); cout.put(c); } getch(); return 0;
}
```

Write a c++ program to demonstrate the use of gcount() function

```
#include<iostream.h>
 #include<conio.h>
 int main()
 {
char name[20]; int len;
clrscr();
cout<<"Enter text:";
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
 cin.getline(name,20);
 len= cin.gcount();
cout<<"The number of chars is:"<<len;
return 0;
}
```

Formatted Console I/O Operations

C++ provides various formatted console I/O functions for formatting the output. There are three types of Formatted I/O Functions:

1. IOS class functions flags 2. Manipulators 3. User defined output functions

IOS Functions

Sl No Function Description 1 width() Used to set the required field width 2 precision() Used to set the number of decimal points to a float value 3 fill() Used to set the character in the blank spaces 4 setf() Used to set various formatting flags 5 unsetf() Used to remove the flag setting Note: all these functions are used along with "cout" object. For example, cout.width(). The cout::width() function can be declared in two ways: i) int width()- used to return the current field width. ii) int width(int)- used to set the width with given integer.

The cout::precision() can be declared in two ways:

i) int precision() –used to return the current precision

ii) ii) int precision(int) –used to set the precision

The cout::fill() function is declared in two ways:

i) char fill()-used to return the current fill character ii) char fill(char) –used to set the filling character The cout::setf() function has the following form:

i) cout.setf(V1,V2) –where v1 is flag, and v2 is bit field ii) cout.unsetf() –used to clear the formatting flags

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

UNIT-II

CLASSES, OBJECTS, CONSTRUCTORS AND DESTRUCTORS

Class, defining a class in C++:

Object Oriented Programming encapsulates data (attributes) and functions (behavior) into packages called classes.

The class combines data and methods for manipulating that data into one package. An object is said to be an instance of a class. A class is a way to bind the data and its associated functions together. It allows the data to be hidden from external use. When defining a class, we are creating a new abstract data type that can be treated like any other built-in data type.

Generally, a class specification has two parts:

1. Class declaration 2. Class function definitions

The class declaration describes the type and scope of its members. The class function definitions describe how the class functions are implemented. The general form of a class declaration is:

class class_name

{ **private:**

variable declarations; function declarations;

public:

variable declarations; function declarations; };

www.FirstRanker.com

✓ The class declaration is similar to structure declaration in C. The keyword class is used to declare a class. The body of a class is enclosed within braces and terminated by a semicolon.

✓ The class body contains the declaration of variables and functions. These functions and variables are collectively called members. They are usually grouped under two sections i,e. private and public to denote which members are private and which are public. These keywords are known as visibility labels.

✓ The members that have been declared as private can be accessed only from within the class. On the other hand, public members can be accessed from outside the class also.

✓ The data hiding is the key feature of OOP. By default, the members are private. The

variables declared inside the class are known as data members and the functions are known as member functions.

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

✓ Only the member functions can have access to the private members and functions.

However, the public members can be accessed from outside the class.

✓ The binding of data and functions together into a single class type variable is

referred to as encapsulation.

A Simple Class Example:

A typical class declaration would look like:

class item

{

int no; // variables declaration

float cost; // private by default

public:

void getdata(int a, float b); // functions declaration void putdata(void); // using prototype };

declaring objects to a class:

Once a class has been declared, we can create variables of that type by using the class name. For example,

item x; // memory for x is created

Creates a variable x of type item.

In C++, the class variables are known as objects. Therefore, x is called an object of type item. We may also declare more than one object in one statement. Example:

item x, y, *z;

The declaration of an object is similar to that of a variable of any basic type. The necessary memory space is allocated to an object at this stage.

Note that class specification provides only a template and does not create any memory space for the objects.

How can you access the class members? The object can access the public class members of a class by using dot operator or arrow operator. The syntax is as follows;

Objectname operator membername; Example:

x.show(); z->show(); {z is a pointer to class item}

What is an access specifier? Explain about various access specifiers and their scope. The access specifier specifies the accessibility of the data members declared inside the class. They are: 1. public 2. private 3. protected

public keyword can be used to allow object to access the member variables of class directly. The keyword public followed by colon (:) means to indicate the data member and member function that visible outside the class. Consider the following example:

class Test

{

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

public:

int x, y; // variables declaration void show() {

cout<<x<<y; }

};

int main()

{

Test t; //Object creation

t.x = 10;

t.y = 20;

t.show();

};

Now, it display 10 and 20

Private keyword is used to prevent direct access to member variables or functions by the object. It is the default access. The keyword private followed by colon (:) is used to make data member and member function visible with in the class only.

Consider the following example: class Test {

private:

int x, y; // variables declaration };

int main() {

Test t; //Object creation t.x = 10; t.y = 20; };

Now it raises two common compile time errors:

„Test::x" is not accessible „Test::y" is not accessible

Protected access is the mechanism same as private. It is frequently used in inheritance. Private members are not inherited at any case whereas protected members are inherited. The keyword private followed by colon (:) is used to make data member and member function visible with in the class and its child classes.

www.FirstRanker.com

Member functions

Member functions are defined in two places. They are;

1. Inside the class 2. Outside the class

Inside the class Member functions can be defined immediately after the declaration inside the class. These functions by default act as inline functions.

Example:

class student

{

private:

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
int rno; char *sname; public:
 void print()
{
cout<<"rno="<<rno;
cout<<"name="<<sname;
 }
 void read(int a, char *s)
{
rno = a;
strcpy(snm, s);
 }
 };
void main()
{
student s; s.read(10, "Rama");
 s.print();
}
```

Outside the class

✓ Member functions that are declared inside a class have to be defined separately outside the class. Their definitions are very much like the normal functions. ✓ The main difference is the member function incorporates a membership identity label in the header. This label tells the compiler which class the function belongs to. General form

return-type classname :: function-name (list of arguments) {
// function body } Example: class student {
private:
int rno; char *sname; public:
void read(int , char*); void print(); }; void student :: read (int a, char *s) {
no = a; strcpy(sname, s); } void student :: print() {
cout<<"rno="<<rno; cout<<"name="<<sname; } void main() {
student s; s.read(10, "Rama"); s.print(); }

4 Output: rno= 10 sname = Rama

Output: rno= 10 sname = Rama

characteristics of member functions

✓ The member functions are accessed only by using the object of the same class. ✓ The same

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

function can be used in any number of classes. ✓ The private data or private functions can be accessed only by public member

functions.

How can you declare outside member function as inline? Explain with an example. (OR) Explain about inline keyword.

✓ Inline function mechanism is useful for small functions. ✓ The inline functions are similar to macros. By default, all the member functions

defined inside the class are inline functions. ✓ The member functions defined outside the class can be made inline by prefixing

"inline" to the function declaration.

General form:

inline Return-type classname :: function-name (list of arguments) {

//function body } Example: class student {

private:

int rno; char *sname;

public:

void print();

void read(int a, char *s)

{

no = a; strcpy(snm, s);

}

};

inline void student :: print()

{

cout<<"no="<<no; cout<<"name="<<snm;

}

void main()

{

Output: student s;

rno= 10 s.read(10, "Rama");

sname = Rama s.print(); }

Data hiding

Data hiding is also called as data encapsulation. An encapsulated object is called as abstract data type. Data hiding is useful for protecting data. It is achieved by making the data variables of class as private. Private variables cannot directly accessible to the outside of the class. The

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

keywords private and protected are used to protect the data.

Access Specifier Members of the Class Class Objects Public Yes Yes Private Yes No Protected Yes No

Example:

```
class ex
{
private:
        int p; protected:
        int q;
public:
        int r; void getp()
        {
            p=10; cout<<"private="<<p;
        }
        void getq()
        {
                q=10; cout<<"protected="<<q;
        }
        void getr()
        {
            r=10; cout<<"public="<<r;
        }
};

void main()
{
ex e;
e.getp();
e.getq();
e.getr();
 e.r=100;
 e.getr();
}
```

www.FirstRanker.com

Overloading a member function

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

Member functions can be overloaded like any other normal functions. Overloading means one function is defined with multiple definitions with same functions name in the same scope. The following program explains the overloaded member functions

```
class absv
{
 public:
      int num(int i);
       double num(double d);
 };
int absv:: num(int i)
 Output: private =10 protected=10 public =10 public =100
{
      return (abs(i));
}
double absv:: num(double d)
 {
      return (fabs(d));
}
void main()
 {
     absv a;
   cout<<"\nThe absolute value of -10 is "<<n.num(-10);
   cout<<"\nThe absolute value of -12.35 is "<<n.num(-12.35);
}
```

Output: The absolute value of -10 is 10 The absolute value of -12.35 is 12.35

nested class with an example.

When a class defined in another class, it is known as nesting of classes. In nested classes, the scope of inner class is restricted by outer class

The following program illustrates the nested classes:

```
class A
{ public :
  class B
  {
    void show()
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
    {
      cout<<"\nC++ is wonderful language";
    }
 };
 };
int main(void)
{
A::B x;
x.show();
}
```

constructor and destructor

A constructor is a special member function used for automatic initialization of an object. Whenever an object is created, the constructor is called automatically. Constructors can be overloaded. Destructor destroys the object. Constructors and destructors having the same name as class, but destructor is preceded by a tilde (~) operator. The destructor is automatically executed whenever the object goes out of scope.

Characteristics of Constructors

✓ Constructors have the same name as that of the class they belongs to.

✓ Constructors must be declared in public section.

✓ They automatically execute whenever an object is created.

✓ Constructors will not have any return type even void

✓ Constructors will not return any values.

✓ The main function of constructor is to initialize objects and allocation of memory to the objects.

✓ Constructors can be called explicitly.

✓ Constructors can be overloaded.

✓ A constructor without any arguments is called as default constructor.

Applications of constructors:

Default constructor :

Constructors are used to initialize member variables of a class. Constructors allocate required memory to the objects. Constructors are called automatically whenever an object is created. A constructor which is not having any arguments are said to be default constructor.

Example: class num {

int a, b; public:

num() {

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

a=5; b=2; } void show() {

cout<<a<<b; } }; Void main() {

num n; n.show(); }

parameterized constructors with an example.

✓ It may be necessary to initialize the various data elements of different objects with different values when they are created. This is achieved by passing arguments to the constructor function when the objects are created.

✓ The constructors that can take arguments are called parameterized constructors.

✓ They can be called explicitly or implicitly.

Example: class num

```
 {
    int a, b ,c;
     public:
num(int x, int y)
{
          a=x; b=y;
           }
           void show()
          {
          cout<<a<<b;
           }
};
void main()
 {
num n(10,20); //implicit call
 n.show();
 num x= num(1,2); //explicit call
x.show();
}
```

www.FirstRanker.com

constructor overloading(Multiple Constructors)

✓ Similar to normal functions, constructors also overloaded. C + + permits to use more than one constructors in a single class.

✓ If a class contains more than one constructor. This is known as constructor overloading.

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

Add( ) ; // No arguments Add (int, int) ; // Two arguments

Example: class num {

int a, b; public:

num() // Default constructor {

a=10; b=20; } num(int x, int y) // Parameterized constructor

{

a=x; b=y; } void add() {

cout<<a+b; } }; void main() {

num n; n.add(); num x(1,2); x.add();

}

The num class has two constructors. They are;

✓ Default constructor ✓ Parameterized constructor

Whenever the object "n" is created the default constructor is executed automatically and a, and b values are initialized to 10, 20 respectively.

Whenever the object "x" is created the parameterized constructor is executed automatically and a, and b values are assigned with 1 and 2 respectively.

constructors with default argument with an example.

✓ Similar to functions, It is possible to define constructors with default arguments. ✓ Consider power(int n, int p= 2);

– The default value of the argument p is two. – power p1 (5) assigns the value 5 to n and 2 to p. – power p2(2,3) assigns the value 2 to n and 3 to p.

Example: class power {

int b,p; public:

power(int n=2, int m=3) {

b=n; p=m; cout<<pow(n,m); } }; void main() {

power x; Power y(5); power z(3,4);

}

copy constructor with an example.

✓ Copy constructor is used to declare and initialize an object from another object. ✓ A copy constructor takes a reference to an object of the same class as itself as an

argument.

Ex: class Test

{

int i; public:

Test() // Default constructor

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
        {
            i=0;
        }
    Test (int a) // Parameterized constructor
      {
          i = a;
      }
    Test (code &x) //Copy Constructor
     {
         i = x.i;
     }
    void show()
     {

cout<<i<<endl;
 }
};
void main()
 {
Test a(100);
a.show();
Test b(a); //Copy Constructor invoked
 b.show();
}
```

www.FirstRanker.com

Destructors with an example.

✓ Destructor is a special member function like a constructor. Destructors destroy the class objects that are created by constructors.

✓ The destructor have the same name as their class, preceded by a ~.

✓ The destructor neither requires any arguments nor returns any values

.✓ It is automatically executed when the object goes out of score

✓ Destructor releases memory space occupied by the objects.

Characteristics of Destructors

✓ Their name is the same as the class name but is preceded by a tilde(~).

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

✓ They do not have return types, not even void and they cannot return values.

✓ Only one destructor can be defined in the class.

✓ Destructor neither has default values nor can be overloaded.

✓ We cannot refer to their addresses.

✓ An object with a constructor or destructor cannot be used as a member of a union.

✓ They make „implicit calls" to the operators new and delete when memory allocation/ memory de-allocation is required.

Example:

```
class Test
{
public:
Test()
{
        cout<<"\n Constructor Called";
 }
~Test()
 {
        cout<<"\n Destructor Called";
}
};
void main()
{
        Test t;
 }
```

www.FirstRanker.com

What is meant by anonymous objects? Explain. It is possible to declare objects without any name. These objects are said to be anonymous objects. Constructors and destructors are called automatically whenever an object is created and destroyed respectively. The anonymous objects are used to carry out these operations without object.

Output: Constructor Called
Destructor Called

```
class noname
 {
   int x;
   public:
```


www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
    noname()
    {
      cout<<"\n In Default Constructor"; x=10; cout<<x;
    }
 noname(int i)
 {
          cout<<"\n In Parameterized Constructor"; x=i; cout<<x;
  }
  ~noname()
 {
         cout <<"\n In Destructor";
 }
};
void main()
{
noname();
noname(12);
 }
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

UNIT-III

OPERATOR OVERLOADING AND INHERITANCE

Operator overloading:

A symbol that is used to perform an operation is called an operator. It is used to perform operations on constants and variables. By using operator overloading, these operations are performed on objects. It is a type of polymorphism. The keyword operator is used to define a new operation for an operator.

Syntax:

return-type operator operator-symbol (list of arguments) {

// Set of statments } Steps:

1. Define a class which is to be used with overloading operators. 2. Declare the operator prototype function is in public section. 3. Define the definition of the operator.

Example # include <iostream.h> # include <conio.h> class number {

public:

int x,y; number() {

x=0; y=0; } number(int a, int b) {

x=a; y=b; } number operator + (number d) {

number t; t.x=x+d.x; t.y=y+d.y; return t; } void show() {

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
        cout<<"\nX="<<x<<"\t Y="<<y; } };
int main() {
number n1(10,20), n2(20,30), n3; n3=n1+n2; n3.show(); return 0; }
```

Output X=10 Y=20 X=20 Y=30 X=30 Y=40

overloading unary operators with an example.

• An operator which takes only one argument is called as unary operator. Ex: ++, --, -, +, !, ~, etc.

• A class member function will take zero arguments for overloading unary operator.

• A friend member function will take only one argument for overloading unary operator. Constraints on increment /decrement operators When ++ and − − operators are overloaded, there exists no difference between the postfix and prefix overloaded operator functions. To make the distinction between prefix and postfix notations of operator, a new syntax is used to indicate postfix operator overloading function. The syntaxes are as follows:

Operator ++(int); //postfix notation Operator ++( ); //prefix notation

Ex: Program for overloading unary operator using normal member function.

```
# include <iostream.h>
# include <conio.h>
class num
{
private:
    int a, b;
    public:
num(int x, int y)
 {
a=x; b=y;
}
void show()
 {
        cout<<"\nA="<<a<<"\tB="<<b;
 }
 void operator ++( ) //prefix notation
{
++a; ++b;
}
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
void operator --(int) //postfix notation
{   a--;
    b--;
}
};
int main()
{
num x(4,10);
x.show();
 ++x;
cout<<"\n After increment";
 x.show(); x--;
cout<<"\n After decrement";
 x.show();
}
```

Output After increment A = 5 B = 11 After decrement A = 4 B = 10

Ex: Program for overloading unary operator using friend function.

```
# include <iostream.h>
# include <conio.h>
class complex
{
private:
      int real, imag; public:
complex() {
          real=imag=0; } complex(int r, int i) {
real=r; imag=i; } void show() {
          cout<<"\n real="<<real<<"\timaginary="<<imag; } friend complex operator -(complex
&c) {
c.r=-c.r; c.i=-c.i; return c; } }; void main() {
complex c(1,-2); c.show(); cout<<"\nAfter sign change"; -c; c.show(); }
```

Output:

real= 1 imaginary=-2

overloading binary operators with example:

• An operator which takes two arguments is called as binary operator. Ex: +, *, /, etc.

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

• A class member function will take one argument for overloading binary operator.

• A friend function will takes two arguments for overloading binary operator. Example: Program for overloading binary operator using class member function. # include <conio.h> # include <iostream.h>

```
class num
{
private:
  int a , b;
  public:
void input()
{
cout<<"\nEnter two numbers:";
cin>>a>>b;
}
void show()
 {
cout<<"\nA= "<<a<<"\tB= "<<b;
 }
num operator +(num n)
{
num t; t.a=a+n.a; t.b=b+n.b; return t;
 }
 num operator -(num n)
{
num t; t.a=a-n.a; t.b=b-n.b; return t;
}
 };
nt main()
{
num x,y,z;
x.read();
 y.read();
 z=x+y; r.show();
return 0;
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
}
```

Output Enter two numbers: 1 2 Enter two numbers: 3 4 A=4 B=6

Program for overloading binary operator using friend function.

```
# include <conio.h>
# include <iostream.h>
class num
{
private:
     int a, b; public:
void input()
{
cout<<"\nEnter two numbers:";
cin>>a>>b;
 }
 void show()
{
          cout<<"\nA="<<a<<"\tB="<<b;
}
friend num operator + (num o1, num o2)
{
num t; t.a=o1.a+o2.a; t.b=o1.b+o2.b;
return t;
 }
};
int main()
 {
num x,y,z;
x.input();
y.input();
 z=x+y;
z.show();
 return 0;
 }
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

Output Enter two numbers: 5 8 Enter two numbers: 1 4 A=6 B=12

Explain about rules for overloading operators.

1. Operator overloading can't change the basic idea. 2. Operator overloading never changes its natural meaning. An overloaded operator "+" can be used for subtraction of two objects, but this type of code decreased the utility of the program. 3. Only existing operators can be overloaded. 4. The following operators can't be overloaded with class member functions

Operator Description

. Member operator .* Pointer to member operator :: Scope resolution operator sizeof() Size of operator # and ## Preprocessor symbols

5. The following operators can't be overloaded using friend functions.

Operator Description ( ) Function call operator = Assignment operator [ ] Subscripting operator –> Class member access operator

6. In case of unary operators normal member function requires no parameters and friend

function requires one argument. 7. In case of binary operators normal member function requires one argument and friend

function requires two arguments. 8. Operator overloading is applicable only within in the scope. 9. There is no limit for the number of overloading for any operation. 10. Overloaded operators have the same syntax as the original operator.

Inheritance:

Inheritance is the most important and useful feature of OOP. Reusability can achieved with the help of inheritance. The mechanism of deriving new class from an old class is called as inheritance. The old class is known as parent class or base class. The new one is called as child class or derived class. In addition to that properties new features can also be added.

www.FirstRanker.com

Advantages:

• Code can be reused.

• The derived class can also extend the properties of base class to generate more dominant objects.

• The same base class is used for more derived classes.

• When a class is derived from more than one class, the derived classes have similar properties to those of base classes.

Disadvantages:

• Complicated.

• Invoking member functions creates overhead to the compiler.

• In class hierarchy, various data elements remains unused, and the memory allocated to them is not utilized.

What are access specifiers? How class are inherited? C++ provides three different access specifiers. They are:

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

1. Public 2. private and 3. protected.

• The public data members can be accessed directly outside of the class with the object.

• The private members are accessed by the public member functions of the class.

• The protected members are same as private but only the difference is protected members are inherited while private members are not inherited.

A new class can be derived from the old one is as follows:

class name_of_the_derived_class : access_specifier name_of_the_base_class {

..... Members of the derived class ..... };

Example: class A : public B {

.... };

class A : private B { ....

}; class A : protected B { .... };

Note: If no access specifier is specified then by default it will takes as private.

Public de rivation: In public derivation, all the public members of base class become public members of the derived class and protected members of the base class becomes protected members to the derived class. Private members of the base class will not be inherited.

Example: # include <iostream.h>

```
class Base {
        public: int x;
         };
class Derived : public Base
      {
        public: int y;
        };
        int main()
         {
Derived d;
 d.x=10;
 d.y=20;
cout<<"\n Member x="<<b.x;
 cout<<"\n Member y="<<b.y;
return 0;
}
```

Output: Member x=10 Member y=20

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

Private derivation

In private derivation, all the public and protected members of the base class become private members of the derived class and private members of the base class will not be inherited.

Example: # include <iostream.h>

```
class Base
{
public: int x;
};
class Derived : private Base
{
        int y; public:
        Derived( )
        {
                x=10;
                y=20;
        }
        void show()
        {
                cout<<"\n Member x="<<x; cout<<"\n member y="<<y;
        }
};
int main()
{
Derived d;
d.show():
return 0;
}
```

Output: Member x=10 Member y=20

Protected derivation In protected derivation, all the public and protected members of the base class become protected members of the derived class and private members of the base class will not be inherited.

Example: # include <iostream.h>

```
class Base
{
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
        public: int x;
};
class Derived : protected Base
{
    int y;
    public:
    Derived( )
   {
        x=10; y=20;
 }
void show( )
{
cout<<"\n Member x="<<x; cout<<"\n member y="<<y;
 }
};
int main()
 {
Derived d;
d.show();
 }
```

Output: Member x=10 Member y=20

types of inheritance with examples

1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance
5. Hybrid Inheritance
6. Multipath Inheritance

Single Inheritance: In this type of inheritance one derived class inherits from only one base class. It is the simplest form of Inheritance..

Here A is the base class and B is the derived class.

Example: #include<iostream.h>

```
 #include<conio.h>
class Emp
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
{
public:
int eno; char ename[20],desig[20];
void input()
{
cout<<"Enter employee no:"; cin>>eno;
 cout<<"Enter employee name:";
cin>>ename; cout<<"Enter designation:";
cin>>desig;
}
};
class Salary: public Emp
{
  float bp, hra, da, pf, np; public:
void input1()
 {
cout<<"Enter Basic pay:";
cin>>bp;
            cout<<"Enter House Rent Allowance: ";
cin>>hra; cout<<"Enter Dearness Allowance :";
cin>>da; cout<<"Enter Provident Fund:"; cin>>pf; }
void calculate()
 {
        np=bp+hra+da-pf;
}
        void display()
{
cout<<"\nEmp no: "<<eno cout<<"\nEmp name: "<<ename;
cout<<"\nDesignation: "<<design;
cout<<"\nBasic pay:"<<bp;
cout<<"\nHRA:"<<hra;
cout<<"\nDA:"<<da;
cout<<"\nPF:"<<pf;
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
cout<<"\nNet pay:"<<np;
}
};
int main()
{
clrscr();
 Salary s;
s.input();
 s.input1();
 s.calculate();
 s.show();
getch();
return 0;
 }
```

Multiple Inheritance:

In this type of inheritance a class may derive from two or more base classes.

(or) When a class is derived from more than one base class, is called as multiple inheritance.

Where class A and B are Base classes and C is derived class.

```
Example: #include<iostream.h>
 #include<conio.h>
class Student
{
protected:
   int rno,m1,m2;
   public:
void input()
{
cout<<"Enter the Roll no :";
cin>>rno;
 cout<<"Enter the two subject marks :"; cin>>m1>>m2;
 } };
class Sports
{
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
protected:
    int sm; // sm = Sports mark public:
 void getsm()
 {
cout<<"\nEnter the sports mark :";
 cin>>sm;
}
};
class Report : public student, public sports
{
    int tot, avg;
     public:
   void show()
  {
    tot=(m1+m2+sm);
avg=tot/3;
 cout<<"\nRoll No : "<<rno<<"\nTotal : "<<tot; cout<<"\n\tAverage : "<<avg;
}
};
int main()
 {clrscr();
 Report r;
 r.input();
 r.getsm();
 r.show();
return 0;
 }
```

Hierarchical Inheritance In this type of inheritance, multiple classes are derived from a single base class.

Where class A is the base class and B, C and D are derived classes. Example:

```
#include<iostream.h>
 #include<conio.h>
  class Vehicle
 {
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
public:
Vehicle()
{
    cout<<"\nIt is motor vehicle";
 }
 };
class TwoWheelers : public Vehicle
{
public:
TwoWheelers()
{
    cout<<"\nIt has two wheels";
 }
void speed()
 {
cout<<"\nSpeed: 80 kmph";
 }
};
class ThreeWheelers : public Vehicle
{
public:
ThreeWheelers()
 {
    cout<<"\nIt has three wheels";
}
void speed()
 {
cout<<"\nSpeed: 60 kmph";
}
};
class FourWheelers : public Vehicle
{
public:
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
FourWheelers()
{
    cout<<"\nIt has four wheels";
}
void speed()
{
   cout<<"\nSpeed: 120 kmph";
}
};
int main( )
 {
clrscr();
TwoWheelers two;
two.speed();
 cout<<"\n----------"; ThreeWheelers three; three.speed();
cout<<"\nSpeed: 90 kmph";
 }
};
class Maruti800 : public Maruti
 {
 public Maruti800()
  {
    cout<<"\nModel: Maruti 800");
}
 void speed()
{
       cout<<"\nSpeed: 120 kmph";
 }
 };
int main( )
{
clrscr();
Maruti800 m;
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
 m.speed();
getch();
 }
```

Hybrid (Virtual) Inheritance

Hybrid Inheritance is combination of one or more types of inheritance.

Where class A to class B forms single inheritance, and class B,C to class D form Multiple inheritance.

Example:

```
#include<iostream.h>
#include<conio.h>
 class Player
 {
protected:
char name[20]; char gender; int age;
};
class Physique : public Player
{protected:
        float height,weight;
 };
class Location
{
protected:
char city[15];
 long int pin;
};
class Game : public Physique, public Location
 {
char game[15];
public: void input()
{
cout<<"\nEnter Player Information";
cout<<"Name: ";
cin>>name;
 cout<<"Genger: ";
 cin>>gender;
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
cout<<"Age: ";
cin>>age;
 cout<<"Height and Weight: ";
 cin>>heigh>>weight;
 cout<<"City: ";
 cin>>city;
cout<<"Pincode: ";
 cin>>pin;
cout<<"Game played: "; cin>>game;
}
void input()
 {
cout<<"\nPlayer Information";
cout<<"\nName: "<<name;
 cout<<"\nGenger: "<<gender; cout<<"\nAge: "<<age; cout<<"\nHeight<<height;
cout<<"\nWeight: "<<weight; cout<<"\nCity: "<<city;
cout<<"\nPincode: "<<pin; cout<<"\nGame: "<<game;
 }
 };
 int main( )
{  clrscr();
Game g;
g.input();
 g.show();
return 0;
 }
```

www.FirstRanker.com

Multi-path Inheritance

In this, one class is derived from two base classes and in turn these two classes are derived from a single base class in known as Multi-path Inheritance.

```
Example class A
 {
        //class A definition }; class B: public A
        {
        //class B definition }; class C: public A
```


www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
 {
//class C definition }; class D :public B, public C
 {
//class D definition
};
```

Virtual base classes with example.

How can you overcome the ambiguity occurring due to multipath inheritance? Explain with an example. To overcome the ambiguity due to multipath inheritance the keyword virtual is used. When classes are derived as virtual, the compiler takes essential caution to avoid the duplication of members.

Uses: When two or more classes are derived from a common base class, we can prevent multiple copies of the base class in the derived classes are done by using virtual keyword. This can be achieved by preceding the keyword "virtual" to the base class.

Example #include<iostream.h>

```
#include<conio.h>
class A
{
protected:
int a1;
};
class B: public virtual A
 {
protected:
int a2;
};
class C: public virtual A
{
protected:
int a3;
};
class D :public B, public C
{
    int a4;
     public:
void input()
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
{
cout<<"Enter a1,a2,a3 and a4 values:";
cin>> a1>>a2>>a3>>a4;
}
void show()
{
cout<<"a1="<<a1<<"\na2="<<a2;
cout<<"\na3="<<a3<<"\na4="<<a4;
 }
};
int main()
 {
D d;
d.input();
 d.show();
 return 0;
 }
```

Constructors and destructors are executed in inherited class:

The constructors are used to initialize the member variables and the destructors are used to destroy the object. The compiler automatically invokes the constructor and destructors.

Rules:

• The derived class does not require a constructor if the base class contains default constructor.

• If the base class is having a parameterized constructor, then it is necessary to declare a constructor in derived class also. The derived class constructor passes arguments to the base class constructor.

Example:

```
#include<iostream.h>
 class A
{
public: A()
{
        cout<<"\n Class A constructor called";
}
~A()
{
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

```
        cout<<"\nClass A destructor called";
} };
class B : public A
{
public: B()
 {
cout<<"\n Class B constructor called";
}
~B()
 {
        cout<<"\nClass B destructor called";
 }
};
class C : public B
{
public: C()
 {
        cout<<"\n Class C constructor called";
 }
~C()
{
        cout<<"\nClass C destructor called";
 }
};
int main()
{
C c;
return 0;
}
```

www.FirstRanker.com

www.FirstRanker.com

FirstRanker.com
Firstranker's choice
www.FirstRanker.com www.FirstRanker.com

FREQUENTLY ASKED QUESTIONS FOR UNIT- I, II & III

UNIT-I

1. Write about the features of Object Oriented Programming Language when compared with conventional programming.
2. Explain the differences between c and c++.
3. Write about the key concepts of object oriented programming.
4. Write about advantages of oop.

UNIT-II

1. Define a Class in C++ and explain the method of defining a class with example.
2. Explain about different access specifiers.
3. Explain the process of method overloading with an example.
4. Write about constructors and destructors.

UNIT-III

1. Write about Operator Overloading.
2. Explain unary operator overloading with an example program.
3. Explain Binary Operator Overloading with an example program.
4. Explain different types of inheritance with examples.

www.FirstRanker.com

www.FirstRanker.com