1 Introduction

Contents

Overview of C++, Sample C++ program, Different data types, operators, expressions, and statements, arrays and strings, pointers & function components, recursive functions, user - defined types, function overloading, inline functions, Classes & Objects - I: classes, Scope resolution operator, passing objects as arguments, returning objects, and object assignment.

POINTS TO REMEMBER

- Object Oriented programming is way to organizing programs.
- The key elements of oop are encapusalation, inheritance and polymorphism.
- Procedure programms employs top down approach.
- OOP over comes all the problems of procedural programming.
- OOP employs bottom up approach.
- The building blocks of oop are classes and objects.
- The real application of oop are in the areas where problems are large and compiler.
- The array is a data structure to store collections of similar values under a given name.
- Elements of an array are stored in contigous memory locations.
- Elements of an array are accessed by name of the array followed by index within brackets. One pair of brackets is used for each dimension.
- Array must be declared and defined before use.
- Arrays can be initialized at the time of declaration and defination.
- C++ does not provide the boundary check on the elements of an array.
- When an array is partially initialized the rest of the elements are set to zero.
- Two dimensional array is a representation of table with rows and columns.
- C++ language does not support data type however strings are handled as an array of characters.
- C++ language uses null terminated variable length strings.
- A string constant in C++ language is a sequence of character enclosed in double quotes.
- The difference between character and string is that the later one is a data type while former one is data structure. Strings uses array of character as its basic type.
- C++ provide a rich library of functions for manipulating strings.
- List of strings are handled by using two dimensional array of characters.
- There is rich library functions for manipulating strings.
- A structure is a collection of related elements, which can be of difference types having a single name.
- Each element of the structure is called the field.
- There are two ways to declare a structuer : tagged structure and type defined structure.

- We can access the field of the structure using direct selection, dot operator (.).
- Structure can be nested. In this case one dot operator is used for each level of nesting.
- At a given time, only one element of a union variable can be assigned a value.
- Bit fields can be used in structures to save memory.
- When C++ program runs, the memory is divided into several different section/area. These sections are : code area, data area, stack area and heap.
- Pointer is a variable that holds an address of an operand.
- Memory allocation is done in two ways statically and dynamically.
- Static memory allocation is that which is done at compilation time.
- Dynamic memory allocation is that which is done at compilation time.
- Pointer are used to pass arguments by addresses.
- A function can return pointer.
- In an algorithm the steps are numbered and these steps are executed in their increasing order.
- Flow chart shows the order of execution of various steps.
- In a pseudocode, the steps are executed in sequence as they appear except in case of a decision or a loop.
- Object oriented programming is a way of organising programs.
- The keys of oop are encapsulation, inheritance and polymorphism.
- Object oriented programming overcomes all the problem of procedural programming approach.
- OOP employs bottom up approach.
- OOP helps to solve large and complex problems.
- C++ language is built on the top of C language.
- C++ language is middle level language and is portable.
- Different stages of program development are creation, compilation, linking, testing and documenting.
- Building block of C++ language are character, tokens, identifier, expressions and statements.
- Data type is a set of values along with set of permissible operations.
- Simple data type consist of integers (short, int, long) (char)

3. Executing a segment repeatedly either for a given number of times or till some conditions are met.

☞ Different functions can be stored or separate files. In this approach you need to compile these files separately and then link them together. To automate this process you can use turbo C++ compiler project facility.

☞ Storage class determine the life time visibility of the variables. It also determine where the variable will be stored. What will be their initial value if they are not intialized.

☞ The different storage classes are auto, static, register and extern.

QUESTION-ANSWERS

Q 1. How is heap memory allocated in C++? **(PTU, May 2009)**

Ans. Every program is provided with a pool of unallocated memory that it can utilize during execution. This pool of unallocated memory is known as heap. For allocation new operator is used. The new operator allocates the memory and returns a pointer to an appropriate type. The new operator is defined as

```
type * new type [size in integer] ;
e.g.  int *n1 ;
      n1 = new int [100] ;
```

It allocates a memory block of 200 bytes, 2 bytes for one integer, total of 10 integers.

Q 2. What is data abstraction? **(PTU, Dec. 2011, 2010, 2008 ; May 2011)**

Ans. Abstraction refers to the act of emphasizing and representing essential features and ignoring the irrelevant features. In object-oriented programming, word abstraction refers to the separation between the specification of data and its implementation. It results in better quality programs and more efficient programming techniques.

Such classes use the concept of data abstraction, they are also known as abstract data types.

Q 3. Relate object and class in OOP. **(PTU, May 2008)**

Ans. The concepts of class and object are interrelated. We cannot talk about an object without regard for its class. There are differences between these two terms. An object is a concrete entity that exists in time and space. A class represents only an abstraction, the essence of an object, as it were. We may speak of the class mammal, which represents characteristics common to all mammals. To identify a particular mammal in this class, we most speak of that mammal. In the context of object-oriented programming, we define class as a set of objects that share a common structure and a common behaviour. A single object is simply an instance of a class.

Q 4. What is a class in C++? Discuss generic class. **(PTU, May 2019 ; Dec. 2007)**

Ans. Class : A class serves as a plan or template. It specifies what data and what functions will be included in objects of that class. Defining the class doesn't create any objects.

A class is thus a collection of similar objects. In OOP, objects are member of classes. An object is a concrete entity that exists in time and space and has an identity, whereas a class represents only an abstraction, the essence of an object.

Generic class : Templates enable us to specify, with a single code segment, an entire range of related functions called template or generic functions or an entire range of related cl
classes or generic classes

We can create a single class template for a stack class and then C++ compiler will generate separate classes such as stack of int class, stack of float class, stack of string class etc. The syntax of class template is

```
template <class t1, class t2, .....>
class classname
{
    // class member with type t1, t2
    // whereever appropriate
}
```

All class templates begin with the keyword template followed by list of formal parameters to the class template where each formal parameters must be preceded by the keyword class or type name.

Q 5. What do you mean by OOP? **(PTU, Dec. 2007)**

Ans. OOP is object oriented programming. Object oriented programming was to remove the flaws of procedural programming approach. OOP treats the data as a most critical element in the program design and does not allow it be scatterd around the program. It ties the data and the functions that operate on it and protects it from the other functions.

Object oriented programming allows decomposition of a problem into a number of entities called objects and the builds data and functions around these objects.

The data of an object can only be accessed by functions associated to it. However, function of one object can access the function of another object. The communications among functions of different objects is called messaging. Characteristics of object oriented programming are :

1. Emphasis is on data rather than functions.
2. Programs are divided into set of related objects.
3. Objects communicate with each other through functions.
4. Data of an object is hidden and cannot be accessed by external functions.
5. Functions that operate on the data of an object are tied together with the data structure.

Q 6. Explain some objectives of object oriented programming. **(PTU, Dec. 2007)**

Ans. Following are the objectives of object oriented programming :

1. Objects : An object is an entity that has state, behaviour and identity. The structure and behaviour of similar objects are defined in their common class. The term instance and object are interchangeable.

2. Classes : A class represents only an abstraction, the essence of an object, as it were. Class is a set of objects that share a common structure and a common behaviour. A single object is simply an instance of a class.

3. Polymorphism : Polymorphism can be defined as a technique that allow to define various forms of a single function or operator that can be shared by various objects to perform the operations.

4. Inheritance : Inheritance is a relationship among classes wherein one class inherits the structure or behaviour defined in one or more classes. A class from which another class inherits is called its super class, parent class and a class that inherits from one or more classes is called sub class or derived class.

5. Encapsulation : Encapsulation is the cancelling of details of a data object from outside world. The data and functions are wrapped into a single unit called class.

Q 7. What is the purpose of cin and cout statement? **(PTU, May 2007)**

Ans. Cin : Input is handled with cin and the extraction operator >>, which cause values to be received from the standard input device usually the keyboard. We can write to files using only the object cin.

Cout : Output is most commonly handled in C++, with the cout object and << insertion operator, which together cause variables or constants to be displayed to the standard output device usually the screen. We can read the file using the only object cout.

Q 8. What is name mangling?

(PTU, Dec. 2006)

Ans. We can use member functions in a derived class that have the same name as those in the base class. In such a case, the member function of the derived class overrides the member function of the base class. Since more than one class contains member function with same name. This is known as name mangling of functions. With this name mangling and overriding functions, the calls in the program work the same way for objects of both base and derived classes.

Q 9. What is a manipulator? Why do we need it? **(PTU, Dec. 2006)**

Ans. Manipulators are operators used with the insertion operator << to modify – or manipulate the way data is displayed. The most common manipulators are :

1. endl : This manipulator causes a linefeed to be inserted into the stream. It has the same effect as sending the single '\n' character.

2. setw : The set w manipulator causes the number (or string) that follows it in the stream to be printed within a field n characters wide, where n is the argument to set w (n). The value is right-justified within the field.

Q 10. What does the name C++ signify? What is information hiding? **(PTU, May 2005)**

Ans. Classes were a major addition to the original C language. Initially it was called 'C with classes'. However, in 1983, the name was changed to C++. The ideas of suffering C with ++ came from the increment operator, since new feature are added to the features that existed and being used since long.

C++ is a superset of C. All C programs are also C++ programs.

Information Hiding : An object functions, called member functions in C++ provide the only way to access its data. If you want to read a data item in an object, you will call a member function in the object. You can't access the data directly. The data is hidden, so it is safe from accidential alteration. Information hiding is important, so that outsiders can use or access the data.

Q 11. What is the main purpose of header files in 'C++' program?

(PTU, Dec. 2005, 2004)

Ans. Header files are added with the preprocessor directive #include. It tells the compiler to add the source file before compiling iostream.h is an example of header file. It contains declarations that are needed by the cout identifier and the <<operator. Without these declarations the compiler won't recognize cout and will think << is being used incorrectly.

Q 12. List out the three weaknesses of 'C' and the three strength 'C++'.

(PTU, Dec. 2004)

Ans. Weaknesses of 'C' :

1. No matter how well the structured programming 'C' is, large problems become excessively complex.
2. Since every function has complete access to data. Anyone can change or destroy them. In the same way, global data can be corrupted by functions.
3. Many functions access the same data, the way the data is stored becomes critical. The arrangement of data can't be changed without modifying all the functions that access it.

Strength of 'C++' :

1. Once a function is written, created and debugged it can be distributed to other for use in their own programs. This is called reusability.

2. C++ class can be divided into subclasses. In C++ the original class is called the base class, other class can be defined that share its characteristics. These are called derived classes.
3. C++ has provision of operator or function overloading. Using operator or functions in different ways, depending on what they are operating on, is called polymorphism.

Q 13. Is it sometimes useful to specify a class from which no object will never be created? **(PTU, Dec. 2004)**

Ans. Yes, it is sometimes useful to specify a class from which no object will never be created. Sometimes classes are created only for deriving other classes, and no actual object of this class is created. These are called abstract classes.

Q 14. When will you use the word protected in c++? **(PTU, May 2004)**

Ans. A protected member can be accessed by member functions in its own class or in any class derived from its own class. It can't be accessed from functions outside these classes such as main ( ). Member functions can access members of the base class if the members are public or if they are protected.

When any data or functions that the derived class might need to access from base class it should be made protected rather than private.

Q 15. Give two advantages of OOP. **(PTU, May 2004)**

Ans. Following are the advantages of OOP :

1. The concept of OOP is data encapsulation and data abstraction that increases the reliability of the software systems by separating the procedural and representational specifications from its implementation.
2. Inheritance enhance the reversability of the code, thus increasing the productivity.

Q 16. What is data encapsulation? **(PTU, May 2004)**

Ans. An object functions called member functions in C++, that provide access to its data. If you want to read a data item in an object, you call member function in the object, it will read the item and return the value to you. The data is hidden, so it is safe from accidental alteration. Data and its function are said to be encapsulated into a single entity. Data encapsulation and data hiding are key terms in the description of object oriented languages.

Q 17. What are the types of operations that can be performed on objects?

Ans. Following are the operations that can be performed on objects :

1. Constructor : An operations that creates an object or initializes its state.

2. Destructor : An operation that frees the state of an object and destroys the object itself.

3. Modifier : An operation that alters the state of an object.

4. Selector : An operation that access the state of the object but does not alter it.

5. Iterator : An opertion that permits all parts of an object to be accessed in some well defined order.

Q 18. Create a class that contain a person's name and telephone number, using new dynamically allocate an object of this class and put your name and phone number into these fields within this objects.

Ans. **(PTU, May 2009)**

```
class person {
    private :
        char name [20] ;
        int telephone;
    public :
```

```
void get data (char nam, int no)
    {
        name = nam ;
        telephone = no ;
    }
    void display ( )
        {
        cout <<"Name is" << name ;
        cout <<"Telephone no. is" << telephone ;
        }
    }
void main ( )
{
    person *obj1 ;
    obj1 = new person ;
    char na [20] ;
    int no ;
    cout <<"Enter name & telephone no." ;
    cin >> na >> no ;
    obj1 → get data (na, no) ;
    obj1 → display ( ) ;
}
```

Q 19. What is meant by the interface of a class?

Ans. The interface of a class provides outside view of a class and encompasses the abstraction while hiding its structure and the secrets of its behaviour. This interface consists of the declarations of all the operations applicable to instances of this class, but it may include the declarations of other classes, constants, variables and exceptions that are needed to complete the abstraction.

Q 20. Compare structured and object oriented programming with the help of suitable examples. **(PTU, Dec. 2009 ; May 2008)**

Ans. Structured programming is also known as procedural programming or functional programming. In this, given problem is divided into small sized sub problems, each sub problem represents a specific task that can be handled efficiently and effectively. Then to solve each sub problem, a function is written.

The technique of hierarchical decomposition has been used to specify the tasks to be completed to find solution of a problem. These functions are written to manipulate the data to accomplish a specific task. Characteristics of structured programming are :

1. Emphasis is on algorithms.
2. Large programs are divided into smaller independent subprograms known as functions.
3. Majority of the funtions share global data.
4. Functions transform data from one form to another.
5. It employs top-down approach to program design.
6. The parts of the program are heavily dependent on each other i.e. tightly coupled to each other. Therefore, it is difficult to make a change in one part of the program without affecting the other.
7. The program logic dependent on the organization of data.

Example of structured programming are Basic C etc.

Object Oriented Programming : Object oriented programming treats the data as a critical element in the program design and does not allow it be scattered around the program. It ties the data and the functions that operate on it and protects it from the other functions.

OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.

The data of an object can only be accessed by functions associated with it. However, function of one object can access the function of another object. The communications among functions of different objects is called messaging. Characteristics of OOP are :

1. Emphasis is on data rather than functions.
2. Programs are divided into set of related objects.
3. Employs bottom-up approach to program design.
4. Objects communicate with each other through functions.
5. Functions that operate on the data of an object are tied together with the data structure.
6. Data of an object is hidden and cannot be accessed by external functions.
7. Data structures are designed in such a way so as to characterize the objects.

The organisation of programs using object oriented programming approach overcomes all the problems of structured programming.

The object-oriented programming use objects as basic building blocks, where each object is an instance of some class and classes are related to each other through inheritance. Some concepts uesd in OOP are :

Objects, Classes, Abstraction, Encapsulation, Data Hiding, Inheritance, Overhiding, Polymorphism, Dynamic binding, Message passing.

Examples of OOP's are C++, JAVA etc.

Q 21. Write a program to read data from the keyboard, write it to a file called INPUT, again the same data from the INPUT file, and display it one the screen. **(PTU, Dec. 2007)**

Ans.

```
#include <iostream.h>
#include <fstream.h>
class student
{
    private :
        char name [20] ;
        int age ;
    public :
        void get data ( )
        {
            cout <<"Enter name" ;
            cin >> name ;
            cout <<"Enter age" ;
            cin >> age ;
        }
    void show ( )
    {
        cout <<"Name is" <<name ;
        cout <<"Age is" << age ;
    }
```

```
};
void main ( )
{
        char ch ;
        student s1 ;
        fstream file ;
        file.open ("Input. Dat", ios : : app I ios : : out ios : : in) ; do {
        cout <<"Enter student data" ;
        s1. get data ;
        file.write (char *) & s1, size of (s1)) ;
        cout <<"Is another student" ;
        cin >> ch ;
        }
        while (ch = = 'y') ;
        file.seekg (0) ;
        file.read ((char*) & s1, size of (s1)) ;
        while (! file.eof ( ))
        {
              cout <<"Student" ;
              s1. show ( ) ;
              file.read ((char *) & s1, size of (s1)) ;
        }
}
```

Q 22. How the structures in c differ from that of structure in C++? **(PTU, May 2007)**

Ans. A structure is a collection of simple variables. The variables in a structure can be of different types. Some can be int, some can be float and so on. The data items in a structure are called members of the structure. In C programming, structures are considered advanced feature and appears at the end. Where as in C++, structures are one of the important concept in understanding of objects and classes. Structures in C++ and C serves a similar purpose to record data of different type. To access a structure, we need a variable of that structure. In C we need to include the keyword struct in structure definitions, where as in C the keyword is not necessary. e.g

```
struct student
{
    int age ;
    char name [20] ;
    char grade ;
} ;
```

In C++, its variable can be defined as

```
student s1 ;
```

where as in C, we need to add keyword struct also

```
struct student s1 ;
```

otherwise use of structures in C++ and C are same.

Q 23. How can the strings be represented in c++? **(PTU, May 2007)**

Ans. Strings are array of characters. Using array of characters, we can store string data. A string in C++ is a variable length array of characters that is delimited by null character ('\0'). C++ language does permit the use of any character except the null character. It is common to use formatting characters such as tabs, format specifiers etc. in strings.

A string is represented as array of characters. It is terminated by the null ('\0') character. use a string is stored in an array, the name of the array and hence string, is a pointer to the nning of the string. **e.g.**

```
char str [ ] = "You are welcome" ;
```

In this str is array of characters that stores string "You are welcome" and compiler automatically nds the null character.

Q 24. Write a program to accept the record of 5 persons with their names, age & addresses o display them. Use structures to implement the program. **(PTU, May 2014)**

Ans.

```
#include<iostream.h>
using namespace std;
struct student
{
char name[50]
int age;
char address[100];
};
int main (void)
{
struct student stud[5];
int i;
cout<<"Enter Students record five times\n";
cout<<".....\n";
cout<<"Enter Students Data\n";
for (i=0;i<5;i++)
{
cout<<"\n Enter Student Name please:";
cin>>stud[i].name;
cout<<"student Age:";
cin>>stud[i].age;
cout<<"student Address:";
cin>>stud[i].address;
}
cout<<"\n.....Display Student Data .....\n";
cout<<".....\n";
for(i=0;i<5;i++)
{
cout<<"\n Student Name"<<i<<":<<stud[i].name;
cout<<"\n Student Age"<<i<<":"<<stud[i].age;
cout<<"\n Student Address"<<i<<":"<<stud[i].address;
}
return 1;
}
```

Q 25. Discuss the features of an object oriented programming in detail. **(PTU, May 2010, 2006)**

Ans. Following are the features of object oriented programming :

1. Objects : An object is an entity that has state, behaviour and identify. The structure and behaviour of similar objects are defined in their common class.

2. Class : The concepts of class and objects are tightly interwoven. A class represents only an abstraction, the essence of an object, as it were. In object-oriented programming we define class as a set of objects that share a common structure and common behaviour.

3. Inheritance : Inheritance is a relationship among classes where in one class inherits the structure and behaviour defined in one or more classes. The class from which another class inherits its base class, super class or parent class and that inherits from one or more classes is a sub class, derived class or child class.

The concept of inheritance provides reusability.

4. Reusability : Once a class has been written, created and debugged, it can be distributed to other programmers for use in their own programs. This is called reusability.

In OOP, the concept of inheritance provides an important idea of reusability. A programmer can take an existing class and without modifying it, add additional features and capabilities to it.

5. Polymorphism and overloading : Polymorphism is a technique that allow to define various forms of a single function or operator that can be shared by various objects to perform the operations polymorphism is achieved using overloading and dynamic binding. When an existing operator such as + or = is given the capability to operate on a new data type, it is said to be overloaded.

6. Dynamic Binding : Binding is the linking of a function call to the function definition that will be executed on the call during program execution. The function call is tied to the function definition during the linking process. This is known as early binding. However, the OOP permits dynamic binding also called late binding, where the tie-up of function call to the address code is delayed until the run-time.

Q 26. Write and explain the syntax of a class declaration in c++. **(PTU, May 2004)**

Ans. The syntax of specifying a class is similar to specifying a structure as shown below:

```
class classname
{
    Access specifier ;
    // data members
    // function members
};
```

The access specifier (private, protected or public) determines which parts of a software can access these members of a class. By default, access specifier for a class is private and an access specifier remain in effect till overridden by another access specifier.

The data members describe the structure of the objects. The function members describe the behaviour of the objects.

Each of these members can be declared as private, protected or public. The private members can only be accessed from within the class. Members of the class and members of it subclass can access a protected member. The public members can also be accessed by non-members of the class, i.e. from outside the class.

Q 27. Name popular C++ compilers available.

Ans. Following are the C++ compilers :

1. C++ Compiler on windows platform : On window based plateform, turbo C++ compiler is the user's preferred choice. The turbo C++ compiler can be used in two different ways :

(i) The frequently used method for creating programs in Turbo C++'s Integrated Developmen Environment. In this, all necessary operations for program development are available in a unifie screen display with menus and windows.

(ii) The second method is a traditional C++ command line system in which the functions of editing, compiling, linking and executing a program are invoked from the command prompt as separate programs.

2. C++ compiler on unix platform : In this we use command prompt. The program will be compiled and if there is no syntax error, it will be linked with system libraries and the executable code is stored in file a out. To run the program, type the name of file at command prompt.

3. C++ compiler on linux platform : In this platform, again command prompt is used. The program will be compiled and if there is no syntax error, it subsequently will be linked with system libraries and the code is stored in a.out. To run the program, enter the name of file at command prompt.

(PTU, Dec. 2010)

Q 28. What is data hiding?

Ans. Data hiding is also called the encapsulation of data. The wrapping up of data and function into a single unit known as class is called the data hiding. In this, the data is not accessible to the outside world and there are accessed only to those functions which are wrapped in the class. These function provide the interface between the objects data and the program.

(PTU, Dec. 2011)

Q 29. Write short note on Information hiding.

Ans. The wrapping up of data and functions into a single unit is known as encapsulation. Data encapsulation is the most striking feature of class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. This insulation of data from direct access by the program is called data hiding or information hiding.

Q 30. What is polymorphism? Give difference between function overloading and overriding with example.

(PTU, May 2011)

Ans. Polymorphism : Polymorphism can be defined as a technique that allow to define various forms of a single function or operator that can be shared by various objects to perform the operation. The polymorphism is achieved using overloading and dynamic binding.

e.g.
```
#include <iostream.h>
Class A
{
    Public :
    Void show ( )
    {
        Cout <<"Base" ;
    } ;
Class B : Public A
{
    Public :
    Void show ( )
    {
        Cout <<"Drived1" ;
    } ;
Class C : Public A
{
    Public :
    Void show ( )
    {
```

```
            Cout <<"Derived 2";
        } ;
    Void main ( )
    {
        B obj ;
        C obj 1 ;
        A* Ptr ;
        Ptr = obj ;
        Ptr = show ( ) ;
        Ptr = obj 1 ;
        Ptr = show ( ) ;
    }
```

This is example of function overloading show ( ) is overloaded in derived class B and C. AND function is type of polymorphism.

Q 31. Different between a structure of the data type and a union. **(PTU, May 2011, 2009)**

OR

How is union different from structure? **(PTU, May 2012, 2007)**

Ans. Union is same way similar to a structure, but in other ways are completely different, allow us to store data items of different data types. But these data items are stored in same memory and store one value of given type at one time and another value at different time. The size of memory allocated depends on the data item whose memory requirements are more. The syntax for declaring a union, declaring variables of union type, accessing elements of a union, is identical to that of structures, except for storage of members of union and their initialization.

Q 32. What is meant by cast operator? **(PTU, May 2009)**

Ans. Cast Operator : When we want to convert between user defined data types and basic types, we can't rely on built in conversion routines, since the compiler doesn't know anything about user-defined types besides what we tell it. When we need to convert a value from one type to another then the compiler will not do it automatically.

e.g.

```
#include <iostream.h>
void main ( )
{
    int n = 25000 ;
    n = (n * 10)/10 ;
    cout <<"No. is" << n;          // wrong answer
    n = 25000 ;
    n = (long (n) * 10) / 10 ;     // cast to long
    cout <<"no. is" << n ;         // right answer
}
```

We can cast n to type long before multipling. This is called cerosion the data is coerced into becoming another type.

Q 33. Explain the use of ternary operator in C++. **(PTU, May 2009)**

Ans. Ternary Operator :

Syntax of ternary operator is

```
exp1 ? exp2 : exp3 ;
```

Where exp1, exp2, exp3 are expressions.

The expression exp1 is evaluateed first. If it is non-zero, then the expression exp2 is evaluated, and that is the value of conditional expression otherwise expression exp3 is evaluated and that is the value of the conditional expression.

e.g. big = (a > b) ? a : b;

Q 34. What is a scope resolution operator? What part does it play in definition of member functions? **(PTU, May 2009, 2005 ; Dec. 2005)**

Ans. Member functions can be defined outside the class using scope resolution operator (: :) is scope resolution operator. Scope resolution operator we use to define function outside the class. For this, function is declared in the class and its definition is written outside the class. Its syntax is

return type class name : : member function (argument list) ;

Q 35. Explain the syntactic rules for setfill manipulator. **(PTU, May 2009)**

Ans. Setfill manipulator is predefined parameterized manipulator. This is defined in iomanip.h header file. It don't need an object to invoke them as in the case of ios member function that can only be invoked on an object of output stream.

It is used to set the file character. Its syntax is

setfill (int)

e.g. output <<setfill ('*') ;

It set fill character as '*'.

Q 36. How is the basefield of a number defined in C++? **(PTU, May 2009)**

Ans. Basefield of a number is defined by using self ( ) function, which is a member function function of the ios class, and can be invoked only on an object of output stream. Its prototype is

long setf (long-setbits, long-field) ;

where the first argument_setbits is one of the flags defined in ios class and it specifies action required for the output. The second argument_field is also defined in ios class specifies the group to which formatting flag belongs.

Flags that are used with self ( ) function for defining basefield of a number :

	Flag	Field	Action
1.	doc	basefield	Integer value is output in decimal base.
2.	oct		Integer value is output in the octal base.
3.	hex		Integer value is output in hexadecimal base.

Q 37. Explain the salient features of the typedef. **(PTU, May 2009)**

Ans. Typedef : A type definition, typedef, gives a name to a data type by creating a new type that can be used anywhere a type is permitted. The syntax for type definition is

typedef datatype identifier ;

where data type is either built-in data type or user-defined data type and identifier, usually in uppercaes, is the new and convenient name of the data type.

The typedef keyword tells the compiler to recognize the identifier as synonymous of data type.

e.g. the statement

typedef float real ;

create real as a new name for float data type.

Q 38. Specify the steps to develop and execute a program.

Ans. Following are the steps to develop and execute a program : **(PTU, Dec. 2008 ;May 2008)**

1. A programmer cannot write program unless he knows how to solve problem manually. It is important that before attempting to write a program, programmer must solve the problem mannually.
2. In order to ensure that the program instructions are appropriate for the given problem and are in correct sequence, program must be planned before they are written.
3. Then write algorithm. An algorithm is a finite sequence of instructions defining the solution of a particular problem.
4. Next step is flowchart. A flowchart is a pictorial representation of an algorithm. A flowchart uses different shapes to denote different types of instructions.
5. Then write program and compile that program to find errors in it and then execute the program.

Q 39. What are static variables? **(PTU, Dec. 2008 ; May 2008)**

Ans. A static variable has the visibility of a local variable but the lifetime of an external variable. Thus it is visible only inside the function in which it is defined, but it remains in existence for the life of the program.

Static variables are used when its necessary for a function to remember a value when it is not being executed that is between calls to the function. When static variables are initialized, the initialized takes place only once at the beginning of the program. They are not reinitialized each time the function is called, as ordinary automatic variables are.

Q 40. What do you mean by precedence of operators? **(PTU, Dec. 2008 ; May 2008)**

Ans. Precedence is used to determine the order in which different operators are evaluated in an expression. Precedence is applied before associativity to determine the order in which expressions are evaluated . The concept of precedence is well defined in mathematics like rule of BODMAS. In algebra division and multiplication is performed before addition and subtraction.

e.g. 10 + 2 * 5

This expression consists of one addition and one multiplication operator. As multiplication has higher precedence than addition and this multiplication is performed before addition. Therefore, the expression will be evaluated as

$(10 + (2 * 5)) \rightarrow (10 + 10) \rightarrow 20$

giving value 20 as value of the entire expression.

Q 41. What is type casting? **(PTU, Dec. 2011, 2008 ; May 2008)**

Ans. The type casting changes the meaning of the basic data type with which they are used. Type casting applies to data conversions specified by the programmer, sometimes we need to convert a value from one type to another in a situation where the compiler will not do it automatically. e.g. suppose an integer variable a has value 25000. When we multiply this by 10 the result 250000, is too large to fit in variable of type int or unsigned int. This leads to wrong answer. We can use typecasting a to type long before multiplying. The expression

long (a)

casts a to type long. It generates a temporary variable of type long with the same value as a is the temporary variable that is multiplied by 10. Since it is type long, the result fits.

Q 42. Briefly explain the use of break with switch statement in C++. **(PTU, Dec. 2008 ; May 2008)**

Ans. The break statement is always used inside the body of switch statement.

In switch statement, it is used as the last statement of the statement block of every case except the last one when executed it transfers the control out of switch statement and the execution of the program continues from the statement following switch statement. Its syntax is

```
switch (expr.)
{
    case val-1 ;
        statements
        break ;
    case val-2 ;
        statements
        break ;
    :
    case val-n ;
        statements
        break ;
    default ;
            statements
}
```

Q 43. What is a control statement? Discuss with an example. **(PTU, Dec. 2007)**

Ans. Some statements transfer the control from one part of the program to another part. These statements are known as control statements. Break, continue and goto statements are control statments.

e.g.

```
#include <iostream.h>
void main ( )
{
    int i = 0, k, m
    while (i < = 100)
    {
        cin >> k ;
        if (k = = 0)
            break ;
        m = i/k ;
        cout << m ;
    }
}
```

In this example, the execution of the while loop will be terminated as soon as the user enters valu 0 for variable K as input or the value of variable i exceeds 100, which ever happens earlier.

Q 44. Why it is better to use symbolic names for size specification in average? **(PTU, May 2007)**

Ans. We should declare the size of an array or specify constants using the preprocessor directive #define. Also known as symbolic name. This directive sets up an equivalence between an indentifier and a text phrase. **e.g.**

```
#define size 20
```

appearing at the begining of your program specifies that the identifier size will be replaced by the text 20 throughout the program. We can't specify the data type of the constant using #define.

Q 45. Why do an array subscripts start from 0 instead of 1? **(PTU, May 2007)**

Ans. An array subscripts start from 0 instead of 1. The value of each subscript expressed as an integer constant or integer variable.

In computer memory, an array will be given memory as much as requirement of its data type and multiplyed by subscript value. And first memory address will assigned subscript 0 by the compiler.

So when we access values or store value in array we start its subscript from 0 instead of 1 as it is assigned to first memory address of array.

Q 46. What is structure? How is it declared? **(PTU, May 2007)**

Ans. A structure permits the data items to be of different data types. These data items occupy contiguous memory locations. The data items in a structure are called members of the structure. The syntax for declaring a structure is

```
struct tag
{
    .....
    field list
    .....
};
```

The declaration starts with the keyword struct .... next element in declaration is tag. The tag is identifier for the structure and will be used to declare variable, arguments of the functions.

e.g.
```
struct student
{
    int roll no ;
    char name [20] ;
    char grade ;
};
```

Q 47. What is dangling reference? **(PTU, May 2019, 2007)**

Ans. Sometimes programmer fails to initialize a pointer, with a valid address. Such an initialized pointer, referred to as dangling reference, can end up pointing anywhere in memory that may include the program code itself. If programmer assign a value to such pointer, the value will overwrite the program and program may show undesirable behaviour or my even crash. When the system stop responding, we say the system has hanged up and the only option left us to reboot the system. Therefore, care must be taken to initialized pointers with valid address.

Q 48. What does a header file contains? **(PTU, May 2007)**

Ans. The preprocessor directive #include tells the compiler to add the source file before compiling. These source files are called header files. e.g. iostream.h it contains declarations that are needed by the cin, cout identifier. Without these declarations the compiler won't recognize these statements.

A header file containing function declaration must be inserted into your source file with an #include statement.

Q 49. What is the advantage contiguous memory allocation in arrays? (PTU, Dec. 2006)

Ans. An array is a collection of homogeneous data elements (i.e. of same data type) described by a single name and each individual element of array is referenced by a subscripted variable formed by affixiing to the array name a subscript or index enclosed in brackets.

Arrays allocates a contiguous memory block. How many bytes to be allocate to a array it depends upon data type and value of subscript.

e.g. int A [10] ;

It allocates 20 bytes to array A as 2-bytes for each element of type int.

Contiguous memory allocation allows us to generate address of any element of array. The array name is a symbolic reference for the address to the first byte of the block of memory allocated for the array. The address of the first byte for the array is known a base address of the array. Whenever we use the arrays name, we refer to the first byte of the array. The index represents an offset from the beginning of the array to the element being referenced

Q 50. If a and b are two arrays of same type, can the statement a = b, work? **(PTU, Dec. 2006)**

Ans. a & b are two arrays of same type.

Name of array gives its base address i.e. address of first byte of array. So a gives base address of array a and b gives base address of array b. So when we execute.

```
a = b ;
```

this statement will assign base address of array b to base address of array a and we will lost previous values of array a. Hence, this statement is not acceptable.

Q 51. How is structure passed to a function? **(PTU, Dec. 2006)**

Ans. A structure can be passed to the function in following ways :

1. Individual members can be passed as an argument to a function by both y value and call by address, just as simple variables.
2. The whole structure can be passed as an argument by value and the function can use the local copy of the structure.
3. The address of a structure can be passed as an argument by reference and the function can access the elements of the structure through indirection and indirect selection operator (→).

Q 52. How is memory allocated to structure variables? **(PTU, Dec. 2006)**

Ans. A structure is collection of data items of different data types. These data items occpy contiguous memory locations. How many bytes to be allocated to a structure depends upon the data items included in a structure. Total memory or bytes to be allocated is the sum of memory required by the data items included in structure.

e.g.

```
struct student
{
    int roll no ;
    char name [20] ;
    char grade ;
}
```

The struct student will be allocated 23 bytes of contiguous memory as 2 bytes for integer variable roll no, 20 bytes for name variable and one byte for character variable grade.

Q 53. What is null pointer? **(PTU, Dec. 2006)**

Ans. Sometimes a pointer points to address 0, which is called NULL. Such pointer is known as NULL pointer e.g. this may happen that if the pointer variable is declared as global since global variables are intialized to 0. Likewise, this can also happen for a local un-initialized pointer variable, particularly for local static variables, they are also initialized to 0 when it happens, then the system will display a message "Null pointer assignment" on termination of the program.

Q 54. Can we write a [i] as i [a]? **(PTU, Dec. 2006)**

Ans. Yes, we can write a [i] as i [a] a [i] points to it h element of a array and similarly i[a], is also points to ith element of a array. So writing a [i] and i [a] are the same things i.e. their meaning and purpose is same.

Q 55. Explain various control statements used in C++ language in short. **(PTU, May 2006)**

Ans. Following are the control statements :

1. Break : It is always used inside the body of the switch statement and looping statements. In switch; it is used as the last statement of the statement block of every case except the last one.

When executed it transfers the control out of switch statement . In for, while and do-while, it is always used in conjunction with if statement.

2. Continue : It is always used inside the body of the looping statements. The continue statement transfers the control to the beginning of the next iteration of the loop thus by passing the statements which are not yet executed. But it is used in conjunction with the if statemnt. Thus, continue statement when exeuted terminates the current iteration of the loop.

3. Goto : Goto statement can transfer the control to any part of the program. The target destination of the goto statement is marked by a label. The syntax for goto is

```
// some statements
goto label ;
// some more statements
label :
// more statements.
```

4. Exit : It is a library function and can be used to terminate program immaturely when certain conditions are met or not met and can cause the program to fail.

Q 56. What do you mean by the lifetime and scope of a variable? **(PTU, May 2006)**

Ans. Lifetime of a variable : Lifetime of a variable is the length of time it retains a particular value. Lifetime of all storage classes are different.

Scope : Scope of visibility of a variable refers to those parts of a program that will be able to recognize it. Scope of all storage classes is also different.

Q 57. What will be the result of execution of the following statements?

(PTU, May 2006)

Ans. 2 * ((8 / 5) + (4 * (5 – 3)) % (8 + 5 – 2))

2 * (1 + (4 *2) % 11)

2 * (1 + 8 % 11)

2 * (1 + 0)

2 * 1 = 2

Q 58. In each of the following, assume that m has the value 5 and n has the value 2 before the statement executes. Tell what the values of m and n will be after each of the following statement execute :

(i) (m + n) (ii) – – m (iii) m < = n (iv) n – – **(PTU, May 2006)**

Ans. m = 5, n = 2

(i) – (m + n) = – (5 + 2) = –7

(ii) – – m = – – 5 = 4

(iii) m < = n ; 5 < = 2 ; 0

(iv) n – – = 2 – – = 1

Q 59. How does an enum statement differ from type def statement?

(PTU, May 2006)

Ans. enum : enum is used to define use-defined data type. In an enumerated data type, each integer value is given an identifier called enumeration constant. To declare an enumerated data type. We must declare its identifier and its values. Since enumerated data type is derived from integer data type. Syntax is

enum typename {identifier list} ;

The keyword enum is followed by an identifier which is followed by an identifier list enclosed in a set of braces and terminated by semicolon.

type def : typedef gives a name to a data type by creating a new type that can be used anywhere a type is permitted. The syntax is

typedef datatype identifier ;

where datatype is either built-in data type or user defined data type and identifier, usually a uppercase is the new name for the data type.

Q 60. What is the purpose of return statement? **(PTU, May 2005)**

Ans. The return statement serves two purposes :

1. Execution of return statement immediately transfers control from the function back to the calling function.
2. Whatever is following the return statement is returned as a value to the calling function.

The syntax of return statement is

```
return ;
or
return (exp) ;
```

where exp can be a constant, variable or expression. Use of parentheses around exp is optional. The first form is used with functions defined with return type as void.

The return statement need not be at the end of the function. It can be used any where in the function. As soon as it is executed, the control will return to the calling function. A function can contain any number of return statements.

Q 61. Illustrate the use of continue statements in 'C++'. **(PTU, Dec. 2004)**

Ans. Continue is used when we want to go back to top of the loop when something unexpected happens. It is always used inside the body of looping statements. The continue statement transfers the control to the begining of the loop of next iteration thus bypassing the statements which are not yet executed. It is always used in conjunction with the if statement.

e.g.
```
for ( )
{
    .....
    if (condition)
    continue ;
    .....
}
```

Q 62. Arrange in order of preference (highest first) the following kind of 'C++' operators. **(PTU, Dec. 2004)**

Ans. C++ operators in order of their preference :

	Category	Operators
1.	Unary operators	!, ~, ++, – –, + , –, *, &, size of
2.	Arithmetic	*, /, %, +, –
3.	Shift	<<, >>
4.	Relational	<, < =, >, > = , = =, ! =
5.	Bitwise	&, ^, |
6.	Logical	&&, ||
7.	Coditional	? :
8.	Assignment	=, + =, – =, * =, / =, % =, & =, ^ =, ! =, << =, >> =

Q 63. Write a for loop that will never be executed. **(PTU, Dec. 2005, 2004)**

Ans.
```
#include <iostream.h>
void main ( )
```

```
{
    int i ;
    for (i = 10 ; i < 10 ; ++i)
{
    cout <<i ;
}
return ;
}
```

Q 64. When will you use user defined data type in C++? **(PTU, May 2004)**

Ans. Enumerated data type is a user defined data type based on the standard integer data type. In an enumerated data type, each integer value is given an identifier called enumeration constant.

To declare an enumerated data type, we must declare its identifier and its values. Since enumerated data type is derived from integer data type, its operations are the same as for integers syntax is

enum typename {identifier list} ;

e.g. enum Boolean {False, true} ;

Q 65. What are different ways of adding comments in C++ program?

Ans. Following are the different ways of adding comments :

1. C++ allows to add multiple comments in a program. This is done by starting the comment with two characters '/*' and ending with the character '*/'. Between these pair of characters, called delimiters, any number of lines can be included, which may contain characters in lowercase as well as uppercase.

2. Starting the comments with two successive slashes does the second style of adding comments ('//'). This style of commenting is preferred if the comments comprise few words or a single line. It can be used as a separate line or one the same line as that of an instruction.

Q 66. What is an expression? List various types of expression used in C++.

Ans. An expression is a formula for computing a value. It consists of a sequence of operands and operators. The operands may contain function references, variables and constants. The operators specify the action to be performed on the operands. In the following expression :

a * b.

Multiplication (*) is an operator and a and b are operands. There are four types of expressions in C++ language. These are :

1. Arithmetic Expression
2. Relational Expression
3. Logical Expression
4. Conditional Expression.

Each type of expression takes certain types of operands and uses a specific set of operators. Evaluation of every expression produces a value of specific type. Expressions are not statements but may be components of statements.

Q 67. What are various data types supported by Turbo C++? Give memory requirement o each type.

Ans. The various kinds of data types supported by C++ are :

1. **Integer Numbers :** It is of three types :
   (i) short int : Small whole numbers and memory requirement is 2 bytes.
   (ii) int : Medium whole numbers. Its requirement is also is 2 bytes.
   (iii) long int : It stores large whole numbers. Its requirement is 4 bytes.

2. **Real Numbers :** It is also of three types :
   (i) Float : It stores small real numbers. And its memory requirement is 4 bytes.
   (ii) Double : It stores large real numbers. Its memory requirement is 8 bytes.
   (iii) Long double : It stores very large real numbers.
   And its memory requirement is 10 bytes.
3. **Character :**
   Char : It stores single character. Its memory requirement is 1 byte.

Q 68. Name and describe the usual purpose of three expression in a for statement.

Ans. The three expressions of for statement are :

1. Initialization Expression : The initialization expression is executed only once, when the loop first starts. It gives the loop variable an initial value.

2. The test expression : The test expression usually involves a relational operator. It is evaluated each time through the loop. Just before the body of the loop is executed. It determines whether the loop will be executed again. If the test expression is true, the loop is executed one more time. If its false, the loop ends and control passes to the statements following the loop.

3. The increment expression : The increment expression changes the value of the loop variable, often by incrementing it. It is always executed at the end of the loop, after the loop body has been executed.

Q 69. What is the principle reason for passing arguments by reference?

Ans. Passing arguments by reference uses a different mechanism. Instead of a value being passed to the function, a reference to the original variable, in the calling program is passed.

The primary advantage of passing by reference is that the function can access the actual variable in the calling program. Among other benefits this provides a mechanism for returning more than one value from the function back to the calling program.

Q 70. List various types of operators available in C++ and also give their precedence.

Ans. Following are the operators available in C++ according to their precedence.

	Operators	Precedence Level
	( ), [ ], →,	1
Unary	!, ~, ++, – –, +, –, *,	
operators	&, size of	2
Arithmetic	*, /, %, +, –	3
Shift operators	<<, >>	4
Relational	<, < =, >, > =	5
Equality	= =, ! =	6
Bitwise	&, ^, |	7
Logical	&&, ||	8
Conditional	?	9
Assignment	=, + = , – =, * =, /= , % = & =, ^ =, ! =, << =, >> =	10
Comma	,	11

Q 71. What is a variable? What are the rules for naming a variable?

Ans. A variable provide us with named storage that can write to, retrieve and manipulate throughout the program. Variables are memory location in the computer memory that holds data. Contents of variable may vary hence the name variable. Variables hold different kind of data and the same variable might hold different values during the execution of a program.

Each variabel is associated with a specific type, which determines the size and layout its associated memory. Variables in C++ can be declared anywhere in the program. By declaring the variables nearer to its first usage and by limiting the scope of the variable to the block where it is to be used, the programmer can economize the memory usage by the program.

Q 72. Describe the I/O using get ( ) and put ( ) functions.

Ans. Get ( ) : get ( ) function reads one character, including white space characters, at a time from the input stream. It is overloaded in istream and its prototype are as follows :

void get (char &) ; // it reads character into variable

It can be used with standard input stream object or with user defined object of istream class. e.g.

```
char ch ;
ch = cin. get ( ) ;
```

put ( ) function : The put ( ) function sends are character at a time to the output stream. Its prototype is

void put (char) ;

It can be used with standard output stream object or with user defined object of the ostream class.

e.g.
```
char ch = 'A' ;
cout. put (ch) ;
```
displays the value of the character variable ch i.e. character A.

Q 73. What does nesting mean?

Ans. Statements can be nested i.e. an statement can be contained within another statement, e.g. If statements can be nested i.e. an if statement can be contained within another if statement. The inner if statement will be executed if the condition of the outer if statement evaluates to non-zero value.

Q 74. What is difference between while and do-while statements?

Ans. While statement : The while statement is suited for problems where it is not known in advance that how many times a statement or statement block will be executed. Its syntax is

```
while (expression)
{
    statements
}
```

where expression is a constant, variable or an expression. The statements are executed till the expression evaluates to non-zero value. Whenever expression evaluates to zero value, the execution of while statement will terminate and controll will pass to a statement immediately following it.

do-while statement : The do-while statement, like while statement, is also suited for problems where it is not known in advance that how many times a statement will be executed. Its syntax is

```
do {
    statements
}
while (expression) ;
```

Where exp. is constant, variable or an expression.

The statement-block is executed repeatedly till the exp. evaluates to a non-zero value. But as compared to while-statement, do-while statement always executes once because in this, exp. is written at the end of loop and it is evaluated at the end when once statements of do-while are executed.

Q 75. What are inline functions? How are they different from normal functions?

Ans. A function call involves transfer of control to a specified address and returning to the instruction following the function call. Before transferring the control, CPU stores the contents of its registers and the address of the instruction following the function call. The time taken during this whole process is called context switch time and constitutes an overhead in the execution of the program.

This overhead is large if the time required to execute a function is small than the context switch time. The C++ language provides an alternative to above problem in the form of inline functions.

Inline functions are those functions whose body is inserted in place of the function call during the compilation process. Therefore, with inline functions, the program will not occur any context-switching overhead. An inline function definition is similar to an ordinary function except the keyword inline preceedes the function definition.

Q 76. Which type of variables should be made register variables?

Ans. The compiler assign a register variable to one of the processor's register instead of storing it in the memory. Value stored in register can be accessed much faster than the value stored in memory. So those variables on which the program spends most of its time to process them such as index variables used of for loop.

Q 77. How a structure is different from an array?

Ans. A structure permits the data items to be of different data types. These data items occupy contiguous memory locations whereas array permits data items of same data type. They also occupy contiguous memory locations.

Q 78. What does nesting of structures mean?

Ans. There can be structures that contain other structures as its elements. This can be powerful way to create complex data types. The only restriction on nesting of structures is that a structure cannot contain a member that is itself a structure of the same type as the outer structure. e.g.

```
struct student
{
    int roll no ;
    char name [20] ;
    struct date
    {
        int day ;
        int month ;
        int year ;
    }
        date of birth ;
    char class [5] ;
} ;
```

Here student structure is acting as a nesting structure whereas structure date is acting as nested structure. The data structure exists only in the scope of student structure.

PROGRAMS

Program : Program to enter temperature in degrees Fahrenheit, converts it to celsius.

```
#include <iostream.h>
void main ( )
{
        int ftemp ;
        cout <<"Enter temperature in Fahrenheit" ;
        cin >> ftemp ;
        int ctemp = ( temp – 32) * 5/9 ;
        cout <<"celsius is" << ctemp ;
}
```

Program : Program to calculate factorial of a number.

```
#include <iostream.h>
void main ( )
{
        unsigned int no ;
        unsigned long fact = 1 ;
        cout <<"Enter a number" ;
        cin >> no ;
        for (int j = no ; j = 0 ; j – –)
            fact * = j ;
        cout <<"Factorial is" << fact ;
}
```

Program : Program to print n fibonnacci numbers ;

```
#include <iostream.h>
void main ( )
{
        int next = 0, last = 1, n, i = 0, sum
        cout <<"Enter how many numbers to print" ;
        cin >> n ;
        while (i < = n)
        {
            cout << last <<" \n" ;
            sum = next + last ;
            next = last ;
            last = sum ;
            i ++ ;
        }
}
```

Program : Program to check whether entered number is prime or not.

```
#include <iostream.h>
#include <process.h>
void main ( )
{
        long n, j ;
```

```
        cout <<"Enter a number" ;
        cin >> n ;
        for (J = 2, J < n/2 ; J++)
            if (n% J = = 0)
            {
                cout <<"Its not prime" ;
                exit (0) ;
            }
        cout <<"Its prime" ;
}
```

Program : Program to find sum of digits of a positive integer number.

```
#include <iostream.h>
void main ( )
{
        int sum = 0, digit ;
        long no, temp ;
        cout <<"Enter any number" ;
        cin >> no ;
        temp = no ;
        while (temp > 0)
        {
            digit = temp % 10 ;
            temp / = 10 ;
            sum +  = digit ;
        }
        cout <<"Sum of digits" << no << " is " ;
        cout <<sum ;
}
```

Program : Program to check whether the given number is palindrome or not

```
#include <iostream.h>
void main ( )
{
        int sum = 0, digit ;
        int no, temp ;
        cout <<"Enter any number" ;
        cin >> no ;
        temp = no ;
        while (temp > 0)
        {
            digit = temp % 10 ;
            temp / = 10 ;
```

```
            cout <<no<< "is not a palindrome" ;
}
```

Program : Program to sort an array of integers using bubble sort.

```
#include <iostream.h>
#define size 100
void main ( )
{
      int a [size], i, n, k, temp ;
      cout <<"enter size of array" ;
      cin >> n ;
      if (n > size)
      {
            cout <<"size of array is more than declared" ;
            exit (1) ;
      }
      cout <<"Enter elements of array" ;
      for (i = 0 ; i < n ; i ++)
            cin >> a [i] ;
      for (K = 0 ; K < n – 1 ; K ++)
      {
            for (i = 0 ; i < n – K – 1 ; i ++)
            {
                  if (a [i] > a [i + 1])
                  {
                        temp = a [i] ;
                        a [i]  = a [i + 1] ;
                        a [i + 1] = temp ;
                  }
            }
      }
      cout << "Sorted array elements are" ;
      for (i = 0 ; i < n ; i++)
            cout << a [i] < " " ;
}
```

Q 79. How is register variable different form an automatic variable? (PTU, May 2009)

Ans. Register class : In register storage classes, the visibility and lifetime of the variable is imited to the block in which it is declared. Compiler assign register variable to one of the processor's egister instead of storing it in the memory. Value stored in register can be accessed must be faster han the value stored in memory. But there are limited number of registers in a processor, and not all variables can be register variables. Features of register are :

1. Stored in register, if a register is available. If no register is available, the variable is stored in memory and works as if it is auto.
2. Local to the block in which the variable is declared.
3. If not initialized in the declaration, their initial value is unperdictable.
4. It retains its value till the control remains in the block in which the variable is declared.

e.g.
```
#include <iostream.h>
void main ( )
{
    register int a ;
    cout << "Enter any number " ;
    cin >> a
    cout <<"Number is" <<a;
}
```

Automatic variable : The variable declared with auto storage class, which is also the storage class by default, have following features :

– Stored in memory.

– If not initialized in the declaration statement, their initial value is undefined value or garbage value.

– Local to the block in which the variable is declared. If the variable is declared within a function, then it is only visible to that function.

– It retains its value till the control remains in the block in which the variable is declared. As the execution of the block terminates, it is cleared/destroyed.

Q 80. What is a macro and how is it different from a preprocessor? (PTU, May 2009)

Ans. Macro : Macros are single identifiers that are equivalent to expressions, complete statements or group of statements. Macros resemble functions in this sense. They are defined in an altogether difference manner than functions and they are treated differently during the compilation process. Macro definitions are placed at the beginning of a file, ahead of the first function definition. The scope of a macro definition extends from its point of definition to the end of the file. A macro defined in one file is not recognized within another file.

Multiple macros can be defined by placing a backward slash (\) at the end of each line except the last.

Preprocessor : The preprocessor is a collection of special statements, called directives, that are executed at the beginning of the compilation process. The #include and #define statements are preprocessor directives. Additional preprocessor directives are #if, #elif, #else, #endif, #ifdef, #ifndef, #line and #undef. The preprocessor also includes special operator like # and ##.

Preprocessor directives usually appear at the beginning of a program though it is not a firm requirement. They may appear anywhere in a program.

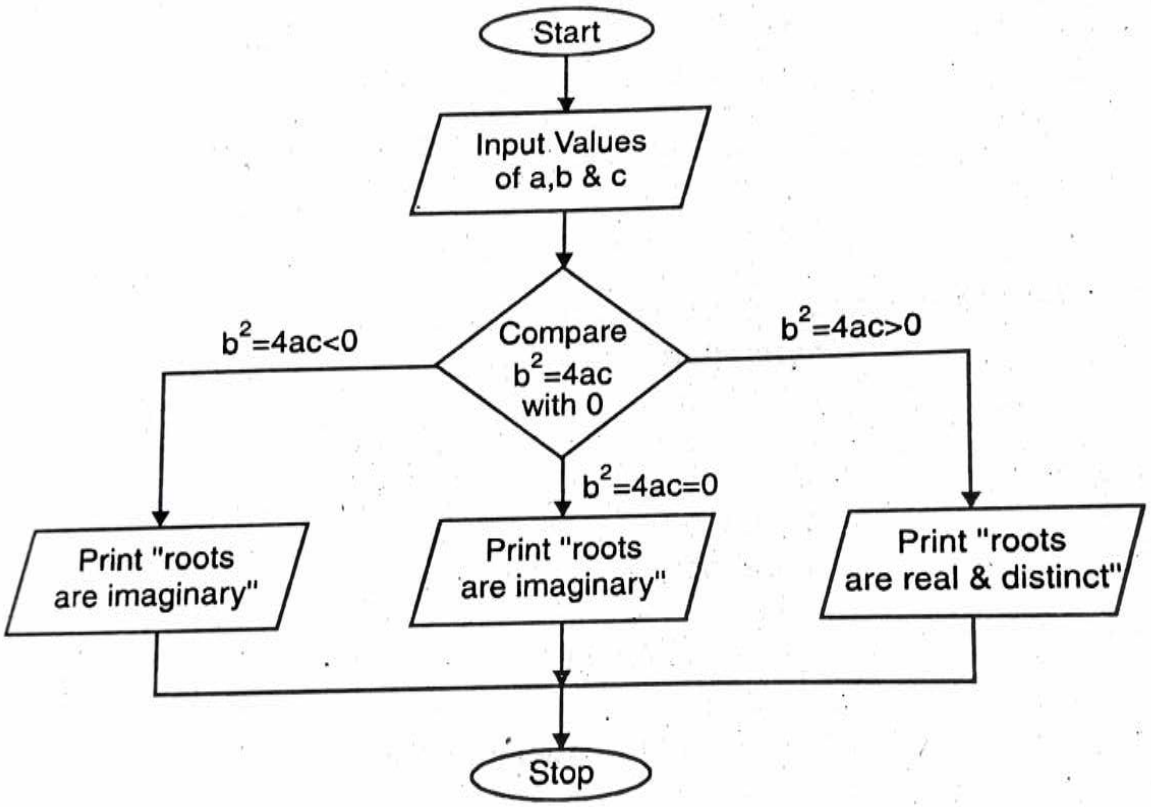
Q 81. What is flow chart? How it is different from algorithm? Explain with the help of example. (PTU, May 2008)

e.g. Write a algorithm and flowchart to find roots of quadratic equation.

Algorithm :

Step 1 : Input values of a, b and c.
step 2 : Compute $b^2 - 4ac$ and denote its value disc.
Step 3 : Is disc < 0? If yes then goto step 7 else goto step 4.
Step 4 : If disc > 0>? If yes then goto step 9 else goto step 5.
Step 5 : Output "roots are real and equal".
Step 6 : Goto step 10.
Step 7 : output "roots are imaginary."
Step 8 : goto step 10.
Step 9 : output "roots are real and distinct".
Step 10 : Stop.

Flowchart :



Q 82. Explain in detail different operators in C++. **(PTU, Dec. 2008 ; May 2008)**

Ans. Following are the different types of operators :

1. Arithmetic operators : Different arithmetic operators supported by C++ are

Symbol	Operation	Associativity
+	Addition	Left-to-right
–	Subtraction	Left-to-right
*	Multiplication	Left-to-right
/	Division	Left-to-right
%	Modulus	Left-to-right

2. Relational operators : These operators are used to compare the values of the operands that must be compatible in order to facilitate decision making. These operators

Symbol	Operation	Associativity
<	less than	Left-to-right
< =	Less than or equal to	
>	Greater than	
> =	Greater than or equal to	
= =	Equal to	
! =	Not equal to	

3. Logical Operators : These operators are used to form compound conditions by joining too or more conditions formed using relational operators.

Symbol	Operation	Associativity
!	Logicat NOT	Left-to-Right
& &	Logical AND	
: :	Logical OR	

4. Bitwise : These operators operate at bit-level and allow us to manipulate the individual bits.

Symbol	Operation	Associativity
>>	Shift-right	Left-to-right
<<	Shift-left	
~	1's complement	
&	Bitwise AND	
^	Exclusive OR	
|	Inclusive OR	

5. Special Operators : These operators are :

Symbol	Operation
++	Increment
– –	Decrement
size of	size of
&	address of
*	Indirection
? :	Conditional
: :	Scope resolution
,	Comma

Q 83. Write a program in C++ to multiply two matrices. **(PTU, Dec. 201**

Ans. #include<iostream.h>

```
cout<<"\n Multiplication is not possible.... ";
goto ab;
}
cout<<"\n Enter the elements of first matrix :-";
for(i=0;i<r1;i++)
for(j=0;j<c1;j++)
cin>>a[i][j];
cout<<"\n Enter the elements of second matrix:";
for(i=0;i<r2;i++)
for(j=0;j<c2;j++)
cin>>b[i][j];
for(i=0;i<r1;i++)
for(j=0;j<c2;j++)
{
c[i][j]=0;
for(k=0;k<r2;k++)
c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
cout<<"\n The multiplication is:";
for(i=0;i<r2;i++)
{
cout<<"\n";
for(j=0;j<c2;j++)
cout<<c[i][j];
}
ab:
getch();
}
```

Q 84. Write a program in C++ to find the sum of first 100 natural numbers.
(PTU, Dec. 2007)

Ans.

```
#include <iostream.h>
void main ( )
{
        int i, sum = 0 ;
        for (i = 1 ; i < = 100 ; ++i)
        {
                sum + = i ;
        }
        cout <<"Sum of first 100 natural numbers is" ;
        cout <<sum ;
}
```

Q 85. What is the role of switch statement in C++? **(PTU, Dec. 2007)**

Ans. The switch statement provides an alternative to else if construct. The switch statement has more flexibility and clear format than else if construct. Its syntax is

```
switch (expression)
{
        case val-1 ;
              statement-1
              break ;
        case val-2 ;
              statement -2
              break ;
        :
        case val-n ;
              statement-n
              break ;
        default ;
              statement-d
        }
```

If expression takes any value from val-1, val-2 .... val-n, the control is transferred to tha appropriate case.

In each case, the statements are executed and then the break statement transfers the contro out of switch statement. If no break statement is used following a case, except the last one in th absence of default keyword. If any value of the expression does not match any of the case values control goes to the default keyword, which is usually at the end of the switch statement.

e.g.

```
#include <iostream.h>
void main ( )
{
       int day ;
       cout <<"Enter day of week as member :" ;
       cin >> day ;
       switch (day)
       {
            case 0 ;
                 cout <<"Day is sunday" ;
                 break ;
            case 1 :
                 cout <<"Day is Monday" ;
                 break ;
            case 2 :
                 cout <<"Day is Tuesday" ;
                 break ;
            case 3 :
                 cout <<"Day is Wednesday" ;
                 break ;
            case 4 :
                 cout <<"Day is Thrusday" ;
```

```
        case 6 :
            cout <<"Day is Saturday" ;
            break ;
        default :
            cout <<"Wrong Input" ;
        }
    }
```

Q 86. Explain the syntax of nested if statement in C++ with suitable examples.

Ans. If statements can be nested i.e. an if statement can be contained within another if statement. The inner if statement will be executed if the condition of the outer if statement evaluates to non-zero value. e.g. **(PTU, Dec. 2007)**

```
#include <iostream.h>
void main ( )
    int no ;
    cout <<"Enter any no" ;
    cin >> no ;
    if (n > 0)
    {
        cout <<"no is positive" ;
        if (no %2 = = 0)
            cout <<"and even" ;
    }
}
```

If you enter positive even number, then this program prints the message "No. is positive and even". But if you enter positive and number, it prints the message "No is positive". However, if you enter a negative number, the program prints nothing.

Q 87. What is difference between library and user defined functions. **(PTU, May 2007)**

Ans. Many activities in C++ are carried out by library functions.Some functions are written and incorporated into C++ language. Rather that each user write these subprorams, C++ compiler provide these built library functions as a convenience. There are many library functions that perform specilized functions and are defined in different header files. e.g. library functions that manipulate strings are defined in string.h, that perform mathematical functions are defined in math.h etc.

User-defined functions : User defined function is independent module that will be called to do designated task. A called function receives control from calling function-when the called function completes its task, it returns control to the calling function. It may or may not return a value to the calling function. C++ library cannot contain all the functions used in all application areas. However, it is possile for users to write their own function whenever such a need arises.

Q 88. What are strings? How are they different from normal character and variables? **(PTU, May 2007)**

Ans. Strings are array of characters. Using array of characters, we can store string data. Strings are not directly supported in C++ language, that is why we refer array of characters as strings. A string in C++ language is a variable length array of characters that is delimited by null character ('\0'). C++ does permit the use of any character except the null character. It is common to use such as tabs, format specifiers etc. in strings.

Normal character : For storing normal character we require only one storage location (1 – byte) where as even for storing one character string we require two storage location (1 – byte each) ; one for character and one for the delimited. For normal character, C++ does not require delimiter but for strings it sobers delimiter also.

Variable : Variable provide us with named storage that we can write to retrieve or manipulate. In other words, variables are memory locations in the computer memory that holds data. Contents of variable may vary. Variables hold different kind of data and the same variable might hold different values during execution of a program. Each variable in C++ is associated with specific type, which determines the size and layout its associated memory. For storing strings, we need to declare a variable array of type character.

Q 89. What is an array of characters? Where are they used? **(PTU, Dec. 2006)**

Ans. Array of characters is actually a string variable using array of characters, we can store string data strings are not directly supported in C++ language so we refer to array of characters as strings. A string is C++ is a variable length array of characters that is delimited by null character ('\0'). Characters comprising a string are selected only from printable characters. C++ does permit the use of any character except the null character. It is common to use formatting characters, such as tabs, format specifiers etc. in strings.

When we want to store more than one character in a variable then we use array of characters or string e.g.

```
char str1 [ ] = "Hello" ;
```

Here s + r1 is array of characters or string variable that stores "Hello".

Q 90. What is structure? How can the structure be nested? **(PTU, Dec. 2006)**

Ans. A structure is a collection of different data items of different data types. These data items occupy contiguous memory locations. Some variables can be int, some can be float and so on. The data items in a structure are called the members of the structure syntax for declaring a structure is

```
struct student
{
    .....
    field list
    .....
};
```

The declaration starts with the keyword struct.

The next element is tag that is identifier for the structure and list of variables or data items to be included and declaration conclude with semicolon after the closing brace.

e.g.
```
struct student
{
    int roll no ;
    char name [20] ;
    char grade ;
};
```

This structure tag is student containing three data items, roll no of integer type, name is string variable and grade of character type.

Nested structure : A structure can be nested i.e. there can also be structures that contain other structures as its elements. This is powerful way to create complex data types. The only restriction on nesting of structures is that a structure cannot contain a member that is itself a structure of same type as the outer structure.

e.g.

```
struct student
{
    int roll no ;
    char name [20] ;
    struct date
    {
        int day ;
        int month ;
        int year ;
    }   date of birth ;
    char grade ;
} ;
```

Here structure student is acting as a nesting (outer) structure, whereas structure date is acting as nested (inner) structure. The date structure don't have its independent existence, it exists only in the scope of student structure. Individual elements of a nested structure are accessed by referring to outer structure variable name.

Q 91. What is void *pointer? What is its use? **(PTU, Dec. 2007, 2006)**

Ans. A void pointer is used to hold address of operands of different data types at certain times. These are also known as generic pointers. Syntax is

```
void *ptr ;
```

A void pointer cannot be directly de-referenced. They need to appropriately type casted and pointer arithmetic cannot be performed on void pointers. That is why compiler cannot determine the size of the operand whose address is held in void pointer.

e.g.

```
#include <iostream.h>
void main ( )
{
void *ptr ;
int a = 3 ;
float b = 5.3 ;
ptr = &a ;// assign address of integer variable
cout <<"Value pointed to void pointer is"
     << (* (int *) ptr) ;
ptr = & b;// assign address of float variable
cout <<"Value pointed to void pointer now is"
     << (*float *) ptr) ;
}
```

In above example we declared void pointer and one integer variable a and float variable b. Firstly, we assign the address of integer variable a and then display the value of this variable using pointer and then assigned floating variable b to pointer and again displayed the value of b using void pointer.

Hence, void pointer is used to hold address of any type of variable at any time in program.

Q 92. Write a program in c++ to find the maximum and second largest number in a given series of data. **(PTU, Dec. 2005)**

```
{
    int n, i, max1, max2, a [20] ;
    cout <<"Enter length of series" ;
    cin >> n;
    cout <<"Enter series" ;
    for (i = 0 ; i < n ; ++i)
{
    cin >> a [i] ;
}
    max = a [0] ;
    for (i = 1 ; i < n ; ++i)
{
    if (a [i] > max1)
    max 1 = a[i] ;
}
    max2 = 0 ;
    for (i = 0 ; i < n ; ++i)
{
    if (a [i] < max 1 & & a [i] > max2)
    max2 = a[i] ;
}
cout <<"Maximum no. is" << max1 <<" second largest"
    <<max2 ;
}
```

Q 93. Explain 'REGISTER' storage class with some example.

(PTU, Dec. 2005, 2004)

Ans. In register storage classes, the visibility and lifetime of the variable is limited to the block in which it is declared. Compiler assign register variable to one of the processor's register instead of storing it in the memory. Value stored in register can be accessed must faster than the value stored in memory. But there are limited number of registers in a processor, and not all variables can be register variables. Features of register are :

1. Stored in register, if a register is available. If no register is available, the variable is stored in memory and works as if it is auto.
2. Local to the block in which the variable is declared.
3. If not initialized in the declaration, their initial value is unpredictable.
4. It retains its value till the control remains in the block in which the variable is declared.

e.g.

```
#include <iostream.h>
void main ( )
{
    register int a ;
    cout << "Enter any number " ;
    cin >> a
    cout <<"Number is" <<a;
}
```

Q 94. What is enumerated data type? Explain with example. **(PTU, Dec. 2005)**

Ans. The enumerated data type is a user-defined data-type based on the standard integer value

is given an identifier called enumeration constant. We can use enumerated constants as symbolic names.

To declare an enumerated data type, we must declare its identifier and its values. Since enumerated data type is derived from the integer data type, its operations are the same as for integers.

The syntax for declaring an enumerated data type is

enum typename {identifier list} ;

The keyword enum is followed by an identifier which is followed by an identifier list enclosed in a set of braces, and finally terminated by semicolon. The identifier list contains enumeration identifiers separated by commas.

Each enumeration identifier is assigned an integer value. If we do not explicity assign the values, the compiler assign value 0 to the first enumeration identifier, value 1 to the second enumeration identifier and so on until all of the enumeration identifier have a value.

e.g. enum color {RED, ORANGE, GREEN, BLUE} ;

the color type has four possible values, the range being 0-3 with the RED enumeration identifier representing value 0 and ORANGE enumeration identifier representing the value 1, GREEN enumeration identifier representing the value 2 and finally the BLUE enumeration identifier representing the value 3.

Q 95. Write a program in C++ language to fit the maximum in a give series of data.
(PTU, Dec. 2004)

Ans.
```
#include <iostream.h>
void main ( )
{
      int a [20], i, max ;
cout <<"Enter series" ;
for (i = 0 ; i < 10 ; ++i)
      cin >>a [i] ;
cout <<"Enter maximum no" ;
cin >> max;
i++ ;
a [i] = max ;
cout <<"series of no.'s is " ;
for (i = 0; i < 11; ++i)
      cout << a [i] ;
getch ( ) ;
}
```

Q 96. How will you replace a while loop statement with for loop statement?
(PTU, May 2004)

Ans. The while loop looks like a simplified version of the for loop. It contains a test expression but no initialization or increment expressions. Syntax of while loop is :

```
While (condition)
{
      statement ;
      statement ;
      ..............
}
```

As long as the condition is true, the loop continues to be executed.

Where as for loop executes a section of code a fixed number of times. It contains initializations, test expression and increment expression. Syntax of for loop is

```
for (initialization, test condition, increment/decrement)
{
        statement ;
        statement ;
        ..............
}
```

Example :

a while loop to be replaced by for loop.

(i) Example of while loop.

```
main ( )
{
        int i = 0 ;
        while (i < = 10)
{
        cout <<i ;
        i++;
        }
}
```

(ii) Above example replaced by for loop :

```
main ( )
{
int i ;
for (i = 0 ; i < = 10 ; i + +)
{
cout < < i;
}
```

In example (i) variable i is initialized outside the while loop and condition is specified with while loop and increment expression is written within the loop. Where as in for loop all the three initialization, condition and increment are specified with the loop.

Q 97. What are user defined data types? **(PTU, Dec. 2009)**

Ans. There are some derived data types that are defined by the user. These are :

Class, Structure, Union and Enumeration. These are explained as follows :

1. Class : A class represents a group of similar objects. To represent classes in C++, it offers a user defined data type called class. Once a class has been defined in C++, objects belonging to that class can easily be created. A class bears the same relationship to an object that a type does to a variable.

2. Structure : A structure is a collection of variables referenced under one name, providing a conventional means of keeping related information together. The following example creates a structure

3. Union : A union is a memory location that is shared by two or more different variables, generally of different types at different times. Following declaration defines union :

```
union share {
        int i ;
        char ch ;
        }
union share cnvt ;
```

4. Enumeration : An alternative method for naming integer constants is often more convient that const. This can be achieved by creating enumeration using keyword enum. For example ;

```
enum {START, PAUSE, GOD} ;
```

defines three integer constants, called enumerators and assigns values to them.

Q 98. List and explain different data types used in C++. **(PTU, May 2019 ; Dec. 2009)**

Ans. Data types are means to identify the type of data and associated operations of handling it.

C++ data types are of two types :

(i) Fundamental types

(ii) Derived types.

There are five fundamental data types in C++ ; char, int, float, double and void that represent character, integer, floating point, double floating-point and value less data respectively.

Derived types constructed from the fundamental types are : arrays, functions, pointers, references, constants, classes, structures, unions and enumerations.

Fundamental Data Types :

int : Integers are whole numbers such as 5, 39 -1917, O etc. They have no fractional part.

Char : Characters can store any member of C++ implementations basic character set. If a character from this set is stored in character variable, its value is equivalent to the integer code of that character. An identifier declared as char becomes a character variable.

float : A number having fractional part is called a floating point number. Floating point numbers have two advantages over integers. First, they can represent values between the integers. Second, they can represent a much greater range of values.

Derived Data Types :

1. Arrays : Arrays refer to the named list of finite number n of similar data elements. Arrays are represented as below :

```
ARY [0], ARY [1], ARY [2], ..... ARY [9]
```

Array can be one dimensional, two dimensional or multidimensional.

```
Ex : float a [3] ;
     int b [2] [4] ;
```

2. Functions : A function is a named part of a program that can be invoked from other parts of the program as often needed.

```
Ex : #include <iostream.h>
     #include <conio.h>
     float cube (float) ;
     int main ( )
     { clrscr ( );
     float num ;
     count <<"Enter a number" ;
     cin >> num ;
     count <<"\n" <<"the cube of" <<num>> "is" ;
```

```
cout <<cube (num) <<"\n" ;
return 0 ;
}
float cube (float a)
{ return (a* a* a) ;
}
```

3. Pointers : A pointers is a variable that holds a memory address. This address is usually the location of another variable in memory. If one variable contains the address of another variable, the first variable is said to point to the second.

Ex : type * ptr ;

4. Reference : A reference is an alternative name for an object. A reference variable provide an alias for a previously defined variable.

The general form of declaring a reference variable is : type & ref-var = var – name ;

5. Constant : The keyword const can be added to the declaration of an object to make that object a constant rather than a variable.

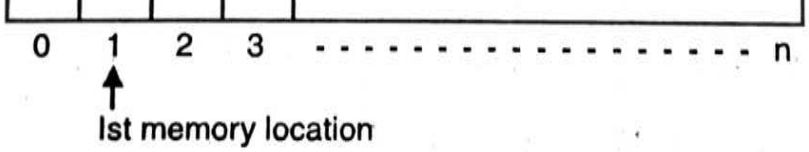
const type name = value ;

Q 99. What are arrays? Explain the concept of arrays with the help of a program. **(PTU, May 2019 ; Dec. 2011**

OR

Illustrate the concept of arrays with the help of a program. **(PTU, Dec. 2009**

Ans. An array is a collection of variables of same data type that are referenced by a common name. In C++ array consists of a contiguous memory locations. The lowest address corresponds to the first element and the highest element corresponds to the last element.



Memory representation of arrays

Types of Arrays : Arrays are of different types :

(i) One-dimensional arrays (ii) Multidimensional arrays.

(i) Single dimensional arrays : It consists of finite homogeneous elements type array name [size];

Example :

```
#include <iostream.h>
#include <conio.h>
    int main ( )
{
    clrscr ( );
    const int size = 3;
    float sales [size], avg = 0, total = 0;
    for (int i = 0 ; i < size ; i ++)
```

```
{   cout <<"Enter sales made on day" <<i+1 << ":" ;
    cin >> sales [i] ;
    total + = sales [i] ;
}
avg = total/size ;
cout <<"\n Total sales = "<<total<<"\n" ;
cout <<"Average sales = "<<avg<<"\n" ;
return 0 ;
}
```

(ii) Multidimensional arrays : C++ allows you to have arrays with dimensions more than two. The maximum limit of dimensions is compiler dependent. The general form of a multidimensional array declaration is

type name [a] [b] [c] ..... [z]

Two-dimensional arrays are also initialized in the same way as single-dimensional ones. For example,

```
int cube [5] [2] = {  1, 1
                      2, 8,
                      3, 27,
                      4, 64,
                      5, 125}
```

Example :

```
#define N3
#define N4
#include <iostream.h>
void main (void)
{int i, j ';
float a [N] [M] =  {
                   {1, 2, 3, 4},
                   {5, 6, 7, 8},
                   {9, 10, 11, 12}
                   } ;
cout <<"contents of the array" <<endl ;
for (i = 0 ; i < = N – 1 ; ++i)
for (j = 0 ; j < = M – 1 ; ++j)
    cout << a [i] [j] << '\t' ;
    cout <<endl ;
}
}
```

Q 100. Why is it necessary to include Header files in C++ Program?

Ans. Header files contain declarations for various functions that are predefined in C++. For

1. Prepare a flowchart of the program.
2. After that prepare a detailed list of steps, called algorithm.
3. Finally, translated these steps in a computer program.
4. The next phase is to translate the program into machine language.This is known as compilation and then finally execute this machine language code or executable code of your program using RUN command.

Q 102. Give the precedence of C++ operators.

Ans. The precedence of C++ operators from low greatest to lowest is as follows :

S.No.	Operator	Description	Associativity
1.	: :	Scope Resolution operator	Left-to-Right
2.	( )	Paranthesis	Left to Right
	[ ]	Array subscript	
	.	Member Selection via object name	
	→	Member seclection via pointer	
	+ +	Unary Post increment	
	– –	Unary Post decrement	
3.	+ +, – –	Unary Preincrement/Predecrement	Right to left
	+, –	Unary plus/minus	
	*, &	indirection and reference	
4.	(Type)	Type Casting	Left-to-Light
5.	*, 1, %	Multiplication/Division/Modulus	Left-to-Light
6.	+, –	Addition/Subtraction	Left to Right
7.	<<, >>	Bitwise Shift	Left to Right
8.	<, >, < =, > =	Relational	Left to Right
9.	= = , ! =	equality	Left to Right
10.	& &	logical AND	Left to Right
11.	ll	Logical OR	Left to Right
12.	? :	Conditional	Right to Left
13.	= , * =, I =, % =, – =	Assignment	Right to Left
14.	,	Comma	Left to Right

In the above Table, Associativity defines the precedence order in which operators are evaluated in the case that there are several operators of the same level in an expression.

Q 103. What is the difference betwen syntax and semantics of a language?

Ans. Syntax : It refers to the ways symbols may be combined to create well-formed in the language. Syntax defines the formal relations between the constituents of a language, thereby providing a structural description of the various expressions that make up legal strings in the language. Syntax deals solely with the form and structure of symbols in a language without any consideration given to their meaning.

Semantics : It reveals the meaning of syntactically valid strings in a language. For natural languages, this means correlating sentences and phrases with the objects, thoughts and feelings of our experiences. For programming languages, semantics describes the behaviour that a computer follows when executing a program in the language.

Q 104. What is a user defined data type?

Ans. The user defined data type enables a programmer to invent his/her own data type and define what values it can take on. This can help more listings more readable, in the case of a complicated program or when more than one programmer works on it. Thus this data type can help a programmer reduce programming errors.

Q 105. What is a pre-incremental and post-incremental operators?

Ans. A post-incremental operator is one in which value is used first and then incremented whereas a pre-increment is one in which value is first incremented and then assigned.

```
int i, j, k;                    int i, j, k;
i = 10;                         i = 10;
j = 2;                          j = 2;
k = j + (i ++);                 k = j + (++i);
cout <<k<<i;                    cout<<k<<i;

The output is :                 The output is :
k = 12                          k = 13
i = 11                          i = 11
i ++ is post incremented
++ i is pre incremented
```

Q 106. What is application of scope resolution operator in C++?

Ans. Scope resolution operator is (: :). It is used to specify functions outside the class. We can just declare function inside the class and then definition of the function can be defined outside the class using the scope resolution operator.

e.g. :
```
void abc : : xyz (abc x, abc y)
{
}
```
Here abc is a function of xyz class.

Q 107. How elements in an array are stored in memory?

Ans. The declaration

```
int a[10] ;
```

Declares an array, named **a**, consisting of ten elements, each of type **int.** Simply speaking, an array in a variable that can hold more than one value.

You specify which of the several values you're referring to at any given time by using a numeric subscript. We can represent the array a above with a picture like

a :

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Here the ten elements of a 10-elements array are numbered from 0 to 9 and are stored seqentially. The subscript which specifies a single element of an array is simply an integer expression in square brackets.

Q 108. What do you mean by operand and operators? **(PTU, Dec. 2011, 2009)**

Ans. Operand : Operand on which operation is applied.

Operator : Operator which specifies what operation is done on operands.

e.g. $2x^2 + 3x - 5$

Here $2x^2$, 3x and 5 are operands and +sign and –sign is operator.

Q 109. Explain the structure of C++ program with following points

(a) Built in functions

(b) Symbolic statements

(c) Special operators

(d) Various data types.

Ans. (a) Built in Function : There are certain set of general purpose operation which are quite frequently used by many programmers in their program. For example the square root of number, to calculate power of a number and many more. Making functions for performing these operations in every program is an unnecessary and time consuming job. So there general purpose operation are programmed and stored in C++ library so that they can be called through any program in the form of function. These functions are called in Built Functions. There are number of library functions available in C++.

e.g. Program to find square root of number

```
#include <iostream.h>
#include <math.h>
int main ()
{
    int n, x ;
    cout << "Enter any number" ;
    cin >> n ;
    x = sqrt (n) ;
    cout <<"Square root of" << x ;
    getch () ;
    return 0 ;
    }
```

(b) Symbolic Statements : Symbolic statements are preprocessor directives that is #define directive. The symbolic constants can be specified using # define directive. The symbolic constants allow the programmer to define a name for constant and use that name throughout the program as shown in the following example :

```
#define PI 3.14
```

It replaces every occurrence of symbolic constant PI with a value 3.14 before the program is compiled. The main advantage of using it is that to change the value of symbolic constant PI, it needs to be modified only once in #define directive.

(c) Special operators : These operators are used in pointers, structures and union etc. Some special are unary operator, comma operator, size of operator, type operator etc.

(d) Various data type : Integer, char, double, float etc. are various data types available in C++.

e.g. program in C++ to show use of above points.

```
# define PI 3.14 // symbolic statement
#include <iostream.h>
#include <math.h>
void main ( )
{
    double number, answer; //special comma operator
    cout <<"Enter number" ; // double data type
    in >> number ;
    answer = sqrt (number) ; //sqrt is built in function
```

```
        cout << "Square root" << answer ;
        answer = PI *number* number ;
        cout << "Area is" << answer ;
        getch ( ) ;
}
```

Q 110. What is the difference between an algorithm and a flowchart? (PTU, May 2012)

Ans. 1. **Algorithm** is a sequence of instructions used to solve a particular problem. **Flowchart** is a tool to document and represents the algorithm.

2. An algorithm can be represented using flowchart but flowchart can't.

3. Flowchart is a graphical representation of algorithm.

Q 111. Write a program in C++ to find whether a number is prime or not? (PTU, Dec. 2011)

Ans.
```
#include <iostream.h>
#include <process.h>
void main ( )
{
long n (j) ;
cout <<"Enter a number" ;
Cin >> n;
for (j = 2, j < n/2 ; j++)
if (n%j = = 0)
{
cout <<"its not prime" ;
exit (0) ;
}
cout <<"its prime" ;
}
```

Q 112. Distinguish between static members and not-static variables? How are they useful? (PTU, May 2018, 2014)

Ans. Class members can be declared using the storage class specifier static in the class member list. Only one copy of the static member is shared by all objects of a class in a program. When you declare an object of a class having a static member, the static member is not part of the class object.

A typical use of static members is for recording data common to all objects of a class. For example, you can use a static data member as a counter to store the number of objects of a particular class type that are created. Each time a new object is created, this static data member can be incremented to keep track of the total no. of objects. You access a static member by qualifying the class name using the :: (scope resolution) operator.

A static member belongs to a class where as a non-static member belongs to an object of a class. This means that a static member can be called by a class and by the object of the class, whereas a non-static member can be called only by the object of the class. A static member can access only other static members of a class, where as a non-static members can access both static and non-static member of a class.

There is only one copy of a static variable and its value remains the same even when the class is instantiated, whereas in case of non-static variables, every time the class is instantiated, the objects have their own copy of these variables.

Q 114. How is the operator keyword used? **(PTU, Dec. 2013)**

Ans. The keyword operator is used to overload the ++ operator in this declarator

void operator ++ ()

The return type (void in this case) comes first, followed by the keyword operator, followed by the operator itself (++), and finally the argument list enclosed in parenthesis (which are empty here).

This declaration syntax tells the compiler to call this member function whenever the ++ operator is encountered, provided the operand (the variable operated on by the ++) is of type counter.

Q 115. What is meant by classes within the class? How can we declare a class and object? Explain with help of example.

Ans. From the classes within the class we mean inheritance. In C++, classes are reused via the concept of inheritance. When a new class is created by inheriting an existing class, the new class becomes a derived class of existing one. The existing class is the base class of the new class. The derived class can define additional data and methods. If the base class has a method that the derived class wants to change, it can do it.

Class is blueprint for an object. A class is an expanded concept of a data structure : instead of holding only data, it can hold both data and functions.

An object is a bundle of variables and related methods. An object is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

Classes are generally declared using the keyword class, with the following format :

```
class class_name
{
    access_specifier_1:
    member1;
    access_specifier_2:
    member2;
    .....
}   object_names;
```

Where class_name is a valid identifier for the class, object_names is an optional list of names for objects of this class. The body of the declaration can contain members, that can be either data or function declarations, and optionally access specifiers.

For example :

```
Class Student
{
    int age;
    char name[20];
    float per;
    average();
}
```

When class is bind Objects and methods then it is called as encapsulation.

Q 116. What are the rules for static data members?

Ans. Rules for static data members are as follow :

1. They can be public as well as private.
2. They are initialized in a special manner. If you forget to initialize them, you will get a compile-ti

```
getch ( );
return 0;
}
```

Q 121. Write a program to pass and return object from the function.

Ans.

```
#include <iostream>
class complex
{
private:
int real;
int imag;
public;
complex ( ) : real (0), img (0); { }
void read Data ( )
{
Cout <<"Enter real and imaginary number respectively:" <<endl;
Cin >> real >> imag;
Complex and complex Number (Complex Comp2)
{
Complex temp;
temp. real = real + comp2. real;
temp. imag = imag + comp2. imag;
return temp;
}
void display Data ( )
{
cout <<"Sum = " <<real<<"+" <<image<<"i";
}
};
int main ( )
{
Complex C1, C2, C3;
C1.readData ( );
C2.readData ( );
C3 = C1. add complex Numbers (C2);
(C3.display Data ( );
return 0;
}
```

Q 122. How the objects are assigned using C++ ?

Ans. If a temporary object is created on the right hand side of the expression, the standard constructor for that object is called to initialize the temporary object, then the copy assignment operator is called on the object being assigned to, with the temporary object.

– static
– extern.

❑ **Auto :** The auto storage class is the default storage class for all local variables.

```
{ int a
auto int b;
}
```

auto can only be used within functions i.e. local variables.

❑ **Register :** The register storage class is used to define local variables that should be stored in a register instead of RAM.

```
{ register int a;
}
```

The register should only be used for variables that require quick access such as counters.

❑ **Static :** The static storage class instructs the compiler to keep a local variable in existence during the life time of the program instead of creating and destroying it each time. It comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function cells.

❑ **Extern :** The extern storage class is used to give a reference of a global variables that is visible to all the program files. When you use 'extern', the variable cannot be initialized as all it does. It point the variable name at a storage location that has been previously defined.

Q 124. Write a program to get character input from the user and store those characters in a file. **(PTU, May 2018)**

Ans.

```
#include <stdio.h>
int main (void)
{FILE, *fptr;
char ch;
fptr = fopen ("username.txt,", "w");
printf ("Enter your name");
While ((ch = getchar()) ! = /n') {
putc (ch, fptr);
}
fclose (fptr);
fopen ("username. txt", "r");
printf ("\n file content:\n");
While ((ch = get c((fptr))! = EOF)
}
printf ("%C", ch);
}
printf ("\n End of file \n");
fclose (fptr);
return 0;
}
```

Q 125. What are the different ways in which we can have abstraction in Object Oriente Programming? Explain each with the help of an example. **(PTU, Dec. 201**

Ans. There are various ways of achieving abstraction in object oriented programming langua C++. One approach is to take modular based code that is broken apart into smaller segments, kno as functions. This functional or modular approach helps the code to be reused again and again wh

needed. For example, a programmer might write a function for computing salary. These fu
programmer might write a function for computing salary. These fu
by anyone. The modular based approach helps to centralize all data of a
control of a type module. Defining module types allow the module to be an abstract data ty
other programming language, there is a small drawback associated with the approach to accessing
module type. In C++, this problem is resolved through the concept of friend functions. The concept of
abstraction brings forth another related term known as encapsulation. Encapsulation is the process of
combining or packaging data with functions and thereby allowing users to create a new data type.
This new data type is termed abstract data type. Though the new data type is similar to that of built-
in data type, it is termed abstract data type because it enables users to abstract a concept from the
problem space into the solution space.

Q 126. What do you mean by an array of objects? Explain how members of objects can be accessed in array of objects with the help of C++ program. **(PTU, Dec. 2014)**

Ans. Arrays of Objects : We can define an array of objects too. However, as objects are required to have constructors associated with them, generation of an array of objects is more specific as compared to arrays of other data type. Once properly constructed, an array of objects can be handled exactly like any other array. Any object, whether built-in or user-defined, can be stored in an array. When you declare the array, you tell the compiler the type of object to store and the number of objects for which to allocate room. The compiler knows how much room is needed for each object based on the class declaration. The class must have a default constructor that takes no arguments so that the objects can be created when the array is defined.

S	A	C	H	I	N	T	E	N	D	U	L	K	A	R

An array of character

Accessing member data in an array of objects is a two-step process. You identify the member of the array by using the index operator ([ ]), and then you add the member operator (.) to access the particular member variable.

Like :

```
int x[10]; // array of 10 integer data types
float y[12]; // array of 12 float data types
child c(5); // array of 5 child class objects
#include<iostream.h>
class CHILD
{
public :
CHILD( ) { itsAge = 1; itsWeight = 5; }
~CHILD( ) { }
int GetAge( ) const { return itsAge; }
int GetWeight( ) const { return itsWeight; }
void SetAge(int age) { itsAge=age; }
private
```

```
int i;
for(i=0; i<5;i++)
suhani[i].SetAge(2*i+1);
for(i=0; i<5; i++)
{
cout <<"Child #" <<i+1<<":";
cout <<suhani[i].GetAge( )<<endl;
}
}
```

Q 127. Why principle of substitution cause slicing problem? What are the different ways to avoid it? **(PTU, Dec. 2014)**

Ans.Substitution principle : Everywhere a base class object is expected, you should be able to use a derived class object instead.

For example, consider this code :

Question MyQuestion;

Question bonus = MyQuestion;

The second line creates bonus as a copy of MyQuestion. By the substitution principle, we should be able to make MyQuestion a short-answer question which is derived from the Question class instead:

ShortAnswer sa;

Question bonus = sa;

There are problem of slicing. So what happen when we say Question bonus = sa;

- It invokes the copy constructor of Question
- We have sliced off things specific to ShortAnswer.

Slicing : The loss of information that occur when putting a derived-class value into a base-class variable.

Avoiding Slicing :

- Using base-class pointers
- Virtual methods

Q 128. Differentiate between encapsulation and Abstraction. **(PTU, May 2015)**

Ans. Abstraction : Abstraction allows us to represent complex real world in simplest manner. It is process of identifying the relevant qualities and behaviours an object should possess, in other word represent the necessary feature without representing the background details.

Encapsulation : It is a process of hiding all the internal details of an object from the outside real world. The word Encapsulation like enclosing into the capsule. It restrict client from seeing its internal view where behaviour of the abstraction is implemented.

Q 129. What do you mean by garbage collection ? **(PTU, May 2015**

Ans. Garbage collection is the systematic recovery of pooled computer storage that is being used by a program when that program no longer needs the storage. Thus free the storage for use by other programs (or processes with in a program). It also ensures that a program using increasing amounts of pooled storage does not reach its quota. Garbage collection is an automatic memory management feature in many modern programming languages such as Java and languages in the .NET framework. Languages that use garbage collection are often interpreted or run with in a virtual machine like the JVM. In each case, the environment that runs the code is also responsible for garbage collection.

Q 130. Differentiate between meta class and abstract class. **(PTU, May 201**

Ans. Meta class : The meta class is often described as the 'class of a class' which may seem a rather odd expression, but can be explained as follows :

The class, as we know holds the attributes and methods which will apply to objects of the class. It is the class of objects. The meta class holds the attributes and methods which will apply to the class itself therefore It is the class of the class. Every class has one metaclass and the metaclass contains those parts of a class which are not appropriate to be duplicated for every specific object.

Abstract class : An abstract class is a class for which one or more methods are declared but not defined meaning that the compiler knows these methods are part of the class, but not what code to execute for that method. These are called abstract methods.

Q 131. What do you mean by Precedence and Associativity ? **(PTU, May 2015)**

Ans. The C++ language includes all C operators and adds several new operators. Operators specify an evaluation to be performed on one or more operands. Operator precedence specifies the order of operations in expressions that contain more than one operator. Operator associativity specifies whether, in an expression that contains multiple operators with the same precedence ; an operand is grouped with the one on its left or the one on its right.

Q 132. Name various (at least four) standard classes of C ++. **(PTU, May 2015)**

Ans. The following are the various standard classes of C ++

(a) <array>
(b) <bi/set>
(c) <memory>
(d) <string>
(e) <istream>
(f) <ostream>
(g) <iomanip>

Q 133. Differentiate between the term Abstract class and container class.
(PTU, May 2015)

Ans. Abstract class and container class :

Abstract class : An abstract class is a class for which one or more methods are declared but not defined meaning that the compiler knows these methods are part of the class, but not what code to execute for that methods. These are called abstract methods.

Container class : Container class is a class which can hold objects of other clases. These classes are very helpful in managing resources required by the application. There is a rich collection of container classes supported by most of the modern C++ compiler in the form of standard Template library (STL). The STL not only contains container classes but also set of functions that can manipulate these container classes.

Q 134. What do you mean by the term Data type ? **(PTU, May 2015)**

Ans. Data type : A data type in a programing language is a set of data with values have predefined characteristics. Examples of datatypes are : Integer, floating point unit number, character, string and pointer. Usually, a limited number of such datatypes come built into a language. The language usually specifies the range of values for a given data types, how the values are processed by the computer and how they are stored.

Q 135. Explain the declaration, accessing and usage of static data members and static member functions with the help of suitable examples. **(PTU, Dec. 2015)**

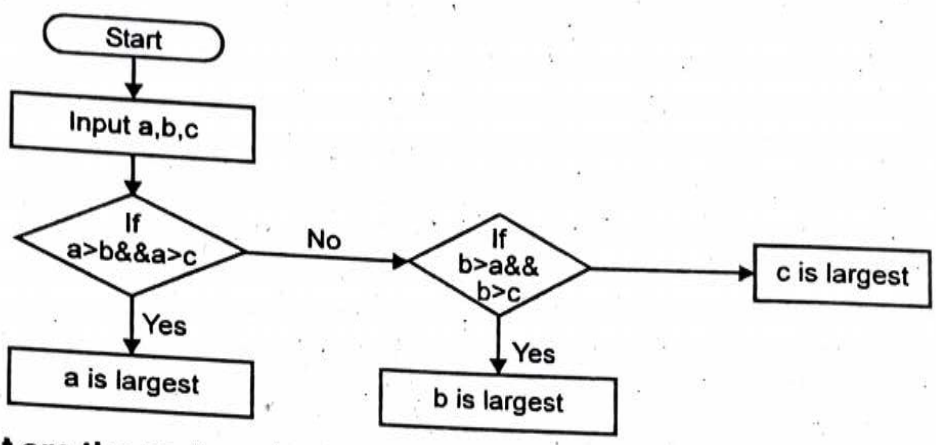
Ans. Static data members : Static data members are data objects that are common to all the objects of a class. They exist only once in all objects of this class. They are already created before the finite object of the respective class. The static members are used in information that is commonly accessible.

The main advantage of using a static member is to declare the global data which should be updated while the program lives in the memory.

background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost and functions to operate on these attributes.

Q 137. Draw flow chart to find the largest of three numbers. **(PTU, May 2016)**

Ans.



Q 138. What are the various input statements of C++ ? **(PTU, May 2016)**

Ans. When the computer gets the data from the keyboard, the user is said to be acting interactively. Putting data into variables using cin and the operator >>. The syntax of cin together with >> is

```
cin >> variable;
```

If two variables then

```
cin >> variable1 >> variable 2;
```

This is called input statement. In C++, >> is called the stream extraction operator

eg.
```
int feet;
int inches
```

then input is cin >> feet >> inches;

Q 139. Write a class to represent a vector (a series of float values). Include member functions to perform the following tasks.

(a) To create the vector

(b) To modify the value of a given element

(c) To multiply by a scalar value

(d) To display the vector in the form (10, 20, 30, ...)

Write a program to test your class.

Ans. **(PTU, May 2016)**

```
#include<iostream.h>
#include<conio.h>
int const size = 50;
class vector
{
float d [size];
int s;
public;
void create (void);
void modify (void);
void multiply (void);
void display (void);
};
```

```
void vector:: create (void)
{
cout <"\n\n Enter size of array you want to create";
cin >> S;
cout <<"Enter" <<S<<"Realnumbers\n";
For (int i = 0; i<S ; i ++)
cin >> d [i];
}
Void vector :: modify (void)
{
int mfy_value;
float with;
cout <<"\n Enter location of array at which value is to be modified";
cin >> mfy_value;
Cout << "Enter value with which you want to replace";
cin >> with;
d [mfy_value] = with;
}
void vector :: mutiply (void)
{
int mul;
Cout <<"\n Enter value with which you want to mutiply:";
Cin >> mul ;
for (int i = 0; i< S; i + +)
d [i] = d [i] * null;
}
void vector :: display (void)
{
cout <<"\n\nDisplay of Array \n";
cout <<"(";
for (int i = 0; i < S ; i + +)
{ Cout << d [i];
If (i ! = S – 1)
cout << ",";
}
cout <<")";
}
Void main ( )
{
clrscr ( );
Vector ol ;
int choice;
do
{
Cout <<"/n/n Choice list /n";
cout <<"1) To create vector Array \n";
```

```
cout <<"2) To modify array \n";
cout <<"3) To multiply with scalar values \n";
cout <<"4) To Display\n";
cout <<"5) Exit \n";
cout << "Enter your choice ";
cin >> Choice;
switch (choice)
{
Case 1 : ol. create ( );
break;
Case 2 : ol. modify ( );
break;
Case 3 : ol. multiply.( );
break;
Case 4 : ol. display ( );
break;
Case 5 : goto end;
}
} While (1);
end; }
```

Q 140. What is the difference between high level and a low level language ?
(PTU, Dec. 2016)

Ans.

Basis	High Level	Low level
. Learning	1. Easy to learn	1. Difficult to learn
. Execution	2. Progres are slow in execution	2. Programs are fast in execution
. Modification	3. Programs are easy to modify	3. Programs are difficult to modify
. Facility at hardware level	4. Do not provide much facility at hardware level	4. Provide facility to write programs at hardware level
. Uses	5. Normally used to write application programs.	5. Normally used to write hardware programs.

```
{
Hanoi (m – 1, a, c, b);
Cout <<"Move disc"<<m<<"from"<<a<<"to"<<C<<endl;
Hanoi (m–1, b, a, c);
}
}
int main ()
{
int discs;
cout <<"enter the number of discs"<<endl;
cin >>discs;
Hanoi (discs, 'A', 'B', 'C');
Cout <<"It took" <<moves<<"moves,"<<endl;
system ("Pause");
}
```

Q 142. Describe the following manipulators :

(a) Setw( ) **(b) Setiosflags** **(c) Setprecision**

(d) Setfill ( ) **(e) Resetiosflags ( )** **(PTU, Dec. 2016)**

Ans. (a) Setw ( ) : The setw ( ) stands for the set width. It sets the number of characters to be used as the field width for the next insertion operation. The setw( ) manipulator is used to specify the minimum number of character positions on the output field a variable will consume.

(b) Setiosflags : The setiosflags manipulator function is used to control different input and output settings. The I/O stream maintains a collection of flag bits.

(c) Setprecision : The setprecision ( ) is used to control the number of digits of an output stream display of a floating point value.

(d) Setfill ( ) : The setfill ( ) manipulator function is used to specify a different character to fill the unused field width of the value.

(e) Resetiosflags ( ) : The resetiosflags manipulator performs the same function as that of the resetios function. All those flags which are set by the setiosflags manipulator are reset by this function.

Q 143. What is the value of the following expression ?

(!((4 – 4% 3) <5 & & (6/4 > 3))) **(PTU, May 2017)**

Ans. = (! (3 < 5 & & 1>3))

= (! (True & & False))

= ! (False)

= True.

Q 144. What is meant by membership operator in C++ ? **(PTU, May 2017)**

Ans. The membership operator is used to access the member of structure variable or class object through the pointer. It is common to dynamically allocate structure, so this operator is commonly used.

Membership operators are . & →.

Q 145. What is a nested class ? What are its advantages ? How is it defined and declared in C++ ? **(PTU, May 2017)**

Ans. A nested class is a class which is declared in another enclosing class. A nested class is a member and as such has the same access rights as any other member. The members of an enclosing class have no special access to members of a nested class; the usual access rules shall be obeyed.

Advantages of nested class :

- The purpose of nested class is to hide the nested class name inside another class, restricting access to that name.
- It does not give either the outer class or the nested class any special privileges to access protected or private members of either class.
- A nested class does not allow the outer class access to its hidden members. The name of the nested class is hidden.

Declaration and Definition of nested class

```
#include <iostream.h>
class Nest
{
public :
class Display
{
private :
int s;
public :
void sum (int a, int b)
{ S = a + b; }
void show ( )
{
cout <<"/n The sum of a and b is" <<S ; 3
};
};
void main ( )
{
Nest : : Display X;
X.Sum (12, 10);
X. show ( );
}
```

Q 146. Call by value. **(PTU, May 2019**

Ans. The call by value method of passing arguments to a function copies the actual value of a argument into the formal parameter of the function. In this case, changes mode to the paramete inside the function have no effect on the argument. By default, C++ uses call by value of pas arguments.

(PTU, May 2019

Chapter 2

Classes & Objects - II

Contents

Constructors, Destructors, friend functions, Parameterized constructors, Static data members, Functions, Arrays of objects, Pointers to objects, this pointer, and reference parameter, Dynamic allocation of objects, Copyconstructors, Operator overloading using friend functions, overloading.

POINTS TO REMEMBER

- Constructors and destructors are member functions that perform the automatic initialization and clean up for the objects.
- Constructors can be overloaded whereas destructors cannot be overloaded.
- A constructor can have default argument.
- If a class has at least one parameterized constructors, it will be programmer's duty to provide the zero argument constructor in the class.
- Constructor and descrutors have same names as that of class.
- A copy constructor is a special constructor that is used to create an object from an existing object. If there is no copy constructor in the class, compiler provide default copy constructor.
- Const data members can only be initialized through initializer list.
- Pointers are memory addresses variable i.e. the variable, which have addresses of the variable having some value, stored in the memory.
- Pointers is a variable having address of another variable and not the value.
- Point increases the execution speed of the C++ programming.
- The address of the memory variable can be accessed by using address operator (&).
- The pointer variable or memory address always has integer data which is also unsigned.
- Comparing two pointers that do not refer to value of same array will generate logical error.
- Scale factor is a scale or measurement of the length of the cell addresses of the different data types.
- A pointer that does not point to any data object is known as null pointer.
- In C++, the null pointer can be represented by the constant 0.
- The pointer pointing to object are referred to as object pointer.
- A string array has memory space allocated only at the time of execution. No memory allocation is done at the time of declaration. This is call dynamic memory allocation.
- To allocate memory dynamically for structure variable, we use new memory allocation operator.
- A dynamic structure can be released using the memory deallocation operators **delete.**
- When accessing member of a class using an object pointer, the arrow (→) operator is used instead of the dot operator (.)

- Object pointers are used to access the public member of an object.
- The address of a public data member can be obtained through an object and also through an object pointer.
- Free store is a pool of memory used to storage for objects during the program execution.
- **'This'** is a pointer to the current object. It is passed implicitly to an overloaded operator function.
- In the overloaded function, the second argreement is passed using **this** pointer.
- Pointer to pointer is just another way of writing multi dimensional array of pointer to objects.
- Pointer to pointers are also known as nesting of pointer.
- Nesting of pointers contains the address of the second pointes i.e. it points to the variable that contains the value desired.
- Class is a group of similar object.
- A class in C++ binds data and associated function together.
- While declaring a class, four attributes are declared : data member function, program access level and class tag name.
- The public member of a class can be accessed outside the class directly by using object of this class type.
- The private and protected members can be accessed within the class by the member functions only.
- The functions defined inside the class definition are automatically inline.
- A class definition can occur within the definition of another class. In such cases, the inner class is called the nested class and the outer class is known as the enclosing class.
- A class support the OOP feature of encapsulation by binding data and associated function together.
- A class supports abstraction by providing only the essential information to the outside world.
- A class supports data hiding through private and protected members.
- The object can be passed to as well as returned from function.
- An object is an instance of class. A class is a logical abstraction, but an object has physical existence.
- Each class member can be of a specific access specifier. Access specifier allowed in C++ are : private, public and protected.
- The scope resolution operator (: :) links a class name with a member name in order to tell the compiler what class the member belong to.
- Like a structure a class can have data member as well as member functions.
- Default access for a structure is public whereas for a class it is private.
- Data members are non static by default. However member can be declared as static.
- A class can be defined inside another class. Such class is called nested class.
- A class can also be defined inside a function. Such class is called local class.
- An abstract class is class from which no object can be created.
- A class that can hold objects of another class (es) is known as container class.
- Operator overloading makes the uses defined data types behave like built in data types.
- Operator function is used to overload an operator. Which can be member function of friend function.
- Only existing operator can be over load. New operator cannot be created.
- The overload operator must have at atleast one operand of uses defined data type.
- The overload operator follow the syntax rules of the original operator. They cannot be overidden.

- In postfix overloading of increment and decrement operator, the additional argument should be of type int.
- Possible type conversions are from basic type of class type, class type to basic type and class type to class type.
- Class type to basic type conversion can be performed by defining conversion function in the source class.
- Class type to class type conversion can be performed by defining conversion function in the source class or through one argument constructor that takes an object of the source class, in the destination class.

QUESTIONS-ANSWERS

Q 1. What is destructor? **(PTU, Dec. 2011, 2008 ; May 2019, 2004)**

Ans. Like a special member function constructor is called automatically when an object is created first. There is another function is called automatically when an object is destroyed. Such a function is called destructor. A destructor has the same name as the constructor but preceded by a tilde.

Like constructors, destructors do not have a return type. They also take no arguments the most common use of destructor is to de-allocate memory that was allocated for the objects by the constructor.

Q 2. Discuss custom new and delete operators. **(PTU, Dec. 2007)**

Ans. New operator : The new operator allocates the memory and always return a pointer to an appropriate type. The new operator is defined as

```
type * new type [size in integer]
e.g. int * ptr ;
     ptr = new int [50] ;
```

It allocates a memory block of 100 byts, 2 bytes for one integer value, total of 50 integers.

The new operator also permits the initialization of memory locations during allocation. The syntax for doing so is

```
type * ptrvar = new type [intial value] ;
```

delete operator : The delete operator is opposite of new operator and it deallocates memory allocated by new operator back to the free poll of memory. The syntax is

```
delete ptrvar ;
```

Where ptrvar can be simple pointer variable or the name of the array of pointer to all typs except class type.

If ptrvar is an array of pointers to objects, then we have to use delete operator as

```
delete ptrvar [ ] ; or delete [ ] ptr var ;
```

The second form is used when we want to deallocate more than one array with single use of delete operator.

Q 3. What do you mean by default constructor? **(PTU, Dec. 2007)**

Ans. In case no constructor is defined by the programmer for a class, then compiler automatically inserts one constructor in the class definition prior to compiling the class. The constructor has following structure.

```
class name ( ) { }
```

This constructor takes no arguments and does nothing as it carries no statements. In this case garbage values will be assigned to the object.

Default constructor is used to invoke when an object is created with no arguments.

Q 4. What is memory leak? **(PTU, May 2007)**

Ans. Memory leak is a serious problem with pointers. It is a situation where the programmer fails to release the memory allocated at run time in a module. When memory is allocated, a pointer variable is used to hold the address of the allocated block, however, when the module completes its execution, the pointer variable goes out of scope and there will be no way to reach that memory block. Therefore, care must be taken to release the allocated memory block in a module where memory was allocated.

Q 5. Explain inline function, constructor and destructor. **(PTU, May 2019, 2009)**

OR

Write short note on Inline functions. **(PTU, Dec. 2011, 2005)**

Ans. Inline functions are those functions whose body is inserted in place of the function call during the compilation process. Therefore, with inline functions, the program will not incur any context-switching overhead.

The inline funtions are best used for small and frequently used functions. An inline function definition is similar to an ordinary function except the keyword inline precedes the function definition.

e.g.
```
inline int a (int b)
{
    return b*b ;
}
```

A constructor is a special member function whose task is to initialize an object of a class when it is created. It is special in the sense that it has a name that is same as that of its class. A constructor is automatically invoked when an object of its associated class is created. It is called constructor because it constructs object with its initial state by assigning values to it data members.

A constructor is a member function whose name is same as that of the class. A constructor is declared and defined in the same manner as other member functions. It is placed in the public section of the class.

e.g.
```
Class student
{
    private ;
        int roll no ;
        char nane [25] ;
        int age ;
    public :
        student ( )
        {
            roll no = 100 ;
            strcpy (name, "ABC") ;
            age = 15 ;
        }
        // other member functions
} ;
student std ;
```

Destructors : A destructor, as its name implies is a special member function that is used to release the resource held by the object before the object is destroyed. Like a constructors the destructor also the same name as that of class but is preceded by character tiled (~).

e.g.
```
class sample
{
    private ;
        // data members
    public :
        ~ sample ( )
        {
            // statments for the destructor
        }
        // other member functions
} ;
```

A destructor is a member function whose name is same as that of the class but is preceeded with character tilde '~'. Destructor takes no arguments. It is declared and defined in the same manner as other member functions. It is placed in the public section of the class. It can be defined in the class itself or outside the class.

Q 6. What are the various types of constructors used with the classes? Explain with examples. **(PTU, May 2005)**

Ans. Constructor : Constructor is special member function whose task to initialize an object of a class when it is created. Following are the types of constructors used with the classes :

1. Zero argument constructor : It takes no arguments. It is used to initialize every object with same initial values.

e.g.
```
Class A
{
    private :
        int no ;
        float a1 ;
    public :
        A ( ) // zero argument constructor
        {
            no = 1, a1 = 5.0 ;
        }
        // other member functions
} ;
```

2. Parameterized Constructor : When it is required to initialize the data members of different objects with different values when they are created. Then we use parametrized constructor.

e.g.
```
Class A
{
    private :
        int no ;
        float a1 ;
    public :
        A (int a, float b)          // parameterized constructor
        {
            no = a ;
            a1 = b ;
        }
```

```
    // other functions
};
```

3. Copy Constructor : A constructor can accept a reference to its own class as a parameter. A constructor that accepts a reference to its own class is called copy constructor. A copy constructor is used to initialize an object from another objects of same class. e.g.

```
Class A
{
    private :
        int no ;
        float a1 ;
    public :
    A (A & obj)                    // copy construcftor
    {
        no = obj. nò ;
        a1 = obj. a1 ;
    }
    // other member functions
};
```

4. Overloaded Constructor : C++ permits to use default constructor, zero argument or copy constructor in a same class. The constructor is actually used in initializing an object depends on the way the object is declared.

e.g.

```
Class A {
private ;
    int no ;
    float a1 ;
public :
    A ( ) ;              // zero argument constructor
{
    no = 3 ;
    a1 = 5.0 ;
}
A (int a, float b)       // parameterized constructor
{
    no = a ;
}   a1 = b ;
A (A & obj)              // copy constructor
{
    no = obj. no ;
    a1 = obj. a1 ;
}
    // other member functions
};
```

5. Constructor with default arguments : Constructor can have default arguments. But default arguments must be trailing ones.

e.g.

```
Class A {
```



```
private ;
    int no ;
    float a1 ;
public :
    A (int a, float b = 3.0)    // parameterized constructor with default argumnt
    {
        no = a ;
        a1 = b ;
    }
    //other functions
};
```

Q 7. What are the characteristics of a constructor? **(PTU, May 2019, 2010)**

Ans. Characteristics of Constructor :

1. A constructor function is a special function that is a member of the class and has the same name as that class.
2. An object constructor is called when the object is created.
3. Constructor does not return any value, not even void.
4. No return type should be mentioned with constructor.
5. A constructor for a class is needed so that the compiler automatically initializes an object as soon as it is created.
6. If there is no constructor in a class, the compiler creates a default constructor, but it does not initialize fundamental type variables.

Q 8. Explain the use of copy constructor with an example. **(PTU, May 2010)**

Ans. Copy constructor : The parameters of a constructor can be of any type except an object of the class to which it belongs. A constructor can accept a reference to its own class as a parameter. A constructor that accepts a reference to its own class is called copy constructor. A copy constructor is used to initialize an object from another objects of same class.

e.g.

```
Class number
{
    private :
        int real ;
        float imagine ;
    public :
        number (number & n)
        {
            real = n. real ;
            image = n.image ;
        }
        // other member functions
};
```

A copy constructor is used to declare and initialize an object from another object of its class.

Q 9. What do you mean by parametrized constructors? **(PTU, Dec. 2010)**

Ans. Parameterized Constructor : When it is required to initialize the data members of different objects with different values when they are created. Then we use parametrized constructor. e.g.

```
Class A
{
        Private ;
        int no ;
        float a1;// parameterized constructor
    Public
        A (int a, float b)
        no = a ;
        a1 – b ;
    }
      // other functions
};
```

Q 10. What is friend function? Explain the features of friend function. Also give an example for friend function. **(PTU, May 2019)**

Ans. Friend function : The function that are declared with the keyword **friend** are known as friend functions. A function can be declared as a friend in any number of classes, it has full access rights to the private members of the class.

Features of friend function :

1. It can be invoked like a normal function, with the help of the object.
2. It has the object as arguments.
3. It is not in the scope of the class to which is has been declared has friend.

Example of friend function :

```
#include <iostream.h>
#include <conio.h>
   class real ;
   class integer
{
   private
       int a ;
   public
       integer ( )                          // constructor
       {
           a = 0 ;
       }
       integer (int i) // constructor
       {
           a = i ;
       }
           void get ( )
       {
           cout <<end / <<"Enter the value :" <<end / ;
           cin >> a ;
       }
           void show ( )
       {
           cout <<end 1 << "a =" << a ;
```



```
       }
           Friend real add (integer i, real j) ;
       } ;
           class read
   {
       private
           float b ;
       public
           real ( )
       {
           b = 0.0 ;
       }
           void get ( )
       {
           cout <<end / << "Enter the value :" ;
           cin >> b ;
       }
           void show ( )
       {
           cout <<end / << "b = " << b ;
       }
           friend real add (integer i, real j) ;
       } ;
           void main ( )
       {
           integer X ;
           real y, z ;
           x.get ( ) ;
           y.get ( ) ;
           z = add (x, y) ;
           z.show ( ) ;
       }
           real add (integer i, real j)
       {
           real K ;
           K.b = i.a + j.b ;
           return K ;
       }
```

Q 11. What are the rules for static data members?

Ans. Rules for static data members are as follow :

1. They can be public as well as private.
2. They are initialized in a special manner. If you forget to initialize them, you will get a compilation error.
3. Static members share the same memory for all instance of the class, that is, a static member belongs to a class as a whole and not to its specific instances.

Q 12. What is a friend function? **(PTU, May 2011)**

Ans. The function that are declared with the keyword **friend** are known as friend functions. A function can be declared as a friend in any number of classes, it has full access rights to the private members of the class.

Q 13. What are the special characteristics of friend function?

Ans. 1. It can be invoked like a normal function, with the help of the object.
2. It has the object as arguments.
3. It is not in the scope of the class to which is has been declared has friend.

Q 14. What do you mean by friend class?

Ans. Declaration of friend is not restricted to method only. You can declare one class as friend of other. For example ;

```
class ABC  {  // some data members
              // some member functions
              friend XYZ ;
           }.
```

In this declaration class XYZ is declared as a friend of class ABC. It implies that XYZ can use all of data members and member functions in its definition.

Q 15. What do you mean by static class members? Explain the characteristics of static class member with suitable example.

Ans. Static class member : Static class member variables are used commonly by the entire class. It store value. No different copies of a static variable are made for each object. It is shared by all the objects. It is just like the C static variable.

Characteristics of static class member :

1. On the creation of the first object of the class a static variable is always initialized by zero.
2. All the objects share the single copy of a static variable.
3. The scope is only within the class but its lifetime is through out the program.

For example :

```
#include <iostream.h>
using namespace std ;
class num
{
static int c ;
int n ;
public :
void getd (int x)
{
    n = X ;
    C++ ;
}
void getc (void)
{
cout <<"cout : "<<c<< "\n" ;
}} ;
    int num : : c ;
    int main ( )
{
```



```
    num 01, 02, 03
    01. getc ( ) ;
    02.getc ( ) ;
    03.getc ( ) ;
    01.getd (1) ;
    02.getd (2) ;
    03.getd (3) ;
cout <<"After reading data" <<"\n" ;
    01.getc ( ) ;
    02.getc ( ) ;
    03. getc ( ) ;
    return 0 ;
}
```

Q 16. What is pointer?

Ans. Pointers are memory addresses variable i.e. the variables, which have addresses of the variable having same value, stored in the memory.

A pointer contains a memory address, further we can say, pointers are directly linked to the memory address. Actually pointer is available having address of another variable and not the value. Now suppose q is an integer variable and this variable has value 440. This value will be stored in memory. It is represented as :

Variable name	Variable-value	Memory address	
q	440	5000	← $ memory-variable (pointer)

Q 17. Explain any fives uses of pointers.

Ans. (i) A pointer enables the variable, which is used outside the function or used in the another sub-program.

(ii) Pointer increases the execution speed of the C++ programming.

(iii) Pointer reduces the length and complexity of the program.

(iv) Pointers are more efficient in handling the data table i.e. two-dimensional arrays.

(v) Use of the pointers to the character array or to the string saves the storage space in the memory.

(vi) Pointer gives accurate result.

Q 18. What do you mean by dynamic meaning allocation?

Ans. A string array has memory space allocated only at the time of execution. No memory allocation is done at the time of declaration. This is called dynamic allocation of memory. To handle dynamic allocation of memory, two operators are used. These are new and delete operator. The new operator can be used to create dynamic structure i.e. the structure for which the memory is dynamically allocated.

The general syntax used for this purpose is as :

```
struct_pointer = new struct_type ;
```

Q 19. Write a program to implement new and delete operator.

Ans. Program to implement new and delete operator.

```
#include <iostream.h>
#include <conio.h>
class abc
    {
        private ;
```

```
        int roll no ;
        float fees ;
    public :
        void read (int r, float f)
{
        roll no = r ;
    fees = f ;
}
        void display (void)
{
    cout <<"Roll No is" <<roll no ;
    cout <<"\n fees PAID is" <<fees ;
}
};
    void main ( )
    {
        clrscr ( ) ;
        class abc *p ;
        p = new abc ;
        p = read (20, 1200.32) ;
        p = display ( );
        getch ( ) ;
    }
```

Q 20. What do you mean by free store?

Ans. Free store is a pool of memory available for you to allocate storage for object during the execution of your program. The new and delete operators are used to allocate and deallocate free store, respectively. You can define your own versions of new and delete for a class by overloading them. You can declare the new and delete operators with additional parameters. When new and delete operate on class. Objects, the class member operator functions new and delete are called if they have been declared.

If you create a class object with the new operator one of the operator functions operator new ( ) or operator new [ ] ( ) is called to create the object. An operator new ( ) or operator new [ ] ( ) for a class is always a static class member, even if it is not declared with the keyword static.

Q 21. Explain array of pointers to an object through an example. (PTU, May 2010)

Ans. Two dimensional array can be taken as array of pointer. Here every element can hold address of any variable and every element of this array is a pointer variable. It contains the collection of addresses.

e.g. int * a [2] [5]

the above array can be represented in terms of pointer as :

int * a[5] ;

The *a [5] is an array of three pointers. The subscript 5 represents the row of the array rather than the column as in the case of pointer to a group of arrays. Precedence of array bracket [ ] is higher than the indirection operator *. Hence int *a[5] can be represented as (int*) (a[5]).

Q 22. What are the uses of 'This' pointer?

Ans. 1. "This" pointer is used when an object that generates the cell from the member function is returned.

2. When the name of arguments in member function are same as of data members. This pointer is used to remove so generated ambiguity.

Q 23. W.A.P to pointer to pointer.

Ans. Program to implement pointer to pointer.

```
#include <iostream.h>
#include <conio.h>
void main (void)
{
    int a1, * P1, ** P2 ;
    clrscr ( ) ;
    a = 20 ;
    P1 = &a ;
    P2 = &P1 ;
    cout <<"The value of A is stored at address" <<P1 ;
    cout <<"\n Address of P1 is stored at address" <<P2 ;
    cout <<"\n The value of A using pointer to pointer is" <<**P2 ;
    getch ( ) ;
}
```

Q 24. Write a program to show relationship of pointer to object.

Ans. Program to show relationship of pointer to object

```
#include <iostream.h>
#include <conio.h>
class abc
    {
        public :
            int a ;
        p = &x ;
        p = &y ;
```

It means void can handle any data-type easily.

Q 25. What is 'This' pointer? Explain with example.

Ans. When you define a class, then member functions are created and store space in the memory only once i.e. only one copy of the member function is created and it is stored by all of the class objects. Only space for data member is allocated separately for each object. So to share member function, a problem occurs i.e. if only one instance of a member function exists, how does it come to know which object's data member is to be mainpulated. To solve this problem we can use this pointer. This pointer is a special pointer which points to the object that invokes member function. When a member function is called, it automatically pass an implicit (in-built) argument that is a pointer to the object that invoke the function i.e. when a member function is called, the compiler automatically assigns the address of the object to this pointer and then the member function is called. For example, below is the program, which illustrate the concept of this pointer.

Program to implement 'This' operator.

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
class abc
```

```
        char nm [20] ;
        float fees ;
        public :
        abc (char *s, float f)
    {
        strcpy (nm, " ") ;
        strcpy (nm, s) ;
        fees = f;
    }
    abc (int i)
    {
        a = i ;
    }
};
void main ( )
    {
        clrscr ( ) ;
            class abc m1 (20) ;
            class abc * p ;
            p = &m1 ;                    // P is an object pointer to object m1
            int *q, tr ;
            q = &m1. a ;                 // q pointer to a
            r = &p – a ;                 // r pointer to a
            cout <<"a =" << m1.a ;
            cout << "*q" << *q ;
            cout <<"tr" << * r ;
            getch ( );
    }
```

Q 26. Write a program to show the behaviour of delete operator.

Ans. Program to show the behaviour of delete operator :

```
#include <iostream.h>
#include <conio.h>
#include <new.h>
struct A {
virtual ~ A \\ { cout <<"~A ( ) " <, end l ; } ;
void operator delete (void *P) {
cout << "A : : operator delete" << end l ;
delete (p) ;
struct B : A {
void operator delete (void *P) {
cout <<"B : : operator delete" << end l ;
delete (P) ;
}
} ;
int main ( ) {
clrscr ( ) ;
```



```
A* ap = new B ;
delete ap ;
getch ( ) ;
}
```

Q 27. Write short note on Dynamic memory management. **(PTU, Dec. 2011)**

OR

Discuss the concept of memory management in C++. Explain the concept of dynamic memory management with the help of suitable examples. **(PTU, Dec. 2009)**

Ans. Many programs have little need for memory management, they use a fixed amount of memory. C++ code is naturally organized by class, a common response to this failure is to overload member operator new for individual classes.

Detailed knowledge of the memory usage patterns of individual classes can be helpful, it is best applied by tuning memory usage for a whole program or major subsystem. In C++, the only way to organize memory management on a larger scale than the class is by overloading the global operator new. To select a memory management it require to adding a placement argument, in this case a reference to a class which implements :

```
extern void *operator new (size t, class heap &) ;
```

When we overload the operator new in this way, the regular operator new is implementing a policy of its own, and we like to tune to it as well.

Dynamic Memory Management : In all programs, we have only as much memory available as we declared for our variables, having the size of all of them to be determined in the source code, before the execution of the program. But if we need a variable amount of memory that can only be determined during run time? In case we need some user input to determine the necessary amount of memory space. For this C++ integrates new and delete operators.

In order to request dynamic memory we use the operator new. New is followed by a data type specifier and if a sequence of more than one element is required the number of these within brackets [ ]. It returns a pointer to the begining of the new block of memory allocated. Its form is :

```
pointer = new type
pointer = new type [number of elements]
```

The first statement is used to allocate memory to contain one single element of type. The second one is used to assign a block of elements of type, where the number of elements is an integer value representing the amount of these. eg :

```
int * abc ;
abc = new int [5] ;
```

In this case, the system dynamically assign space for five elements of type int and returns a pointer to the first element of the sequence, which is assigned to abc. Therefore, abc points to a valid block of memory with space for five elements of type int.

Q 28. Explain the use of implicit this pointer with example. **(PTU, Dec. 2013)**

Ans. The implicit this pointer : Each object declared has its own copy of data members. But note that there is only one copy of each class member function available for all the class objects. Each member function has a pointer which holds the address of the object itself. And it is called the this pointer. This pointer is a special pointer which points to the object that invokes member function. When a member function is called, it automatically pass an implicit (in-built) argument that is a pointer to the object that invoke the function i.e. when a member function is called, the compiler automatically assigns the address of the object to this pointer and then the member function is called. Then this pointer is a built-in pointer. This pointer points to the object being processed. That is, using the this

pointer any class member function can find out the address of the class object for which the member function is called.

The this pointer is also used to access the data members of the object. The this pointer can be treated just like an ordinary pointer to an object. Since it is a pointer to an arrow operator (–>) is used as member access specifier. This pointer is more useful in returning values from member functions and overloaded operators.

The following program illustrates the use of this pointer in accessing the data members of an object.

```
#include<iostream.h>
class number
{
private : int x;
public : void setvalue (int d)
{
this –> x = d; //this pointer
}
Void display (void)
{
cout<<"x = "<<this –> x <<endl;
cout<<"Address of the object = "<<this;
}
} ; //End of class definition
void main()
{
number n1,n2; //n1 and n2 are two objects
n1.setvalue(30);
n2.setvalue(80);
n1.display();
n2.display();
} //Ends of main()
```

Q 29. Discuss pointers related problem with examples. **(PTU, Dec. 2017)**

Ans. Two of the major problems that plague C++ programmers are memory leaks and memory corruption. A memory leak occurs when memory is allocated but never freed. This causes wasting memory and eventually leads to a potentially fatal out of memory. A memory corruption occurs when the program writes data to the wrong memory location, overwriting the data that was there, and failing to update the intended location of memory. Both of these problems falls squarely on the pointer.

Q 30. What are the rules of operator overloading?

Ans. Rules for operator overloading :

1. Only the operators which are part of the C++ language can be overloaded. No new operator can be created using operator overloading.

2. You can change the meaning of the operator i.e. a+ operator can be overloaded to perform multiplication operation or >> can be overloaded to perform addition operation.

3. Any overloaded operator function must have at least one operand which is user defined type. All of the operands cannot be of basic type if this is the case then function must be friend function of some class.

4. In case of overloading binary operators left hand side operator must be an object of class when overloaded operator function is a member function of the class.

5. Binary operators overloaded through member function of the class take one argument and overloaded through friend function take two arguments.

6. Many operator overloaded through member function of the class does not take any argument and overloaded through friend function must take are argument.

7. : :, ., *, ? : , size of these operator cannot be overloaded.

8. =, –>, [ ], ( ) these operator cannot be overloaded using friend function.

Q 31. What is friend function? **(PTU, Dec. 2011)**

Ans. A friend function is used for accessing the non-public members of a class. A class can allow non-member functions and other classes to access its own private data, by making their friends. Thus, a friend function is an ordinary function or a member of another class.

Q 32. Which operators cannot be overloaded? **(PTU, Dec. 2010)**

Ans. There are five operators which can't be overloaded. They are ;

1. **.*** – Class member access operator.
2. **: :** – It scope resolution operator.
3. **.** – dot operator
4. **? : :** – Conditional operator
5. **Size of ( )** – Operator.

Q 33. Explain the mechanism of unary operator overloading. **(PTU, May 2010)**

Ans. Unary Operator Overloading : We can overload a unary operator with either a non-static member function that has no parameters or a non-member function that has one parameter. Suppose a unary operator x is called with a statement xt, where t is an object of type T.

A non-static member function that overloads this operator would have the following form :

return_type operator x ( )

A non-member function that overloads the same operator would have the following form :

return_type operator x (T).

When you write overloaded operator functions, it can be useful to implement separate versions for the prefix and postfix versions of these operators. To distinguish between the two, the following rule is observed. The prefix form of the operator is declared exactly the same way as any other unary operator ; the postfix form accepts an additional argument of type int. When specifying an overloaded operator for the postfix form of the increment or decrement operator the additional argument must be of type int ; specifying any other type generates an error.

A unary expression contains one operand and a unary operator. All unary operators have the same precedence and have right-to-left associativity. A unary expression is therefore a postfix expression.

Q 34. What is meant by operator overloading? Explain the operation of overloading of an assignment operator. **(PTU, May 2009)**

Ans. Operator Overloading : It is important feature of OOP using operators in different ways, depending on what they are operating on is called polymorphism. When an existing operator, such as +or =, is given the capability to operate on a new data type, it is said to be overloaded, overloaded is kind of polymorphism.

Assignment operator overloading : Assignment operators can be overloaded in a same as arithmetic or unary operators. Let us look at example of overloading of arithmetic assignment operator.

```
#include <iostream.h>
```

```
class dist
{
    private ;
        int f ;
        float in ;
    public ;
        dist ( )
        {
            f = 0, in = 0.0 ; }
        void get dist ( )
        {
            cout <<"Enter feet and inches" ;
            cin >> f >> in ;
        }
    void show dist ( )
        {
            cout << f << "–" << in ;
        }
    void operator + = (dist d2)
    {
        f + = d2. f ;
        in + = d2. in ;
        if (in > = 12.0)
        {
            in – = 12.0 ;
            f ++ ;
        }
    }
} ;
void main ( )
{
    dist obj1, obj2 ;
    obj1. get dist ( ) ;
    cout <<"dist 1 = " ;
    obj1. show dist ( ) ;
    obj2. get dist ( ) ;
    cout <<"dist 2" ;
    obj2. get dist ( ) ;
    objt1 + = obj2 ;
    cout <<"After addition dist 1" ;
    dist 1. show dist ( ) ;
}
```

Q 35. Discuss operator overloading in C++. **(PTU, Dec. 2007)**

OR

What is operator overloading? Explain binary operator overloading. **(PTU, Dec. 2006)**

Ans. Operator overloading is one of the exiting features of C++. Operator overloading facilitates

the use of most of the operations permissible on user-defined data types. These operations include input/output, arithmetic operations, comparisons and assignment. The mechanism of giving new meanings to an operator is known as operator overloading. Semantics of an operator can be extended but its syntax can't be changed.

To give an additional meaning to an operator, we must specify what it means in relation to the class to which the operator is applied. The syntax for overloading is

```
returntype operator op (arglist)
{
    // funciton body
}
```

where op is the operator being overloaded and arglist is the list of arguments for the operator that may be empty. Operator function is either non-static member function or friend function. The process of overloading involves the following steps :

1. Create a class that defines a data type that is to be used for overloading an operator.
2. Declare the operator function op (arglist) in the public part of the class. It can be member function or friend function.
3. Define the operator function to implement the required operation.

Only existing operators can be overloaded. New operators cannot be created.

Binary Operator Overloading : Binary operators include arithmetic operators '+', '–' and '/' operators and relational operators and arithmetic assignment operators. Binary operator overloaded by means of a member function take single parameter and return a value but overloaded by means of friend function take two parameters of its class and return a value. **e.g.** we take an example of arithmetic operator '+', overloading by means of member function

```
#include <iostream.h>
class distance
{
    private :
        int feet ;
        float inches ;
    public :
        distance ( )
        {
            feet = 0 ;
            inches = 0.0 ; y
        distance (int ft, float in)
        {
            feet = ft ;
            inches = in ;
        }
    void getdist ( )
    {
        cout <<"Enter feet" ;
        cin <<feet ;
        cout <<"Enter inches" ;
        cin >> inches ;
    }
```

```
        void showdist ( )
        {
            cout <<feet <<"–" <<inches ;
        }
        distance operator + (distance) ;
};
distance distance : : operator + (distance d2)
{
    int f = feet + d2. feet ;
    float i = inches + d2. inches ;
    if (i > = 12.0)
    {
        i –3 = 12.0 ;
        f + + ;
    }
    return distance (f, i) ;
    }
void main ( )
{
    distance dist1, dist3, dist 4;
    dist1. getdist ( ) ;
    distance dist2 (11, 5.5) ;
    dist 3 = dist 1 + dist 2 ;
    dist 4 = dist 1 + dist 2 + dist 3 ;
    cout <<"dist 1" ;
    dist 1 . show dist ( ) ;
    cout <<"dist 2" ;
    dist 2. show dist ( ) ;
    cout <<"dist 3" ;
    dist 3. showdist ( ) ;
    cout <<"dist 4" ;
    dist 4. show dist ( ) ;
}
```

The above example shows the overloading of binary operator '+' using the member function operator.

Q 36. What is the function overloading? Explain the concept of constructor overloading. Can a template function be overloaded? **(PTU, May 2006)**

Ans. Function overloading : A function can be overloaded. An overloaded function appears to perform different activities depending on the kind of data sent to it. In C++, it is possible to use the same function name to perform identical operation. Such functions are called overloaded functions and the process of defining such functions is called function overloading. The overloaded function must differ in argument list either different number of arguments, or different in data types of arguments or they may differ both in number and type of arguments.

e.g.
```
void no ( ) ;
void no (int) ;
void no (int, char) ;
```


These three functions has same name but differ in number and type of arguments.

Constructor overloading : As functions constructors can be overloaded. The overloaded constructor also differ in number of arguments it required. And which of the constructor is executed depends on an object that how many arguments are used in definition.

e.g.
```
class no {
    private :
        int n, m ;
    public :
        no ( )
        {
            n = 1 ;
            m = 2 ;
        no (int p, int q)
        {
            n = p ;
            m = q ;
        }
            // other member functions
} ;
```

Above class no contains two constructors with different argument list.

Template function : A template function can be overloaded either by another template function or normal function. In this compiler uses overloading resolution to call the proper function. The compiler performs a matching process to determiner what function to call when a function call is encountered in the source code. Template overloading is also different in number and type of arguments.

e.g.
```
template <class T>
    void display (T arguments) {
        cout <<"Value of arg" << argument ;
    }
    void display (float argument) {
        cout <<"value of arg "<<argument ;
    }
```

In above example display template is overloaded with different argument list.

Q 37. Write a program to overload the +, –, x, % operator to find the addition, subtraction, multiplication and division of Complex numbers. **(PTU, May 2014)**

Ans.
```
#include<iostream.h>
#include<conio.h>
class complex
{
float x,y;
public:
complex() {}
complex (float real, float img)
{
x = real, y = img;
}
complex operator+(complex);
```

```
    complex operator–(complex);
    complex operator*(complex);
    void operator/(complex);
    void display()
    {
    cout<<x<<"+" <<y<<"i"<<endl;
    }
    };
    complex complex::operator+(complex c)
    {
    complex temp;
    temp.x = x+c.x;
    temp.y = y+c.y;
    return(temp);
    }
    complex complex::operator–(complex d)
    {
    complex temp;
    temp.x=x–d.x;
    temp.y=y–d.y;
    return(temp);
    }
    complex complex::operator*(complex e)
    {
    complex temp;
    temp.x = x*e.x+y*e.y*(–1);
    temp.y = x*e.y+y*e.x;
    return(temp);
    }
    void complex::operator/(complex f)
    {
    complex temp;
    temp.x = x*f.x+y*(–f.y)*(–1);
    temp.y = x*(–f.y)+y*f.x;
    float demo;
    demo = f.x*f.x–f.y*f.y*(–1);
    cout<<temp.x<<"+"<<temp.y<<"i/"<<demo;
    }
    void main()
    {
    clrscr();
    complex C1(5,3), C2(3,2), C3 = C1+C2, C4 =C1–C2, C5=C1*C2;
    C1.display();
    C2.display();
    cout<<"Addition"<<endl;
```



```
    C3.display();
    cout<<"Subtraction"<<endl;
    C4.display();
    cout<<"Multiplication"<<endl;
    C5.display();
    cout<<"Division"<<endl;
    C1/C2;
    getch ();
    }
```

Q 38. Differentiate between friend function and member function with the help of an example. **(PTU, Dec. 2011)**

Ans. Friend function : Friend function allows to operate an object of two different classes. Function will take objects of two classes as arguments and operate on their private data.

e.g.

```
#include <iostream.h>
Class B ;
Class A  ;
{
Private :
int no ;
public
A ( ) {n = 2 ;}
friend int fun (A, B) ;
} ;
Class B
{
Private
int no ;
Public ;
B ( ) {no = 5 ;}
friend int fun (A, B) ;
} ;
int fun (A, a, B, b)
{
return (a not b.no) ;
}
void main ( )
{
A obj 1 ;
B obj 2 ;
cout <<fun (obj1, obj2)
}
```

Member function : The variables declared inside the class are known as data members and the functions are known as member functions. Only the member functions can have access to the private data members and private functions. However, the public members (both function and data) can be accessed from outside the class.

Q 39. How a friend function is different than member function? **(PTU, Dec. 2008 ; May 2008)**

Ans. Friend function : Friend function allow to operate an objects of two different classes. Function will take objects of two classes as arguments and operate on their private data.

e.g.

```
#include <iostream.h>
class B ;
class A
{
    private :
        int no ;
    public
        A ( ) {no = 2 ; }
        friend int fun (A, B) ;
} ;
class B
{
private :
    int no ;
public :
    B ( ) {no = 5 ; }
    friend int fun (A, B) ;
} ;
int fun (A a, B b)
{
    return (a.no + b.no) ;
}
void main ( )
{
    A obj1 ;
    B obj2 ;
    cout << fun (obj1, obj2) ;
}
```

Member function : Member function are those functions which are included in classes. These functions are used to access private data members of the class. The syntax for accessing a class member is similar to that of accessing a member of a structure i.e. using dot operator as show below:

object name. public data member name ;

Only public members can be accessed outside the class specification. The private and protected class members can only be accessed by member functions that can be private or public.

Q 40. Write a program to illustrate overloading operators using friends.

Ans.

```
#include <iostream.h>
const size = 3;
class vector
{
int v [size];
public;
vector ( );
```



```
vector (int * x);
friend vector operator * (int a, vector b);
friend vector operator * (vector b, int a);
friend istream & operator >> (istream & vector &);
friend ostream & operator << (ostream & vector &);
};
vector : : vector ( )
{
for (int i = 0; i < size ; i + +)
v [i] = 0;
}
vector :: vector (int * x)
{
for (int i = 0; i < size ; i + +)
v [i] = x [1];
}
vector operator * (int a, vector b)
{
vector C;
for (int + i = 0; i < size ; i + +)
C. v [i] = a * b. v [i];
return C;
}
vector operator * (vector b, int a)
{
vector C;
for (int i = 0; i < size ; i + +)
C. v [i] = b. v[i] * a;
return C;
}
istream & operator >> (istream & din, vector & b)
{
for (int i = 0, i < size ; i + +)
din > > b. v[i];
return (din);
}
ostream & operator <<(ostream * dout, vector * b)
{
dout << "(" <<b.v [0];
for (int i = 1; i < size ; i + +)
dout <<"<<b.v [i];
dout << ")";
return (dout);
}
int x [size] = {2, 4, 6};
int main ( );
{
vector m;
```

```
vector n = x;
Cout <<"Enter elements of vector m" <<"/n";
Cin >>m;
Cout <<"/n";
Cout <<"m = " <<m <<"/n";
vector p, q;
p = 2 * m;
q = n * 2;
cout <<"\n";
cout <<"P = " << P << "\n";
cout <<"q = " <<q <<"\n";
return 0;
}
```

Q 41. What is static storage class ? What are its characteristics ?

(PTU, May 2019 ; Dec. 2017)

Ans. The variable of class has the visibility of a local variable but the lifetime of an external variable. That means once declared inside a function block, it does not get destoryed after the function is executed, but retains its value so that it can be used by future function calls.

Q 42. What is the difference between calling method of constructor and destructor?

(PTU, Dec. 2014)

Ans. A constructor is a function that is called, either explicitly or by compiler generated behind the scenes code, for the purpose of initializing the state of an object. It is given raw uninitialized memory of the object's size. It can, of course, explicitly allocate memory for noncontiguous field, but the compiler considers only the resulting pointer to be part of the actual object. We can also say that constructors are automatically executed when the object of that class is created. A constructor is written as a function that has the same name as the class. It returns no value, not even void.

Complex x(2.5, –1.0), y,z ; we do need to code an explicit destructor whenever the constructors allocate a resource such as memory. If a constructor allocates memory, the destructor for that class must free it. Furthermore, if any constructor for a class allocates a resource, then all constructors for that class should do so, unless some complicated scheme allow the destructor to figure out when to free the resource. The destructor has the same name as the class but is prefixed with a tidle character. It takes no parameter since programs never invoke it explicitly.

~Complex() {... code to free resources.....}. The destructor is invoked whenever the object is to be destroyed. We can also say that these are the type of member function which are automatically executed when an object of that class is destroyed is called a destructor.

Q 43. What is the need of friend function? **(PTU, Dec. 2014)**

Ans. The need of friend function : Suppose you are going out of your house for some days and your house is under repair. Now to complete the repair either you leave the house unlocked or give the keys of your house to some family friend. You will avoid to unlock your house and hand over the keys to your family friend. So similarly friend function can share the private and protected data member and member function.

Q 44. Explain the constructors with default arguments. How constructors can be called explicitly? **(PTU, Dec. 2014)**

Ans. It is possible to define constructors with default arguments. For example, the constructor complex() can be declared as follows :

complex(float real, float imag = 0);

The default value of the argument imag is zero. Then, the statement

complex c(5.0);

assigns the value 5.0 to the real variable and 0.0 to imag (by default). However, the statement

complex c(2.0,3.0);

assigns 2.0 to real and 3.0 to imag. The actual parameter, when specified, overrides the default value. As pointed out earlier, the missing arguments must be the trailing ones.

It is important to distinguish between the default constructor A::A( ) and the default argument constructor A::A(int=0). The default argument constructor can be called with either one argument or no arguments. When called with no arguments, it becomes a default constructor. When both these forms are used in a class, it causes ambiguity for a statement such as

A a;

The ambiguity is whether to 'Call' A::A( ) or A::A(int=0).

Constructor can be called explicitly.

For example :

```
#include<iostream.h>
#include<conio.h>
class b
{
public:
b( )
{
cout<<"constructor"<<endl;
}
};
void main( )
{
clrscr( );
b s;
b( );
}
```

Here b( ) explicitly calls the constructor.

Q 45. What is the difference between overloading and overriding of a function? Write a program in C++ to overload == operator and compare two objects using the operator.

(PTU, Dec. 2014)

Ans. The function overloading and function overriding are the concepts in C++ in which the same function name is used. But in function overloading different number of parameters can be passed whereas in function overriding the number of parameters that are passed to the function are same.

The function overloading may have different return type, but in function overriding the return type of base and derived member function is same.

Function overriding is a method that allows defining multiple member functions with the same name but different signatures. The signature means its name plus the number and types of the parameters it accepts. The compiler will pick the correct function based on the signature. Overriding is a method that allows the derived class to redefine the behaviour of member functions which the derived class inherits from a base class. The signatures of both base class member function and derived class member function are the same; however, the implementation and therefore, the behaviour will differ.

```
#include<iostream>
```

```
using namespace std;
class Rational
{
private:
        int num ; //numerator
        int den; // denominator
        public:
        void show( );
        Rational(int =1,int =1);
        void set numden(int,int);
        Rational add(Rational object);
        Bool operator == (Rational object);
        };
        Rational Rational::add(Rational object)
        {
int new_num = num*object.den+den*object.num;
int new_den = den*object.den;
return Rational(new_num, new_den);
}
void Rational::show()
{
        cout<<num << "/" << den << "\n";
}
Rational:Rational(int a, int b)
{
setnumden(a,b);
}
void Rational::setnumden(int x, int y)
{
int temp, a,b;
a = x;
b = y;
if(b>0)
{
temp = b;
b = a;
a = temp;
}
while(a!=0 && b!=0)
{
if(a%b ==0)
break;
temp = a%b;
a = b;
b = temp;
}
```



```
num = x/b;
den = y/b;
}
bool Rational::operator == (Rational object) {
return(num == object.num && den == object.den);
}
int main()
{
Rational obj1(1,4), obj2 (210, 840), result1;
result1 = obj1.add(obj2);
result1.show();
if(obj1 == obj2)
{
cout <<"The two objects are equal."<<endl;
}
else
{
cout <<"The two fractions are not equal."<<endl;
}
return 0;
}
```

Q 46. What is Overloading and its use ?

(PTU, May 2015)

Ans. C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively. An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope except that both declarations have different arguments and abviously different definition. When you call another loaded function or operator, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution.

Q 47. What do you understand by array of class objects ? Discuss with example.

(PTU, Dec. 2015)

Ans. The declaration of array of objects is very much similar to declaration of array of structures. As an array can be of any data type, we can have arrays of variables that are of the class type. Such variables are called as "Array of Objects". Array of objects are greatly used while dealing with applications pertaining to database.

Syntax :

```
class <class_name>
{
private:
– .....
public :
– .....
};
class <class_name> <Object_name[SIZE]>;
```

Ex. class employee

```
{
char name [20];
float salary;
public:
void get_details (void);
void display_details (void);
};
```

Q 48. What is Operator overloading ? Write a program in C++ to overload binary operator*. (PTU, Dec. 2015)

Ans. Operator overloading : Operator overloading refers to overloading of one operator for many different purpose. For example, the binary can be used to add two integer numbers, two float numbers, two structures variables, two union variables or two class objects. Use of operator overloading permits us to see no difference between built in data type and user defined data types. It is one of the powerful and fascinating features of the C++ which fine additional meaning to built in standard operator like +, –, *, /, >, <, < =, > = etc.

Program :

```
#include <iostream.h>
#include <conio.h>
class arithmetic
{
float n;
public :
void get ( )
}
cout <<"/n Enter member : \n";
cin >>n;
}
arithmetic operator + (arithmetic & a)
{
arithmetic t;
t. n = n+a–n;
return t;
}
arithmetic operator – (arithmetic & a)
{
arithmetic t;
t.n = n–a.n;
return t;
}
arithmetic operator(arithmetic & a)
{
arithmetic t;
t.n = n*a.n;
return t,
}
arithmetic operator/(arithmetic &a)
{
arithmetic t;
```



```
t – n = n/a–n;
return t,
}
void display ( )
{
cout <<n;
}
};
void main ( )
{
arithmetic a1, a2, a3;
a1 – get ( );
a2 – get  ( );
a3 = a1 + a2;
cout <<"\n Addition of two number : ";
a3,display ( );
a3 = a1–a2;
cout <<"\n Subtraction of two number : ";
a3.display ( );
a3 = a1 + a2;
cout <<"\n Multiplication of two number :";
a3. display ( )
a3 = a1/a2;
cout <<"\n Division of two number : ";
a3 display ( );
getch ( );
}
```

Q 49. What are static functions ? (PTU, May 2016)

Ans. Like static member variables, we can also have static member functions. A member function that is declared static has the following properties :

- A static function can have access to only other static members declared in the same class.
- A static member function can be called using the class-name a follows :
  Class_name : : function_name;

Q 50. Write a program to overload the plus operator to add two complex numbers. (PTU, May 2019, 2016)

Ans.

```
#include <iostream.h>
using namespace.std;
class complex
{
float x;
float y;
public:
complex ( ) { }
Complex (Float real, Float imag)
{ x = real; y = imag; }
complex operator + (complex);
void display (void);
};
```

```
Complex Complex :: operator + (Complex C)
{
Complex temp;
temp.x = x + c.x;
temp.y = y + c.y;
return (temp);
}
void complex :: display, (void)
{
cout << x << "+ J" <<Y<<"\n";
}
int main ( )
{
complex C1, C2, C3;
C1 = Complex (2.5, 3.5);
C2 = Complex (1.6, 2.7);
C3 = C1 + C2;
Cout <<"C1 = "; C1. display ( );
Cout << "C2 = "; C2. display ( );
Cout << "C3 = "; C3. display ( );
return 0;
}
```

Q 51. Consider a pointer declaration int i = 10, *p;p=&x;
Is p– –; a valid statement, justify. **(PTU, Dec. 2016)**

Ans. *No, P – – is not a valid statement as x variable is not declared.*

Q 52. What is the main difference between array of pointers and pointer to an array? Explain with the help of a suitable example. **(PTU, Dec. 2016)**

Ans. For array of pointers : *Refer to Q.No. 21*

For pointer to an array : Arrays and pointers are intimately linked in C + +. Here the address of the first element at memory location of block is known as base address which is assigned by array name which is turn behave like pointer variable.

As we know that pointers were incorporated into the C++ language from C where strings are expressed as character string arrays such as :

Char name [5] = "NEHA"

N	E	H	A	\0

Look at an array with three elements :

```
int      A [3];
 0       2       4
25      35      45
A [0]   A [1]   A [2]
```

By reference we can use arrays
& A [0] & A [1] & A [2]
With the help of pointers use of Array.
*A *(A + 1) *A (A + 2)
Address example : 1000 1002 1004

Q 53. Write a program to search a key string in an array of strings, if a key string is found then return its position and then replace that key string by any string using pointers. **(PTU, Dec. 2016)**

Ans.

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
char *rep_str (const char *s, const char *old, const char * new1)
{
char * ret;
int i, Count = 0;
int new len = str len (newl);
int old len = strlen(old);
for (i = 0; s [i]! = '\0'; i++)
{
if (str str [&sli], old) = = & s[i])
{
Count ++;
i + = old len –1;
}
}
ret = (char *) malloc (i + count * (newlen – oldlen));
If (ret = = NULL)
exit (EXIT – FAILURE);
i = 0;
while (*S)
{
If (strstr (S, old) = = S)
{
Strcpy (& ret [i], new l);
i + = newlen;
& + = oldlen'
}
else
ret [i + +] = * S ++;
}
ret [i] = '\0';
return ret;
}
int main (void)
{
char mystr [100], C[10], d [10];
Cont <<("Enter a string along with characters to be rep_str d :\n");
gets (mystr);
Cont <<("Enter the character to be rep_str d:\n");
cin>> C
Cont <<"Enter the new character:\n");
cin >>d
char * new str = NULL;
```

```
pus (mystr);
newstr = rep_str (mystr, c, d);
cout <<newstr;
free (newstr);
return 0;
}
```

Q 54. When are C++ copy constructors, assignment operators, and destructors, respectively, invoked ? **(PTU, Dec. 2016)**

Ans. (a) Copy constructors : There are three general cases where the copy constructor is called :

- When instantiating one object and initialize it with values from another object.
- When passing an object by value.
- When an object is returned from a function by value.

Assignment Oprator : An assignment operator is called when an already initialized object is assigned a new value from another existing object.

Destructors : If the object was created as an automatic variable, its destructor is automatically called when it goes out of scope. If the object was created with a new expression, then its destructor is called when the delete operator is applied to a pointer to the object.

Q 55. At what time are overloaded methods (as opposed to overridden) resolved ? **(PTU, Dec. 2016)**

Ans. In C ++ you can have two or more functions with the same name so long as they differ in their parameter list. This is called function overloading. The function is invoked whose parameter list matches the arguments in the call. Normally the compiler can deal with overloaded functions fairly easily by comparing the argument with the parameter lists of candidate functions. However, this is not always a straightforward matter. Consider the following code fragments :

```
Void f(double  dl, int i1)
{
.....
}
void f (double d1, double d2)
{
.....
}
int main ()
{
cout << f (1,0,2);
}
```

How does the compiler know which version of f ( ) to call ? The compiler works through the following checklist and if it still can't reach a decision, it issues an error :

1. Gather all the functions in the current scope that have the same name as the function called.
2. Exclude those that don't have the right number of parameters to match the argument in the call.
3. If no function matches, the compiler reports an error.
4. If there is more than one match, select the best match.
5. If there is no clear winner of the best matches, the compiler reports an error ambiguous function call.

Q 56. What do you understand by the following two declarations ?
(i) const int *pOne;
(ii) int const *pTwo;

Ans. (i) Const int *pOne; : It means that *p One is a const int i.e. pOne is a pointer to a const int. Here Its the "int" part that can't change. **(PTU, May 2017)**

(ii) int const *pTwo; : It means that just the variable value itself cannot be changed.

Q 57. Write a program to compare the two given strings using a pointer. **(PTU, May 2017)**

Ans.

```
#include <iostream>
#include <stdio.n>
using namespace std;
main( )
{
char str 1[50], str 2[50];
int str_cmp (char *, char *);
cout <<"Enter first string";
gets (str1);
cout <<"Enter second string";
get (str2);
If (str_cmp (str1, str2))
cout <<"nstrings are equal";
else
cont <<"nstrings are not equal";
return 0;
}
int str_cmp (char * S1, char * S2) {
while (*S1 = = * S2)
{
If (*S1 = = ' ' || * S2 = = ' ')
break;
S1 + +;
S2 +.+;
}
If (*S1 = = ' ' && * S2 = ' ')
return 1;
return 0;
}
```

Q 57. Differentiate between static and dynamic memory allocation. **(PTU, May 2019)**

Ans. The major difference between static and dynamic memory allocation :

Static memory allocation	Dynamic memory Allocation
1. In this case, variables get allocated permanently	1. In this case, variables get allocated only if your program unit gets active.
2. Allocation is done before program execution	2. Allocation is done during program execution
3. It uses the data structure called stack for implementing static allocation	3. It uses the data structure called heap for implementing dynamic allocation.
4. Less efficient	4. More efficient
5. There is no memory reusability.	5. There is memory reusability and memory can be freed when not required.

❑❑❑

Chapter 3

Inheritance

Contents

Base Class, Inheritance and protected members, Protected base class inheritance, Inheriting multiple base classes, Constructors, Destructors and Inheritance, Passing parameters to base class constructors, Granting access, Virtual base classes.

POINTS TO REMEMBER

- Inheritance is a mechanism that allow new classes to be derived from existing class (es) that facilitates reusability where existing class (es) is/are known as base class (es) and the newly created class is known as derived class.
- Derived class inherits all the properties of the parent class and add its own.
- Inheritance represents is a kind of relationship.
- A class can be derived using public, private and protected derivation.
- In public derivation, public and protected member of the base class remain public and protected member of the derived class, respectively.
- In protected derivation, public members of the base class become protected members of the derived class.
- The public and protected members of the base class become private members of the derived class.
- Various forms of inheritance are single, multiple, multilevel, hierarchical, hybrid and multi path.
- In single inheritance only one class is derived from single base class.
- In multiple inheritance, a class is derived from two or more than two base class.
- In multi level inheritance a class is derived from single base class and further can be used as base class for deriving another class.
- In hierarchical inheritance more than one class is defined from same base class.
- In hybrid, a class is derived involving two or more from of inheritance.
- Multiple inheritance may also cause ambiguity when a class's derived from more than one base class that is turn are derived from the same base class (es). This ambiguity is resolved using virtual base class.
- A technique for reusing functionality is object composition, where an object is contained in another class as a data a member.
- Composition represents has a relationship.
- Order of execution of destructor is reverse of constructor execution.
- Object slicing is a concept where additional attributes of derived class object is sliced to form a base class object.

- The constructors play an important role in initializing an object's data members.
- A class can contain objects of their classes. This is known as containership or nesting.

QUESTIONS-ANSWERS

Q 1. What do you mean by inheritance? **(PTU, Dec. 2010)**

Ans. Inheritance is one of the powerful technique of OOP which allows new classes to be built from existing classes. It is the mechanism of deriving new class from old class, new class is called sub class or derived class. Old class is called super class or base class. The derived class inherits all the properties of base class and its own, where the base class remain unchanged.

The general form of inheritance is :

```
{
Class derived_name :
} ;
```

Q 2. What are applications of inheritance? **(PTU, May 2009)**

Ans. Applications of Inheritance :

1. It permits code reusability. Once a base class is written and debugged, it need not be touched again but can be used to work different situations.
2. Reusing existing code saves time and money and increases a program's reliability.
3. Inheritance also help in original conceptualization of a programming problem, and in overall design of the program.
4. Reusability increases the ease of distributing class libraries. A programmer can use a class created by another person or company and withut modifying it, derive other classes from it that are suited to particular situations.

Q 3. What do you mean by derived class? **(PTU, Dec. 2011, 2008)**

Ans. Reusability and extensibility is among the most important features of object oriented programming. And it is possible through only inheritance. Inheritance allows new class, called derived class, to be built from existing class called base class instead of building from the scratch. The derived class is called child or subclass.

A derived class may itself be a base class from which additional classes can be derived. There is no specific limit on the number of classes that may be derived from one another, which forms a class hierarchy. The syntax for defining a derived class is

```
class derive class name : [Accessspecifier] Base class name
    {
        .....
        .....
        // members of the derived class
        .....
    } ;
```

The accessspecifier is optional and can be public, private or protected. The default acces specifier is private.

Q 4. What is function overriding? **(PTU, Dec. 2009 ; May 200**

Ans. The derived class can have a member functions with same name, same return type ar same list of arguments are defined in the base class. In such a case, the member function of t derived class overrides the member function of the base class. The overriding function can ma

refinements or enhancements suited to the derived class. With overriding functions, the calls in the program work the same way for objects of both base and derived clases.

The overriding methods in the derived class invokes the corresponding overridden methods of the base class to handle the parts of the object that pertains to the class and adds additional instructions to handle the parts of the object that are added in the derived class.

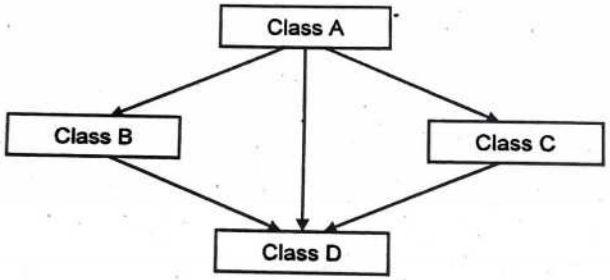
Q 5. What do you mean by visibility mode?

Ans. Inheritance is a process of creating a new class from the existing one. A visibility mode specifies the way in which the properties of the base class are derived. A derived class can have one of the following access specifies :

1. Public inheritance
2. Private inheritance
3. Protected inheritance.

Q 6. What do you mean by virtual base class? **(PTU, Dec. 2010)**

Ans. A different structural pattern with multiple inheritance occurs when a class derives from a more than one base and there is a common base shared among derived classes. The shared base class is called a virtual base class. For example we have one class A. This class A is inherited by two other classes B and C. Both these classes are inherited in new class D. This is shown in fig. given below :



Q 7. Does multiple inheritance lead to ambiguity? How can the ambiguity be removed? **(PTU, May 2005)**

Ans. Yes, multiple inheritance can lead to ambiguity. When the base classes have members with the same name, while the classes derived from these classes have no member with that name, then accessing that common member causes ambiguity when an object of the derived class access the member of the base class, the compiler will not be able to figure out which of these members should be used.

This ambiguity can be resolved using scope-resolution operator to specify the class in which the function lies. Thus obj. class name : : function name ( ) ;

refers to the version of function ( ) that is in the specified class.

Q 8. What do you mean by object slicing?

Ans. Object slicing is a concept where additional attributes of a derived class object is sliced to form a base class object. Object slicing doesn't occur when pointers or references to objects are passed as function arguments sinch both the pointers are of the same size. Object slicing will be noticed when pass by value is done for a derived class object for a function accepting base class object.

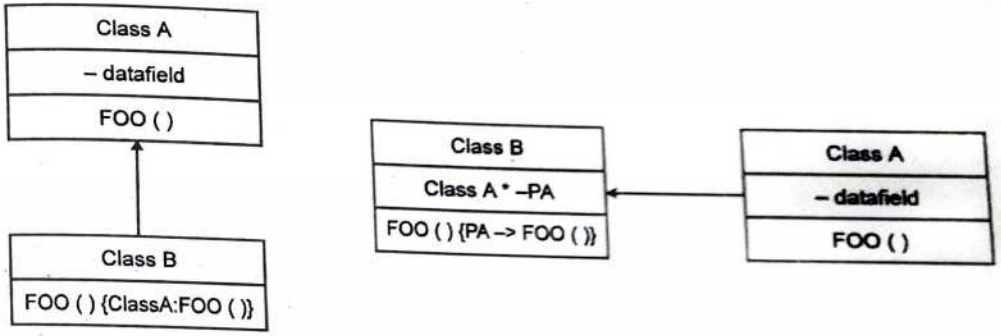
Q 9. Explain the concept of composition and delegation.

Ans. We say that class B is composed with class A is class B has a class A or class A* member ; for short we can say class B has a class A and as before, class B inherits from class A if class B is derived from class A as a child class ; for short we say class B is a class A.

As it turns out, you can always replace an inheritance relationship by a composition relationship as indicated in figure below if class B has a class A member object *_PA, then

(a) a class B object gets a set of class A data fields wrapped up inside *_PA

(b) class B can implement the same methods as class A simply by passing these method calls to *_PA. When you pass method calls to a composed object, this is called delegation.



Inheritance and composition

Q 10. What is inheritance? What are its different classifications? How the classes ar initialized in inheritance? **(PTU, Dec. 2008 ; May 2011, 200**

OR

What are the various types of inheritance in C++? Give an example of each. **(PTU, May 2019, 2012, 2004 ; Dec. 201**

Ans. Inheritance allows new class, called derived class to be built from the existing cla called base class, instead of building from the begining. New classes can be build by first inherting general features of some given class and then adding specific features.

The derived class inherits all of the capabilities of the base class and can add other feature its own. There is no specific limit on the number of classes that may be derived from one anot which forms a class hierarchy.

C++ supports three access specifiers : private, protected and public

Syntax for inheritance is

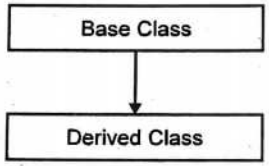
```
Class Base
{
private :
.....
.....
protected :
.....
.....
public :
```

```
.....
.....
};
class derived : [Access specifier] Baseclass
{
.....
.....
};
```

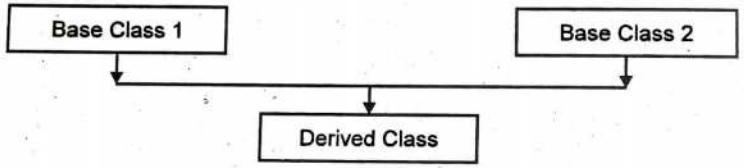
The access specifier is optional and can be public, private or protected. The default specifier is private.

Forms of Inheritance : Inheritance is classified into the following forms :

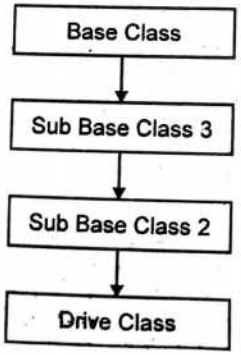
1. Single Inheritance : When only one class is derived from single base class, such derivation of class is known as single inheritance. Further, the derived class is not used as a base class for another class derivation.



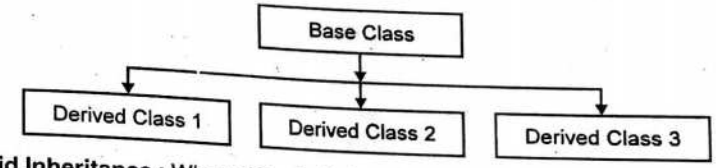
2. Multilevel Inheritance : When a class is derived from single base class and further can be used as base class for deriving another class. This derivation can continue upto any level.



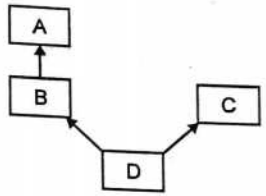
3. Multiple Inheritance : When a class is derived from two or more than two base class. The derived class inherits the properties of all the base classes. The derived class is not used further in deriving another classes.



4. Hierarchical Inheritance : Whem more than one class is derived from same base class. Further, derived classes are not used as base classes.



5. Hybrid Inheritance : When class is derived involving two or more forms of inheritance.



Q 11. How we can defining a derived class?

Ans. A derived class can be defined by specifying its relationship with the base class in addition to its own details. The general form of defining derived class is :

```
Class derived_Class-name : visibility mode base-class-name
{
.....//
.....// member of derived class
.....//
};
```

The colon indicates that the derived class-name is derived from the base-class-name. The visibility mode is optional and, if present may be either private or public. The default visibility-mode is private visibility mode specifies whether the features of the base class are privately derived or publicly derived.

Example

```
Class ABC : Private XYZ        // Private derivation
{
member of AB
};
Class ABC : Public XYZ         // Public derivation
{
member of ABC
};
Class ABC : XYZ                // Private derivation by default
{
member of ABC
};
```

When a base class is privately inherited by a derived class, public members of the base clas become 'private members' of the derived class and therefore the public member of the base class ca only be accessed by the member function of the derived class. They are in accessible to the objec

of the derived class. Remember, a public member of a class can be accessed by its own objects using the dot operator. The result is that no member of the base class is accessible to the objects of the derived class.

On the other hand, when the base class is publicly inherited, 'public members' of the base class become 'public members' of the derived class and therefore they are accessible to the objects of the derived class. In both the cases, the private members are not inherited and therefore, the private members of a base class will never become the members of its derived class.

Q 12. Write a program in C++ that illustrates the concepts of function overloading and function overriding. **(PTU, May 2010)**

Ans. Program to illustrate function overloading and function overriding

```
#include <iostream.h>
#include <conio.h>
#include <process.h>
class student
{
    private
        char name [20] ;
        char address [30] ;
    public
    void readdata ( )
{
    cout <<"Enter name & address" ;
    cin >> name >> address ;
}
    void read data (char a, char add)          // function overloading
{
    name = na ;
    address = add ;
}
    void display ( )
{
    cout <<"Name" << name ;
    cout <<"Address <<add ;
}
};
    class body : public student
{
    private ;
    in ht, wt ;
    public
    void getchar ( )
{
    cout <<"Enter height & weight of student" ;
    cin >> ht >> wt ;
}
    void display ( )                           // function overriding
{
```



```
    student : display ( ) ;
    cout <<"Height and weight is" <<h + <<wt ;
}
} ;
    void main ( )
    body S + S2 ;
    s1.readdata ( ) ;
    s1.getdata ( ) ;                           // calls overloaded member function read data ( )
    s1.display ( ) ;
    s2.readdata ("AB", "234, 16, PB") ;
    s2.get data ( );
    s2. display ( ) ;
    getch ( ) ;
}
```

Q 13. What are the various types of functions used in the classes? What do you mean by classes within the classes? **(PTU, May 2006)**

Ans. Following are the functions used in the classes :

1. Simple functions : Class includes simple functions which is group of number of program statements. These functions are known as member functions of the class they are used to access private member data.

2. Constructor : Constructor is special member function who is used to initialize an object of a class when it is created. Its name is same as that of its class. A constructor is automatically invoked when an object of its associated class is created.

3. Virtual function : A member function whose function declaration is preceded by virtual keyword is known as virtual function. These functions are defined in a base class in public section and they provide a mechanism by which the derived classes ca override it.

In many cases, virtual functions are declared without any body i.e. they don't have any definition. Such virtual functions are called pure virtual function.

4. Friend function : If want a function to operate an objects of two different classes then we create friend function. Function will take objects of two classes are arguments and operate on their private data. In such function we place friend keyword before definition of the function.

Classes within the classes : A class can be defined within another class. Such a class is called nested class. A nested class is a member of its enclosing class. The definition of a nested class can occur within a public, protected or private section of its enclosing class. The name of a nested class is visible only in its enclosing class scope.

Nested classes can use member of its enclosing class. Nested classes cannot have static data members.

e.g.

```
class student
{
    private :
        int roll no ;
        char name [20] ;
    public :
        class date
        {
            private :
```

```
        int day, month, year ;
    public :
        // member functions of data class
    } date of birth ;
// member functions of student class
};
```

Q 14. Distinguish between single and multiple inheritance. **(PTU, May 2010)**

Ans.

Single Inheritance	Multiple Inheritance

When only one class is derived from single base class, such derivation of class is known as single inheritance. Further, the derived class is not used as a base class for another class derivation
e.g.
# include <iostream.h> 	
# include <string.h> 	
Class A
{ // base class
Protected
char name [30] ;
int age ;
} ;
Class B : Public A // derived class
{
private ;
int weight ;
public
void get data ( ) {
cout <<"enter name" ;
cin >> name ;
cout <<"Enter weight" ;
cin >> weight ;
}
void showdata ( ) {
cout <<"Name" << name
cout <<"Age" <, age ;
cout <<"weight" << weight
} ;
void main ( ) {
B obj b ;
objb. get data ( ) ;
objb. show data ( ) ;
}
	When a class is derived from two or more than two base class. The derived class inherits the properties of all the base classes. The derived class is not used further in deriving another classes.
	e.g.
	#include <iostream.h>
	class A
	{
	Protected
	int age ;
	} ;
	Class B
	{
	Protected
	Char name [30] ;
	} ;
	Class C : Public A, Public B
	{
	Private
	int z
	} ;
	void main ( )
	{
	C objc ;
	}

Q 15. What are base and derived classes? How do they help in inheritance? **(PTU, May 2007)**

OR

What is inheritance? How it is implemented? Distinguish between public and private inheritance. **(PTU, Dec. 2006)**

Ans. Inheritance allows new class, called derived class, to be built from the existing class called base class instead of building from scratch. New classes can be build by first inheriting the general features of some given class and then adding specific features.

The derived class inherits all of the capable of base class and can add refinements and extension of its own. A base class is often called parent, superclass or ancestor. The derived class is called descendent, child or subclass. A derived class may itself be a base class from which additional classes can be derived. There is no specific limit on the number of classes that may be derived from one another.

C++ supports three access specifiers :
private, protected and public.
The syntax for defining a derived class is

```
class derived class name : [Access specifier] Base class name
{
    .....
    ..... // members of the derived class
};
```

The access specifier can be public, private or protected. The default access specifier is private.
The role of access specifier is :

1. When public access specifier is used, the public members of the base class remain publi members of the derived class. And protected members of the base class remain protecte members of the derived class.
2. When private access specifier is used, the public and protected members of the base clas become private members of the derived class.
3. When protected access specifier is used, the public members of the base class becom protected members of the derived class and protected members of the base class remai protected members of the derived class.

Many types of inheritance is allowed in C++. These are :

1. Single inheritance : Derivation of single class from only one base class.
2. Multilevel : Derivation of a class from another derived class. The class A is top base cla and B is derived of A further C is derived from class B.
3. Multiple : Derivation of a class from two or more base classes.
4. Hierarchial : Derivation of two or more classes from a single base class.
5. Hybrid : Derivation of a class involving two or more forms of inheritance.
e.g. This example shows the implementation of inheritance

```
#include <iostream.h>
#include <string.h>
class A   // base class
{
    protected :
        char name [20] ;
        int age ;
    public :
        // some member functions
```

```
};
class B : public A                    // derivd class
{
    private :
        int weight ;
    public :
        void getdata ( )
        {
            cout <<"Enter name" ;
            cin >> name ;
            cout << "Enter age" ;
            cin >> age ;
            cout <<"Enter weight" ;
            cin >> weight ;
        }
        void show data ( )
        {
        cout <<"Name" << name ;
        cout <<"Age" << age ;
        cout <<"weight" << weight ;
    }
};
    void main ( )
    {
        B obj ;
        obj. getdata ( ) ;
        obj. show data ( ) ;
    }
```

In this program class A is the base class and have two data members name and age. These data members are accessible from any class that is derived class which adds one data member weight and two member functions getdata ( ) and show data ( ).

Public inheritance and private inheritance : C++ provides different ways to access class members one such access control mechanism is the way derived classes are declared. In public inheritance, the keyword public is used. The keyword public specifies that objects of the derived class are able to access public member functions of the base class.

And for private inheritance, the private keyword is used. When this keyword is used, objects of the derived class cannot access public member functions of the base class. Since objects can never access private or protected members of a class, the result is that no member of the base class is accessible to objects of the derived class.

Q 16. Is it possible to inherit a base class in protected mode ?

Ans. Yes, It is possible to inherit a base class in protected mode. In protected derivation, both the public and protected members of the base class become protected members of the derived class.

Q 17. Explain multiple inheritance with the help of example.

Ans. Multiple inheritance allows us to combine the features of several existing classes as a starting point for defining new classes.

Example :

```
#inlcude <iostream>
class M
{
```



```
    protected;
    int m;
    public:
    void get_m (int);
};
    class N
{
    protected;
    int n;
    public
    void get_n (int);
};
    Class P : public M, public N
{
    public;
    void display (void);
};
    void M : : get_m (int x)
{
    m = x;
}
    void N : : get_n (inty)
{
    n = y;
}
    void P : : display (void)
{
    cout <<"m = " <<m <<"/n";
    cout <<"n = " <<n <<"/n";
    cout <<"m*n = "<<m*n<<"/n";
    int main ( )
    {
        Pp;
        p : get_m(10);
        p. get_n (20);
        p.display ( );
        return 0;
    }
```

Q 18. How protected base class inheritance is implemented?

Ans.

```
#include <iostream>
class base
{
protected:
int i, j;
public:
void setij (int a, int b).
{
i = a;
```

```
J = b;
}
void show ij ( )
{
cout <<"\n:" <<i <<"\nJ: <<j;
};
class derived : protected base
{
int K;
public :
void set K ( )
{
set ij ( );
K = i + j;
}
void showall ( )
{
Cout <<"\nK:" <<K <<Show ( );
}
};
int main ( )
{
derived ob;
Ob.setK( );
ob.showall ( );
return 0;
}
```

Q 19. How constructors are handled in multiple inheritance? **(PTU, Dec. 2007)**

Ans. Multiple Inheritance : A class can be derived from more than one class. This is called multiple inheritance. The syntax for multiple inheritance is similar to that for single inheritance like

```
Class A
{
} ;
Class B
{
} ;
Class C : public A, public B
{
} ;
```

Here class C is derived from two classes A and B. This is multiple inheritance.

Constructors : Constructors in multiple inheritance are handled in the same way as member functions. We can include zero-arguments or multi argument constructors in base and derived class. When zero-argument constructor is included in base and derived class then the names of base-class constructors follow the colon and are separated by commas are included in derived class. Similarly, when with argument construtors are used in classes. Then the derived class constructor must include arguments of base class constructor with it own arguments. Because when derived class constructor calls the base-class constructor then value of their arguments will be supplied by derived class constructor. e.g.


```
#include <iostream.h>
class A
{
        private :
                int no ;
                float no1 ;
        public :
        A ()
        {
                no = 3 ;
                no1 = 3.5 ;
        }
        A (int a, float b)
        {
                no = a ;
                no1 = b ;
        }
        void get ( )
        {
                cout <<"Enter integer & floating no." ;
                cin >>no>> no1 ;
        }
        void show ( )
        {
        cout <<"No. is" << no<<"Floating is" <<no1 ;
        }
} ;
Class B
{
        private :
                char ch ;
        public :
                B ( )
                {
                        ch = 'A' ; }
                B (char C)
                {
                        ch = C ; }
        void get ( )
        {
                cout <<"Enter character" ;
                cin >>ch : }
        void show ( )
        {
                cout <<"character is" <<ch ;
        }
} ;
Class C : public A, public B
```

```
{
    private :
        int n2 ;
    public :
        C ( ), A ( ), B ( )
        { n2 = 5 ; }
    (C int r, float s, char t, int u) : A (r, s), B (t)
    }
        n2 = u ;
    }
    void get ( )
    {
        A ( ) : : get ( ) ;
        B ( ) : : get ( ) ;
        cout <<"Enter any no." ;
        cin >> n2 ;
    }
    void show ( )
    {
        A ( ) : : show ( ) ;
        B ( ) : : show ( ) ;
        cout <<" No. is" << n2 ;
    }
} ;
void main ( )
{
    C obj ;
    cout <<"Data is" ;
    obj. get ( ) ;
    C obj1 (5, 5.5, 'B', 8) ;
    cout <<"Data using zero argument constructor" ;
    obj. show ( ) ;
    cout <<"Data using multi argument constructor" ;
    obj1. show ( )
}
```

In above example derived class C object will call constructors of base classes A and B and then it own constructor.

Q 20. How are Constructors and destructors invoked in derived classes? What actually happens when a destructor is invoked? **(PTU, May 2014)**

Ans. In an inheritance hierarchy, each default constructor invokes its parent's default constructor before it executes itself and each destructor invokes its parent's destructor after it executes itself. The effect is that all the parent default constructors executes in top-down order, and all the parent destructors execute in bottom up order.

```
class Person
{ public:
Person (const char* s)
{ name = new char[strlen(s) + 1]; strcpy(name.s); }
~Person() { delete [] name : }
```



```
protected:
char* name:
};
class Student : public Person
{ public:
Student (const char* s, const char* m): Person(s)
{ major = new char [strlen(m)+1] : strcpy (major, m); }
~Student() {delete [] major ; }
private :
char*major:
};
int main ()
{ Person x("Ram");
{ Student y("Medha", "Biology");
}
}
```

When x is instantiated, it calls the Person constructor which allocates 4 bytes of memory to store the string "Ram", Then y instantiates, first calling the Person constructor which allocates 6 bytes to store the string "Medha" and then allocating 8 more bytes of memory to store the string "Biology" immediately, scope of y treminates because it is declared within an internal block. At that moment, y's destructor deallocates the 8 bytes used for "Biology" and then calls the Person destructor which deallocates the 6 bytes used for "Medha". Finally, the Person destructor is called to destroy x, deallocating the 4 bytes used for "Ram".

Q 21. How do the different constructors and destructors behave in an inheritance hierarchy? **(PTU, May 2005)**

Ans. Constructors in inheritance : In case of inheritance, usually the objects of derived class are created instead of base class. So, derived class has a constructor and pass arguments to the constructor of the base class. When an object of the derived class is created, the constructor of the base class is executed first and then the constructor of the derived class.

Destructors in inheritance : Unlike constructor, destructors in the class hierarchy are invoked in the reverse order of constructor. The destructor of that class, whose constructor was executed last while building the object of derived class, will be executed first whenever the object of the derived class goes out of scope.

When the object of the derived class goes out of scope, the destructors in the class hierarch destroy their own part of sub object, i.e. part of the object that corresponds to that class.

Q 22. How parameters are passed to base class constructors.

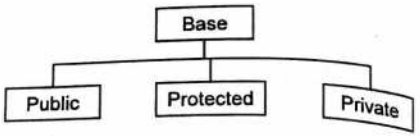
Ans.

```
Class X
{
public;
X (int_, int b_) {a = a_; b = b_;}
private:
int a;
int b;
}
Classs y : public x
{
y ( ) : X (10, 20)
{ };
};
```

Q 23. What are the implications of public, protected and private visibility modes ? (PTU, Dec. 2017 ; May 2017)

Ans. You can declare a derived class from a base class with different access control i.e. public inheritance protected inheritance or private inheritance.

```
#include <iostream.h>
using namespace std;
class base
{
.....
};
class derived; access_specifier base
{
.....
};
```



Example of public, protected and private inheritance in C++

```
Class base
{ public;
int x;
protected:
int y;
private :
int z;
};
Class public Derived : public base
{     // x is public
      // y is protected
      //z is not accessible from public Derived
};
Class protected derived : protected base
{
      //x is protected
      // y is  protected
      // z is not accessible from protected Derived
};
Class private derived : private base
{     // x is private
      // y is private
      // z is not accessible from private Derived
};
```

In the above example, we observe the following things :

- base has three member variables ; x, y and z which are public, protected and private member respectively.
- public Derived inherits variables x and y as public and protected. z is not inherited as it is a private member variable of base.
- protected Derived inherits variables x and y. Both variables become protected. Z is not inherited. If we derive a class derived from protected Derived from protected Derived, variables X and Y are also inherited to the derived class.
- private Derived inherits variables X and Y. Both variables become private. Z is not inherited

If we derive a class derived from private Derived from private Derived, variables x and y are not inherited because they are private variables of private Derived.

Q 24. Create a class String with the following attibutes and declare appropriate constructors and methods

char *str
int maxlen (default = 80)
int length

- **Does the String class require a copy constructor ? If required, define one.**
- **Define a destructor for the String class. What would happen, if you do not define a destructor? (PTU, May 2015)**

Ans. Yes the string class require a copy constructor. Here is the example that defines constructor and destructor for a string class.

```
Class string
{
String (char * astr)
{
str = new char [size of (astr)];
strcpy (str, astr);
}
string (string & strob])
{
tmpstr = strob].get chars ( );
str = newtmstr [size of (tmpstr)];
strcpy (str, tmpstr);
}
~ string ( )
{
del str;
}
char * str;
};
```

A destructor is a special member function that is called when the lifetime of an object ends. The purpose of the destructor is to free the resources that the object may have acquired during the lifetime. The destructor is called whenever an objects is lifetime ends, which includes.

- Program termination
- thread exit
- end of scope
- delete expression
- end of the full expression
- stack unwinding.

Q 25. What is reusability of code ? How it is achieved in C++ ? Define a base class called Animal with three members : Name, Age and a method to display name and age of the Animal. Derive two classes Cat and Dog from Animal class and write a driver program to create Cat and Dog objects with suitable value. Display the contents of the objects by calling the Display method on the derived objects. (PTU, May 2015

Ans. Reusability : In computer science and software engineering reusability is the use existing assets in some form within the software product development process. It helps you to reus the existing code rather than trying to create the same over again. Inheritance helps the code to

reused in many situations. The base class is defined and once it is compiled, it need not be reworked. Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed. Inheritance is one of the most powerful features of object oriented programing language.

Program :

```
#include <iostream.h>
#include <conio.h>
class animal.
{
char name [10];
int age;
public;
void getdata ( )
{
{
cout <<"\n Enter name";
cin.get (name, 10);
cout<<"\n Enter age";
cin> > age;
}
void putdata ( )
{
cout <<"\n The name is ;" <<name<<endl;
cout <<"\n The age is ;" <<age <<endl;
}
};
Class derived cat : public animal
{
charname[10];
int age;
public
void indata ( )
{
cout <<"\n. enter the name";
cin. get (name, 10);
cout <<"\n enter age";
cin >> age;
}
void outdata ( )
{
cout <<"\n The name is :" <<name<<endl;
cout <<"\n The age is :" <<age<<endl;
}
class derived dog: public animal
{
char name [10];
int age;
public
```



```
void input ( )
{
cout <<"\n Enter name";
cin.get (name, 10);
cout <<"\n Enter age";
cin >> age;
}
void output ( )
{
cout <<"\n the name is" <<name <<endl;
cout <<"\n the age is" <<age <<endl;
}
};
void main ( )
{
derivedcat obj1;
deriveddog obj2;
obj1. getdata ( );
obj1. indata ( );
obj2. getdata ( );
obj2. input ( );
obj1. putdata ( );
obj1. outdata ( );
obj2. putdata ( );
obj2. outdata ( );
getch ( );
}
```

Q 26. What do you mean by Multiple Inheritance ? Explain with the help of an example. (PTU, Dec. 2015)

Ans. A class gets inherited from one or more classes are given in the figure this is known as multiple inheritance. Multiple inheritance allows to combine features several existing class for defining one new class. The syntax for derived class using multiple inheritance is as follows :

```
Class B1
    { };
Class B2
    { } ;
Class Bn
    { };
Class D : visibility mode B1, Vm B2, ....., Vm Bn;
    {
    };
```

Where visibility mode may be public or private. For example : declare a class p derived from m & n class m has one variable & one member function class n has one private variable & one public member function where class n is privately inherited class n is public give class p representation.

```
Class m
    {
    int x;
    public : void get d ( );
    };
```

```
Class n
    {
    private :
    int ( );
    public:
    void sum  ( ); }
class p: private m, private n;
    {
    public:
    void disp ( );
    }
```

Q 27. What are virtual constructors ? Give relevent examples to explain it.
(PTU, May 2016)

Ans. A constructor cannot be virtual. There are some valid reasons that justify this statement. First, to create an object the constructor of the object class must be of the same type as the class. But, this is not possible with a virtually implemented constructor. Second, at the time of calling a constructor, the virtual table would not have been created to resolve any virtual functions calls. Thus, a virtual constructor itself would not have anywhere to look up to. As a result, it is not possible to declare a constructor as virtual.

Q 28. Explain how to call a base member function from derived class member function.
(PTU, Dec. 2016 ; May 2016)

Ans.
```
class emp
{
public:
void empName()
{
cout<<"emp-XYZ"<<endl;
}
};
class dept : public emp
{
public:
void empName ()
{
cout<<"dept–XYZ"<<endl;
emp::empName();
}
};
```

Following example shows how stream operators are overloaded :

```
#include <iostream>
using namespace std;
class Distance
{
Private;
int feet;
int inches;
public :
Distance ( ) {
feet = 0;
```



```
inches = i;
}
friend ostream & operator <<(ostream & output, const Distance & D)
{
output <<"F". << D. feet <<"I"<<D.inches;
return output;
}
friend istream & operator >> (istream & input, Distance & D)
{
input >> D. feet >> D. inches
return input;
} 3;
int main ( ) {
Distance D1 (1, 10) D2 (5, 11), D3 ;
cout <<"Enter the value of object" <<endl;
cin >>D3;
cout <<"First Distance; "<<D1 <<endl;
cout <<"Second Distance;" <<D2<<endl;
cout <<"Third Distance;" <<D3<<endl;
return 0;
}
```

Q 29. What are the advantages of scope resolution and referencing ?
(PTU, Dec. 2016, 2014)

Ans. Advantages of scope resolution : The scope resolution operator (::) is used to define a function outside a class or when we want to use a global variable but also has a local variable with same name.

The :: (scope resolution) operator is used to qualify hidden names so that you can still use them. You can use the unary scope operator if a namespace scope or global scope name is hidden by an explicit declaration of the same name in a block or class.

For example :

```
int count = 0;
int main(void) {
int count = 0;
:: count = 1; //set global count to 1
count = 2; //set global count to 2
return 0;
}
```

The declaration of count declared in the main() function hides the integer named count declared in global namespace scope. The statement :: count = 1 accessess the variable named count declared in global namespace scope.

Advantages of referencing : The address where a variable is stored in memory can be referred using address or reference operator.

For example : The address of a variable ; can be accessed by the expression &i, when & is an address of reference operator that evaluates the address of its operand. We can also say that the reference operator & is used to simplify the usage of pointers in C++.

Q 30. What is virtual destructor ? What is the use of declaring it under multiple inheritances?
(PTU, May 2017

Ans. Virtual Destructor : If an object (with a non-virtual destructor) is destroyed explicitl

applying the delete operator to a base-class pointer to the object, the base class destructor function (matching pointer type) is called on the object. There is a simple solution to this problem, declare a virtual base-class destructor. This makes all derived class destructor virtual even though they don't have the same name as the base class destructor, if the object in the hierarchy is destroyed explicitly by applying the delete operator to a base-class pointer to a derived class object, the destructor for the appropriate class is called.

A progam to demonstrate how to define, declare and invoke the virtual destructor member function in multiple inheritance using the polymorphic technique.

```
#include <iostream.h>
Class base {
public :
virtual void display ( );
virtual ~base ( );
};
class derived : public base
{
public :
virtual void display ( );
virtual ~derived ( );
}
Void base : : display ( )
{
cout <<"Base class member function";
cout <<endl;
}
base : : ~base ( )
{
cout <<"base class destructor is called";
cout <<endl;
}
void derived : : display ( )
{
cout <<"Derived class member function";
cout <<endl;
}
derived : : ~derived ( )
{
cout <<"derived class destructor is called";
cout <<endl;
}
void main ( )
{
base * ptr = new derived;
ptr →display ( );
delete ptr;
}
```

❑❑❑

Chapter 4

Virtual Function and Polymorphism

Contents

Virtual function, calling a Virtual function through a base class reference, Virtual attribute is inherited, Virtual functions are hierarchical, pure virtual functions, Abstract classes, Using virtual functions, Early and late binding.

POINTS TO REMEMBER

- Polymorphism is the ability of the objects to take different forms.
- The virtual functions allow programmers to declare functions in a base class and redefine the same with the same name in its derived classes.
- The prototype of virtual function in base class and derived class must be identical.
- A pure virtual function has no implementation in the base class, hence, a class with pure virtual function cannot be instantiated.
- A class containing pure virtual functions cannot be used to define any objects of its own. Such classes are called abstract classes.
- Polymorphism is technique that allows defining various forms of a single function that can be shared by various object.
- Polymorphism can be implemented at compile time as well as run time.
- A compile time, polymorphism is implemented using function overloading and operator over loading.
- At run time, polymorphism is implemented using virtual functions.
- Functions and operator over loading are the examples of compile time polymorphism.
- In run time polymorphism, an appropriate member function is selected while the program is running. C++ support the run time polymorphism with the help of virtual function.
- Binding is the process of typing the function call to function defination.
- When binding happens at compile time, it is known as early binding.
- When binding happens at run time, it is known as late binding.
- To implement run time polymorphism you need to make a function virtual and access the virtual function through pointer to the base class.
- A function in a class prefixed by keyword virtual becomes virtual function.
- If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class. In such cases, the respective calls will invoke the base class function.
- A virtual function, equated to zero is called a pure virtual function.

QUESTIONS-ANSWERS

Q 1. What do you mean by binding?

Ans. Binding refers to the process that is used to convert identifiers (such as variable and function names) into machine language addresses. Although binding is used for both variable and functions. OR

Binding refers to the linking of a procedure call to the code to be executed in response to the call.

Q 2. What is the difference between static and dynamic binding? **(PTU, May 2006, 2005)**

OR

Explain in brief bindings in C++. **(PTU, Dec. 2007)**

Ans. Static Binding : The information about the number and type of arguments is available the compiler at the time of compilation. Therefore, the compiler is able to select the function for particular call at the compile time. This is called early or static binding. It is early in the sense that a function definition is bound to its call at the compile time and static in the sense that, this binding cannot be changed at execution time.

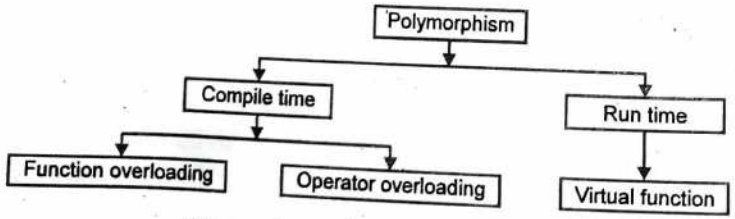
Dynamic Binding : Sometimes, it is required to select member function at that time. At run time, when it is known what objects are under consideration, the appropriate member is invoked. Since the function is linked with a particular class after the compilation i.e. during program execution, the process is called late or dynamic binding. It is late in the sense that a function definition is bound to its call at the execution time and dynamic in the sense that this binding can be changed at execution time.

Q 3. What do you mean by polymorphism? **(PTU, Dec. 2011, 2008)**

Ans. Polymorphism is one of the most powerful feature of OOP language and is one of the key feature that distinguishes an OOP language from other languages. Polymorphism allows a programmer to design a highly generalised class with operations that are common to all classes derived from the class and at the same time allow derived classes to add additional operations specific to the derived classes. Thus, polymorphism gives a programmer the opportunity to design a common interface based on a base class which dictates the general operation of the interface, while derived class define multiple specialised operations. This is why polymorphism is frequently referred to as 'one interface, multiple method'.

Q 4. Explain the types of polymorphism. **(PTU, Dec. 2011)**

Ans.



Types of polymorphism in C++

(i) Compile time : A compile time polymorphism in which a function name can operate in a

variety of different ways. Compile time polymorphism is of two types which are :

- **Function overloading :** It performs the different types of task using single function.
- **Operator overloading :** It is the process of making an operator to exhibit different behaviour in different instances.

(ii) Run time polymorphism : It exists when dynamic binding and inheritance combine. C++ support run-time polymorphism through the use of pointer to derived class and virtual function.

- **Virtual function :** When the form of a member function is changed at run time, that member function is referred to as virtual function.

Q 5. What do you mean by pure virtual function? **(PTU, May 2012)**

Ans. If a virtual function is not defined in the base class in which it is first declared, then it is referred to as pure virtual function. The declaration syntax of a pure virtual function is

Virtual return_type_specifier member function (/* ..... */) = 0 ;

Q 6. Why do we need virtual functions? Explain with the help of an example. **(PTU, May 2013)**

Ans. Virtual function should be used when you want to provide default behaviour in the base class. The child classes can then override this function and provide their own, more specific behaviour. Virtual function is used to reduce or solve the ambiguity (confusion) arise in C++ programming.

For example :

```
Class Animal
{
public : virtual void say ( )
{
    // default behaviour
    console.writeline ("Animal makes generic noise") ;
}
} ;
    Class Dog : Animal
    {
        public : override void say ( )
        {
            Console.Writeline ("Dog barks") ;
        }
    } ;
```

Virtual function works only in polymorphism.

Q 7. What are the rules for virtual function?

Ans. The following are the rules for virtual functions :

1. The virtual function must be member function of some classes.
2. They cannot be static member of the class.
3. They can be friend function to another classes.
4. They are accessed using object pointer.
5. A virtual function is a base class must be defined even though it is not used.
6. The prototype of virtual function in the base class and derived class must be identical.
7. The class cannot have virtual constructor, but can contain a virtual destructor.
8. To realize the potential benefits of virtual functions supporting run-time polymorphism, t should be declared in the public section of the class.

Q 8. What do you mean by abstract class? (PTU, Dec. 2009 ; May 2008)

Ans. A class containing pure virtual functions cannot be used to define any objects of its own. Such classes are called abstract classes. Abstract classes can be used as a framework upon which new classes can be built to provide a new function.

Q 9. What are the virtual destructor?

Ans. If an object (with a non-virtual destructor) is destroyed explicitly applying the delete operator to a base-class pointer to the object, the base class destructor function (matching pointer type) is called on the object. There is a simple solution to this problem, declare a virtual base-class destructor. This makes all derived class destructor virtual even though they don't have the same name as the base class destructor, if the object in the hierarchy is destroyed explicitly by applying the delete operator to a base-class pointer to a derived class object, the destructor for the appropriate class is called.

Q 10. Can we make a destructor virtual? What purpose a virtual destructor will serve?

Ans. Yes, we can make a destructor virtual. We can make destructor of base class as virtual. This makes all derived class destructors virtual even they do not have the same name as the base class destructor. Now, if the object in the hierarchy is destroyed explicitly by applying a delete operator to a base class pointer to the derived class object, the destructor for the appropriate class is invoked. The base class destructor automatically executed after the derived class destructor.

Q 11. Can we make a constructor virtual? If not, why?

Ans. Constructor cannot be virtual. Declaring a constructor as a virtual is a syntax error. The reason why constructors cannot be virtual is that virtual mechanism works on objects, an object exists only after the successful execution of constructor.

Q 12. What is virtual function? How we can declare the virtual function? Also explain the need for virtual function.

Ans. Virtual Function : A member function whose function declaration is preceded by virtual keyword is known as virtual function. These functions are defined in a base class in the public section and they provide a mechanism by which the derived clas can override it. These functions are bound dynamically. The syntax for defining a virtual function is

```
Classs ABC
{
Private :
.....
Public ;
Virtual return type function name (arguments)
{
..... //body of the virtual function
}} ;
```

At run time, it allows to decide which overriden form of the function is to used based on the type f object pointed by the base pointer rather than the type of derived pointer.

Declaring function virtual

Let us define a class CREATURES

```
Class CREATURES
{
Public :
char name [40] ;
CREATURES (char ≠ P) ;
```



```
void virtual move ( ) ;
} ;
```

We have added keyword virtual to the base class function move ( ).

Need of virtual function : Logically when a pointer is pointing to derived class object, method from the corresponding derived class should be executed. When the pointer in question is a base class pointer this is not lapping. Somehow the language must have a provision to circumvent this difficulty. This calls for the concept of virtual function. If we declare such a function as virtual in the base class then the problem sets solved.

Q 13. Differentiate between static and runtime polymorphism giving example. (PTU, Dec. 2010)

Ans. Compile time polymorphism is achieved using Operator and function.

Here same function name can be used for different functions with argument differences.

Function overloading

```
Void CompileTime ( int param);
Void CompileTime ( int param, int param);
Void CompileTime( double param );
CompileTime(10,20);
CompileTime(10.5);
```

Operator overloading

Default operator's works fine for built-in data types. However to use them on user defined data types, User needs to implement their own operators to operate on user defined object types.

Runtime polymorphism can be achieved using virtual functions.

In virtual function mechanism, any derived class function can be called using Base class pointer. Function declared inside that particular class will be called of which object is assigned to base class pointer.

```
Virtual void PrintClassName()
{
Cout << "Level1Derived " << endl :
}
};
Class LevelOneDerived : public Base
{
Virtual void PrintClassName()
Cout << "LevelOneDerived " << endl ;
Class LevelTwoDerived : public Base
Virtual void PrintClassName()
Cout << "LevelTwoDerived " << endl ;
Base *pBasePointer = NULL;
LevelOneDerived levelOneObject;
LevelTwoDerived levelTwoObject;
pBasePointer = &baseObject;
pBasePointer->PrintClassName();
pBasePointer =&levelOneObject;
pBasePointer->PrintClassName();
pBasePointer =&levelTwoObject;
pBasePointer->PrintClassName();
```

Q 14. What is the concept of late binding? How the pointers play role to achieve late binding? **(PTU, Dec. 2008)**

OR

What is the concept of late binding? Discuss the role of the pointers to achieve late bindings. Explain the concept with the help of an example. **(PTU, Dec. 2011)**

Ans. Late Binding : When it is appropriate that member function could be selected while the program is running. At run time, when it is known what objects are under consideration the appropriate version of member is invoked. This is known as run time polymorphism. Since the function is linked with a particular class after the compilation i.e. during program execution, the process is called late binding.

Pointers help in late binding : Run time polymorphism is achieved using following steps :

- preceding the member function with keyword virtual in base class
- accessing the member function through a pointer of base class.

Example :

```
#include <iostream.h>
class Base
{
      private :
            int a ;
      public :
            base ( ) { }
            base (int n)
            {
                  a = n ;
            }
      virtual void show ( ) {
            cout <<"display method of base class" ;
            cout << a;
            }
    } ;
class derived : public base
    {
          private :
                int b ;
          public :
                derived ( ) { }
                derived (int n) {
                      b = n;
          }
          void show ( ) {
          cout <<"Display method of derived class" ;
          cout <<a << b ;
          }
    } ;
    void main ( )
    {
```



```
          base *ptr ;
          base obj (5) ;
          ptr = &obj ;
          ptr → show ( ) ;
          derived obj1 (10) ;
          ptr = &obj1 ;
          ptr → show ( ) ;
    }
```

In above program, all calls to show ( ) member functions are resolved at execution time, where as for other functions, still all calls are resolved at compilation time.

Q 15. Elucidate the concept of early binding with respect to constructor function. **(PTU, Dec. 2008 ; May 2008)**

Ans. The information about the number and type of arguments in available the compiler at the time of compilation. Therefore, the compiler is able to select the appropriate function for particular call at the compile time itself. This is called early binding or static binding. It is early in the sense that a function definition is bound to its call at the compile time and static in the sense that this binding cannot be changed at execution time.

Constructor is a member function who is used to initialize an object of a class when it is created. Its name is same as that of its class. A constructor is automatically invoked when an object of its associated class is created. It is class constructor because it constructs objects with its initial state by assigning initial values to its data members. An object of the class is created during compilation which automatically calls its constructor functions which intialize data members of that class for that particular object. It is known as early binding.

Q 16. Distinguish between virtual functions and pure virtual functions. (PTU, May 2019)

OR

What are virtual functions and pure virtual functions? Explain the use of having abstract classes. **(PTU, May 2012)**

Ans. Virtual function : A member function whose function declaration is preceded by virtual keyword is known as a virtual function. These functions are defined in a base class in the public section and they provide a mechanism by which the derived class can override it. These functions are bound dynamically. The syntax for defining a virtual function is :

```
class ABC
{
      private :
      .....
      public :
      .....
      Virtual return type function name (arguments)
      {
      ..... // body of the virtual function
      }
} ;
```

At run time, it allows to decide which overridden form of the function is to used based on type of object pointed by the base pointer rather than the type of derived pointer.

pure virtual function :

Sometime virtual functions are declared without any body i.e. they don't have any defini

Such virtual functions are called pure virtual functions. These functions are also called do-nothing functions. The syntax of pure virtual function is :

```
class ABC
{
        private :
        .....
        public :
        .....
        Virtual returntype function Name (arguments) = 0 ;
} ;
```

Any derived class of a base class containing pure virtual functions must either define these functions or re-declare them as pure virtual function.

Q 17. How does C++ uses concept of reusability? Write a program in C++ to illustrate use of polymorphism. **(PTU, May 2012)**

Ans. In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. The real appeal and power of the inheritance mechanism is that is allowed the programmer to reuse a class that is almost, but not exactly what he wants, and to tailor the class in such a way that it does not introduce any undesirable side effects into the rest of the classes.

Polymorphism : Polymorphism means ability to take more than one form. An operation may exhibit different behaviour in different instances. The behaviour depends upon the type of data used in the operation. Polymorphism is of two types complie time and runtime polymorphism. Early binding simply means that an object is bound to its function call at compile time.

Now let us consider a situation where the function name and prototype is the same in both the base and derived classes. For example, consider the following class definition :

```
Class A
{
int x ;
public :
void show ( ) {.....} //show ( ) in base class
} ;
Class B : Public A
{
int y ;
public :
void show ( ) {.....} //show ( ) in derived class
} ;
```

Q 18. What is a virtual member function? Explain with example.

(PTU, Dec. 2013)

Ans. A virtual member function is a member function preceded by the keyword virtual or a member function with the same signature as a virtual function declared in a base class. In this context virtual means "overridable". More specifically, the keyword virtual means that the runtime system automatically invokes the proper member function when it is overridden by a derived class (dynamic binding).

A member function should be made virtual when there will be derived classes that will need to provide their own implementation for the member function. This doesn't require as much clairvoyance

as it seems to imply. Normally the virtual function represent specifically architected places where extensibility is supposed to take place. Overriding a virtual member function is also straight forward ; simply declare the member function in the derived class and define a new implementation for that member function.

Q 19. Give an example where virtual functions are hierarchical.

Ans.

```
#include <iostream>
class base {
public;
virtual void vfunc ( ){
cout <<"This is derived is v func ( ).\n";
}
};
class derived1 : public base{
public :
void vfunc ( ) {
cout <<"This is derived is vfunc ( ).\n";
}
};
class derived 2 : public base {
public :
// v func ( ) not overridden by derived 2;
base's is used
};
int main ( )
{
base * p, b;
derived1 d1;
derived 2 d2;
P = &b;
P → vfunc ( );
p = &d1;
p → vfunc ( );
p = &d2;
p → vfunc ( );
return 0;
}.
```

Q 20. Explain virtual function. inheritance with example.

Ans. In C + +, once a member function is declared as a virtual function in a base class, i becomes virtual in every class derived from that base class. In other words, it is not necessary to us the keyword virtual in the derived class while declaring redefined version of the virtual base clas function

e.g.

```
#include <iostream>
class A {
public :
virtual void fun ( )
{ cout <<"\n A : : fun ( ) called";}
};
class B : public A
{
```

```
public:
void fun ( )
{
cout <<"\n B : : fun ( ) Called " ; }
};
Class C : public B {
public;
void fun ( )
{cout <<"\n C : : fun ( ) Called" ; }
};
int main ( )
{
C c;
B *b = &C;
b → fun ( );
get char ( );
}
```

Q 21. How a virtual function is called through a class reference.

Ans.
```
#include <iostream>
Class Base
{
int x;
public:
virtual void fun ( ) = 0;
int get x ( ) { return x ; }
};
Class Derived : public Base
{
int y;
public :
void fun ( ) { cout <<"fun ( ) called "; }
};
int main (void)
{
Derived d;
d.fun ( );
return 0;
}
```

Q 22. What are the different ways to achieve the polymorphism in C++? Explain the pure polymorphism with example. **(PTU, Dec. 2014)**

Ans. The following are the different ways of achieving polymorphism :

- Function Overloading
- Operator Overloading
- Dynamic Binding

Pure Polymorphism : This capability also called parametric overloading, is provided in C++ by the evocation of the same name with different signatures inside the class scope. Pure polymorphism within a class is implemented by creating several methods with different signatures. Based on static binding, a parametric overloaded member function is recognized by the number, types and order of the arguments in an invocation.

In C++, pure polymorphism is achieved through inheritance and virtual functions.Pure polymorphism occurs when a single function can be applied to arguments of a variety of type. In pure polymorphism, there is one function (code body) and a number of interpretations.

Other words, we can also say that many authors reserve the term polymorphism (or pure polymorphism) for situations where one function can be used with a variety of arguments, and the term overloading for situations where there are multiple functions all defined with a single name. Such facilities are not restricted to object-oriented languages. In Lisp or ML, for example, it is easy to write functions that manipulate lists of arbitrary elements; such functions are polymorphic, because the type of the argument is not known at the time the function is defined. The ability to form polymorphic functions is one of the most powerful techniques in object oriented programming. It permits code to be written once, at a high level of abstraction, and to be tailored as necessary to fit a variety of situations. Usually, the programmer accomplishes this tailoring by sending further messages to the receiver for the method. These subsequent messages often are not associated with the class at the level of the polymorphic method, but rather are deferred methods defined in the lower classes.

Q 23. Differentiate between the term Dynamic and static memory allocation. **(PTU, May 2015)**

Ans. Dynamic and static memory allocation :

Static allocation means that the memory for your variables is automatically allocated either on the stack or in other sections of your program. You do not have to reserve extra memory using them but on the otherhand have also no control over the lifetime of this memory.

e.g. a variable in a function is only there until the function finishes.

```
Vdd func ( )
{
int i ; /* 'i' only exists during 'func'*/.
}
```

Dynamic memory allocation is a bit different. You now control the exact size and the lifetime of these memory locations. If you don't free it, you'll run into memory leaks, which may cause your application to crash since it, at same point cannot allocate more memory.

```
int * func ( )
{
int * mem = malloc (1024);
return mem ;
}
int * mem = func ( );
```

In the upper example, the allocated memory is still valid and accessible, even though the function terminated. When you are done with the memory, you have to free it.

Q 24. Explain the use of get and put pointer in file handling. **(PTU, May 2016**

Ans. Each file has two associated pointers known as the file pointers. One of them is calle the input pointer or get pointer and the other is called the output pointer or put pointer. We can us these pointers to move through the files while reading or writing. The input pointer is used f reading the contents of a given file location and the output pointer is used for wirting to a given f location. Each time an input or output operation takes place the appropriate pointer is automatica advanced.

Q 25. What are dangling pointers ? Give example. **(PTU, Dec. 20**

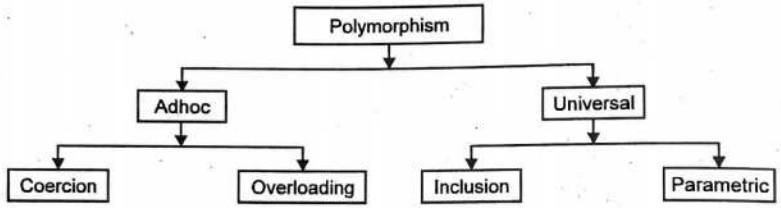
Ans. Dangling pointers : The most common pointer error is to use a pointer that has not b initialized, or that has already been deleted. Such a pointer is called a dangling pointer, becaus does point somewhere, just not to a valid object. You can create real damage by writing the loca to which it points.

We can also say that in this, the user tries to use the pointer after it has been deleted.

```
int *x = new int (12);
*q *=* q; //find the square of value i.e. 144
delete q; //q pointer has been deleted.
cout <<*q; //Error. q has become dangling pointer.
```

Q 26. What is the difference between ad-hoc and universal polymorphism? Discuss with the help of a suitable example. **(PTU, May 2015)**

Ans. Polymorphism refers to the multiplicity of meanings attached to an identifier. Polymorphic stands for 'of many forms'. A polymorphic language selects an operation from a multitude based on its object's type.



Adhoc Polymorphism : Adhoc polymorphism is obtained when a function works or appears to work, on several different types and may behave in unrelated ways for each type.

(i) Coercion : Coerscion covers convertible in argument type to match the type of a corresponding function parameter. It is a semantic operation to avoid a type error. If the compiler encounters a mismatch between the type of an argument and the type of the corresponding parameter, the language allows conversion from the type of the argument to the type of the corresponding parameter. The function definition itself only ever executes on one type that of its parameter. C++ implements coercion at complie time. If a complier succeeds in matching the type of an argument to the type of the corresponding parameter in the function call, the compiler inserts the conversion.

Code immediately before the functions call.

Coercion may

- narrow the argument type
- widen the argument type

For e.g.

```
# include <iostream>
void display (int a ) const {
std : : cout << "one argument ("<<a <<");
}
int main ( )
{
display (10);
std : : cout << std : : endl;
display (12,6);
std : : cout <<,std : : endl;
display ('A');
std : : cout << std : : endl;
}
```

(ii) Overloading : Overloading covers accepted variations in a function's definition to match the argument types to corresponding parameter types. It is a syntactic abbreviation that associates the same function identifier with a variety of function definitions by distinguishing its parameter sets. The same

function name can be used with a variety of unrelated argument types. Each set of argument types has its own function definition. The compiler binds the function call to the matching function definition.

e.g.

```
# include <oistream>
void display ( )
const {
std : : cout << "No arguments";
}
void display (int a ) const
{
std : : cout << "one argument ("<< a<<");
}
int main ( )
{
display ( );
std : : cout << std : : endl;
display (10);
std : : cout << std : : enld;
}
```

Universal Polymorphism : Universal polymorphism is true polymorphism. Its polymorphic character survives at closer scrutiny. Universal polymorphism imposes no restriction on the admissible types. The same function applies to a potentially unlimited range of different types.

Inclusion : Inclusion polymorphism covers selection of a member function definition from a se of definitions based on an object's type. The type is one of the types belonging to an inheritance hierarchy. The term inclusion refers to one type including another type within the hierarchy. All functio definitions share the same name throughout the hierarchy.

Parametric : Parametric polymorphism covers definitions that share identical logic independentl of type. The logic is common to all possible types without restriction. The types need not be related i any way. For example, a function that sorts in its uses the same logic as a function that sort doubles. C++ implements parametric polymorphism at compile time using template syntax.

Q 27. Discuss pointer arithmetic of C++ with examples. **(PTU, Dec. 201**

Ans. As a pointer holds the memory address of a variable, some arithmetic operations can b performed with pointers. C++ supports four arithmetic operators that can be used with pointers, such a

Addition +
Subtraction –
Incrementation ++
Decrementation – –

Pointers are variables. They are not integers but they can be displayed as unsigned intege The conversion specifier for a pointer is added and subtracted.

ptr + + / ptr – – > it causes the pointer to be incremented/decremented, but not by 1.

The integer value would occupy bytes 2000 and 2001

```
int value, + ptr;
value = 120;
ptr = &value;
ptr + +;
cout <<ptr;
```

It will display 2002.

❑❑❑

Chapter 5

Exception Handling

Contents

Basics of exception handling, exception handling mechanism, throwing mechanism, catching mechanism, I/O System Basics, File I/O: Exception handling fundamentals, Exception handling options. C++ stream classes, Formatted I/O, fstream and the File classes, Opening and closing a file, Reading and writing text files.

POINTS TO REMEMBER

- In C++ errors can be divided into two categories : compile time errors and run time errors.
- Compile time errors are syntactic errors which occurs during the writing of the program.
- The logic errors occur due to the poor understanding of the problem and solution procedures.
- Logical errors cause the unexpected or unwanted output.
- Exceptions are runtime errors which a programmer usually does not except.
- Exception occurs accidently which may result in abnormal termination of the program.
- Exception handling mechanism is used to trapesception and running program smoothly after catching the exception.
- Or exception handling provides a type-safe, integrated approach, for coping with the unusual predictable problems that arise while executing a program.
- Exceptions are of two kinds : synchronous exceptions and asynchronous exceptions.
- Synchronous exceptions are those which occur during the program execution due to the some fault in input data.
- Asynchronous exceptions caused by events or a fault unrelated to the program and beyond the control of the program.
- Exception handling is built upon three keywords : try, catch and throw.
- The purpose of the exception handing mechanism is to provide means to detect and report an "exceptional circumstance" so that appropriate action can be taken.
- The keyword try is used to preface a block of statements which may generate exceptions. This block of statement is known as **try** block.
- When an exception is detected, it is thrown using a **throw** statement in the try block.
- A **catch** block defined by keyword catch 'catches' the exception 'thrown' by the throw statement in the try block and handles it appropriately.
- The general form of using an exception specifications is

```
type function (org_list) throw (type_list)
{
    .....
    ..... function body
    .....
}
```


- We can place two or more catch block together to catch and handle multiple type of exceptions thrown by a try block.
- We may restrict a function to throw only a set of specified exceptions by adding a throw specification clause to the function definition.
- Input output function of C++ works with different physical devices. It also act as interface between the user and the device.
- Console I/O operations are used to accept input from standard input device and direct the output to standard output device.
- A stream is a series of bytes that acts as a source and destination for data. The source stream is called input stream and the destination stream is called output stream.
- The cin, cout, cen and log are predefined stream.
- The header file to stream.h must be include when we use cin and cout functions.
- The istream and ostream are derived classes from iosbase class.
- The formatting of output can be effectively done with mernber functions of ios class. The member function width( ), precision( ), fill( ) and setf( ) allows user to design and display the output in formatted form.
- The putback( ) replaces the given character into the input stream. The member function igore, ignores a number of given characters till it finds termination character.
- Manipulators also help the user in formatting of output. The programmer can also create his/her own manipulators.
- The header file iomanip.h contains pre-defined manipulators.
- A file is a collection of related information normally representing programs, both source and object forms and data.
- A stream is a general name given to a flow of data.
- There are two type of streams input stream and output stream.
- Input stream read the data from a disk file.
- Output stream is used to write data into the file.
- Files are either text files or binary files.
- If stream is the class for input operation on files similar to standard input.
- If stream includes the methods open ( ), close ( ), read ( ), get ( ), getline ( ).
- Of stream is the class for output operation on files similar to the standard output.
- Of stream methods are open ( ), close ( ), put ( ), write ( ).
- Fstream class derived from both ifstream and ofstream.
- The function put ( ) writes a single character to the associated stream.
- The function get ( ) reads a single character from the associated stream.
- Bad ( ) function returns true if a failure occurs in a reading or writing operation.
- The function fail ( ) returns true in the same cases are bad ( ) plus in case that a format error happens, as trying to read an integer number and an alphabetical character is received.
- eof ( ) function returns true if a file opened for reading has reached the end.
- Good ( ) function is the most generic : returns false in the same cases in which calling any of the previous functions would return true.
- The file management system associates two pointers with each file known as file pointers : input pointer and output pointer.
- A file accessed sequentially means that all the preceding data items how to be read and discarded while accessing a particular data item.
- In random access, the file pointer can directly reach the location of interest.

QUESTIONS-ANSWERS

Q 1. What is an exception? **(PTU, Dec. 2010)**

Ans. An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that aries while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords : try, catch and throw.

Q 2. Explain the different types of exceptions.

Ans. The exception are of two kinds :

1. Synchronous Exception : The exceptions which occur during the program execution, due to some fault in input data, within the program, is known as synchronous exception.

2. Asychronous Exception : The exceptions caused by events or a fault unrelated to the program and beyond the control of the program is known as asychronous exception.

Q 3. Explain the meaning of keywords try, throw and catch.

Ans. Try : This keyword defines a boundary within which an exception can occur. A block of code in which an exception may occur must be prefixed by this keyword.

Throw : Throw is used to raise an exception when an errors is generated in the computation it initializes a temporary object to be used in throw.

Catch : This keyword represents exception handler. It must be compulsorily used immediately after the statements marked by try keyword. It can also occur immediately after catch keyword. Each handler will only evaluate an exception that matches or can be converted to the type specified in the argument list.

Q 4. What are the tasks to be performed by error handling code?

Ans. (a) Hit the exception : Detect the problem causing exception.

(b) Throw the exception : Inform that an error has occured.

(c) Catch the exception : Receive the error information.

(d) Handle the exception : Take corrective actions.

Q 5. List the functions for handling un caught exceptions.

Ans. (a) Terminate ( ) : It is invoked when an exception is raised and the handles is not found.

(b) Set_terminate ( ) : Allows the user to install a function that defines the program's actions to be taken to terminate the program when a handler for the exception cannot be found.

(c) Unexpected ( ) : This function is called when a function throws an exception not listed in its exception specification.

(d) Set_unexpected ( ) : It allows the user to install a function that defines the program's actions to be taken when a function throw an exception not listed in its exception specification.

Q 6. What do you mean by rethrowing exceptions?

Ans. In some cases, an exception handler may process an exception, then either rethrow the same exception or throw a different exception. If the handler wants to rethrow the current exception, it can just use the throw statement with no parameters. This instructs the compiler to take the current exception object and throw it again. For example

```
catch (EIntegerRange & range Err) {
// code here to do local handling for the exception
   throw ; // rethrow the exception
}
```


Q 7. Write a short note on exception specification.

Ans. It is possible to specify which exceptions a function may throw. It is run time error to throw an exception of the wrong type past a function. The syntax for an exception specification is :

```
exception specification :
throw (type-id-list) // type-id-list is optional
type-id-list :
type-id
type-id-list, type-id
The following examples are functions that exception specification
void f1 ( ) ;                    //The function can throw any exception
void f2 ( ) throw ( ) ;//should not throw any exception
void f3 ( ) throw (A, B*) ;      //can throw exceptions publicly derived from A, or a pointer
to publically derived B.
```

Q 8. Write the steps to be performed when an exception is raised.

Ans. 1. The program searches for a matching handler.

2. If a program is found, the stack is unwound to that point.

3. Program control is transferred to the handler.

4. If no handler is found, the program will invoke the terminate function if no exception are thrown, the program executes in the normal fashion.

Q 9. What are the blocks used in the exception handling?

Ans. The exception handling mechanism uses three blocks :

1. Try block
2. Throw block
3. Catch block

The try block must be followed immediately by a handler, which is a catch block. If an exception is thrown in the try block.

Q 10. Explain the advantages of using exception handling in a program. (PTU, May 2019)

Ans. Exception handling provides the following advantages :

1. Separating error handling code from "regular" one : Provides a way to separate the details of what to do when something out of the ordinary happens from the normal logical flow of the program code.

2. Propagating errors up the call stack : Lets the corrective action to be taken at the higher level. This allow the corrective action to be taken in the method that calling that one where an error occurs.

3. Grouping error type and error differentiation : Allows to create similar hierarchical structure for exception handling so group them in logical way.

Q 11. What are exceptions? What are two types of exceptions? Draw the exception handling model.

Ans. Exception Handling : Exceptional handling is the process to handle the exception if generated by the program at run time. The aim of exception handling is to write code which passes exception generated to a routine which can handle the exception and can take suitable action. Any exception handling mechanism must have the following steps.

Step 1. Writing exception class (optional)

Step 2. Writing try block

Step 3. Throwing an exception

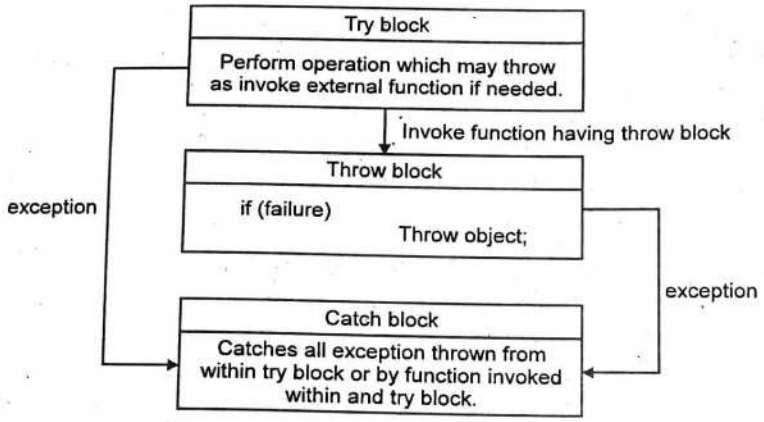
Step 4. Catching and handling othe exception thrown.

Types of Exception Handling : The exception are of two kinds :

1. Synchronous Exception : The exceptions which occur during the program execution, due to some fault in input data, within the program, is known as synchronous exception.

2. Asychronous Exception : The exceptions caused by events or a fault unrelated to the program and beyond the control of the program is known as asychronous exception.

Exception Handling Model :



Q 12. Explain the syntax of try and catch block.

Ans. 1. Try Block : The exception is to be thrown is to be written in the try block. Whenever an exception is generated in the try block, it is thrown. An exception is an object so we can say that an exception object is thrown. The thrown keyword is used for throwing or exception. The usual practice of using the throw statement is as

throw exception ;

for throwing an exception and simply throw ;

For rethrowing or exception

The syntax of try block is as shown below :

```
try
{
statements ;
statements ;
statements ;
throw exception ;
}
```

2. The catch block : An exception thrown by try block is caught by the catch block. A try block must have at least one catch, though there can be many catch block for catching different types of exceptions. A catch block must have a try block prior written which will throw or exception. The catch block is used as ;

```
try
{
statements ;
```



```
statements ;
statements ;
throw exception ;
}
catch (object or argument)
{
statements for handling the exception ;
}
```

The catch block catches any exception thrown by the try block. If exception thrown matches with the argument or object in the block, the statement written the catch block and we say that exception thrown by try block was caught successfully by the catch block. After the successful execution of the catch block statements, any statements following the catch block will be executed. If argument does not match with the exception thrown, catch could not handle it and this may result in abnormal program termination.

Q 13. Explain the exception handling mechanism with example.

Ans. The try, throw and catch all together provide exception handling mechanism in C++. The exception is generated by the throw keyword which is written in the try block. An exception generated within this try block is thrown using this throw keyword. Immediately, following the try block, catch block is written. As soon as some run time errors occurs an exception is thrown by the try block using throw which informs the catch block that an error has occured in the try block. This try block is also known as exception generated block. The catch block is responsible for catching the execution thrown by the try block. When try thrown an exception, the control of the program passes to the catch block and if argument matches as explained earlier, exception is caught. In case no exception is thrown the catch block is ignored and control passes to the next statement after the catch block.

For example :

```
#include <iostream.h>
void main ( )
{
try
{
throw "DEMO of exception" ;
}
catch (char *E)
{
cout <<"Exception caught=" <<E <<endl ;
}
cout <<"continue after catch block" <<endl ;
}
```

Q 14. Write a program to throwing in one function and catching in the other.

Ans.

```
#Include <iostream.h>
void divide {int a, int b}
{
if (b! = 0)
cout <<"a/b=" <<a/b ;
else
```

```
throw b ;
}
int main (void)
try {
divide (4, 2) ;
divide (4, 0) ; }
catch (int X)
{
cout <<"Exception caught = " << X ;
}
return 0 ;
}
```

output a/b = 2
Exception caught = 0.

Q 15. Explain the concept of catch all the exception in a single catch block.

Ans. In many real world programming situations we may not be able to predict all the possible types of exceptions and as a consequence, we will not be able to design independent catch block to handle all the unforeseen exceptions. Fortunately, C++ provides a mechanism of handling this situation, where in a single catch block would suffice to catch all the exceptions.

The syntax of the catch block is as follows :

```
catch (.....)
}
statements
}
```

Program to catching all the exception in a single catch block

```
#include <iostream.h>
int main (void)
{
int X ;
cout <<"Enter a value for X \n" ;
Cin >> X ;
try
P
Switch (X)
{
Case 1 :
throw 'a'
break ;
Case 2 :
throw X ;
break ;
Case 3 :
throw (float X) ;
break ;
Case 4 :
throw "XYZ"
```



```
break ;
}}
Catch (....)
{
cout <<"Exception caught" ;
}
return 0 ;
}
```

Output : Entervalue of X
3
Exception caught.

Q 16. Discuss the exception handling features of C++? **(PTU, May 2010, 2004)**

Ans. Exception handling mechanism depends upon three keywords : try, throw and catch. The keyword try is used to preface the block of statements that may generate exception. This block of statements is known as try block. When an exception is detected, it is thrown using keyword throw in the try block. A catch block that is defined by prefacing a block of statements with the catch keyword, catches the exception thrown by the throw statement is the try block and handles it appropriately. Following is the syntax :

```
Void main ( )
{
.....
.....
try
      {
.....
..... // statements which can cause,
throw exception ;// detect and throw an exception
.....
}
catch (datatype argument)
{
..... // statements that handless
..... // the exception
}
.....
}
```

As soon as the try block throws an exception, the program control leaves the try block and enters the catch block. If the type of the object thrown matches the data type of the argument in the catch statement, then catch block is executed for handling the exception.

Q 17. List the name for pre-defined streams in C++.

Ans.

Stream	Meaning
Cin	Standard input
Cout	Standard output
Cerr	Standard erroroutput
Clog	Buffered version of cerr

Stream cin, cout and cerr correspond to C's stdin, stdout and stderr. By default, the standard stream are used to communicate with the console.

Q 18. What is difference between get ( ) and put ( ) function?

Ans. The get is used to read a character and put ( ) is used to display the character on the screen.

The get function has two syntaxes :

(a) get (char*) ;

(b) get (void) ;

if syntax (a) is used, the get function assigns the read data to its argument, whereas if syntax (b) is used, the get ( ) function returns the data read. The data is assigned to the variable present at left hand side of the assignment operator. Thus functions are members of I/O stream classes and can be called using object.

Put function ( ) : This function is used to display the string on the screen. It is a member of ostream class. The syntax of put ( ) is given below :

(a) cout.put ('A') (b) cout.put (X) ;

The statement (a) displays the characters 'A' on the screen and the statement (b) displays the contents of variable x on the screen.

Q 19. Explain the difference between getline ( ) and write ( ) function.

Ans. The getline( ) and write( ) functions are used with line oriented I/O. The setline function reads a line of text terminated by ENTER key. For example :

```
char arr [21] ;
cin.getline [arr, 21) ;
```

Here, arr is an array of size 21 and can store 20 character at more. When the newline character '\n' is read, it is replaced by the null character in the array.

Write ( ) : This function is used to display the string on screen its format is the same as getline ( ) but the function is exactly opposite. The syntax is given below :

cout.write (variable, size) ;

For example :

```
void main ( )
{
cout.write ("INDIA", 6) ;
cout.write ("IS", 3) ;
cout.write ("Great", 5) ;
}
```

Output : India is great.

Q 20. Explain the predefined stream objects cerr and clog. **(PTU, May 2007)**

Ans. cerr : If we want to display a particular message even when output of program has been redirected to a file, we can use cerr object. **e.g.** inserting a line

```
cerr <<ch ;
```

Each character ch will be sent to the screen by cerr, even though the output to cout is going to a file. The cerr object is normally used to display error messages, hence the name.

clog : This is another object. It is similar to cerr except that its output is buffered while cerrs is not.

Q 21. What are input/output stream in C++? **(PTU, May 2011 ; Dec. 2009, 2008)**

Ans. The C++ I/O subsystem is designed to work with a wide variety of devices that includes keyboard, monitor etc. The C++ I/O subsystem is designed in such a way that it provides a uniform

interface that is independent of the actual device being used. Thus, the programmer can write a C++ program that can receive input form, and send output to any device without bothering about the characteristics of device being used. This interface is called a stream.

A stream basically is a sequence of bytes. It can either act as a source from which the program can take input data or as a destination to which the program can send output data. The source stream that supplies data to the program is called input stream and the destination stream that receives data from the program is called output stream. These streams are called data streams.

Q 22. List the name of IOS format functions.

Ans. Width ( ) : To specify the required field size for displaying an output value.

Precision ( ) : To specify the number of digits to be displayed after the decimal point of a float value.

Fill ( ) : To specify a character that is used to fill the unused portion of a field.

Setf ( ) : To specify format flags that can control the form of output display.

Unsetf ( ) : To clear the flags specified.

Q 23. What is manipulator?

Ans. The output formats can be controlled using manipulators. The header file iomanip.h has a set of functions. Effect of these manipulator is the same as ios class member functions. Every ios member function has two formats. One is used for setting and second format is used to know the previous setting. But the manipulator does not return to the previous setting. The manipulator can be used with cout ( ) statement as given below :

cout << m1 << m2 << V1 ;

Here m1 and m2 are two manipulators and V1 is any valid C++ variable.

Q 24. What is the syntax for creating manipulator?

Ans. Syntax of a manipulator without any argument

```
ostream & name-of-manipulator (ostream & output)
    {
        .....
        ..... // body of manipulator
        .....
        return output ;
    }
```

Q 25. How manipulator is different from ios functions?

Ans. 1. ios function returns value while manipulators does not

2. We cannot create own ios functions while we can create our own manipulators.

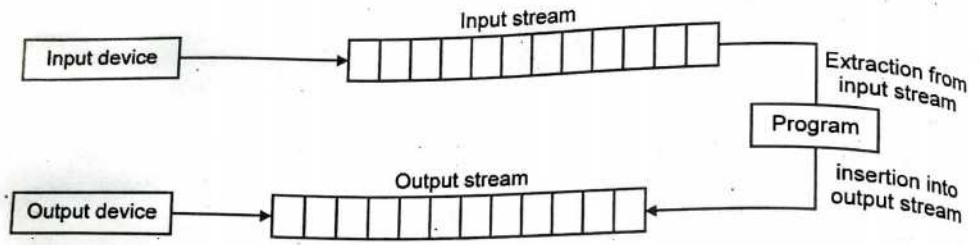
3. ios functions are single and not possible to be combined while manipulator are possible to be applied in chain.

4. ios function needs <iostream> while manipulator needs <iomanip>

5. ios function are member functions while manipulator are non-member functions.

Q 26. What is stream? Explain the features of C++ stream.

Ans. Stream : Stream is a sequence of bytes. It acts either us a source from which the input data can be obtained or as a destination to which the output data can be sent. The source stream that provide data to the program is called input stream and the destination stream that receives output from the program is called the output stream. In other words, a program extracts the bytes from an input stream and inserts bytes into an output stream as illustrates below :



Data stream

Input stream : Input streams are used to hold input from a data producer, such as keyboard, a file or a network. For example, the user may press key on the keyboard while the program is currently not expecting any input. Rather than ignore the users keypress, the data is put into an input stream, where it will wait until the program is ready for it.

Output stream are used to hold output for a particular data consumer, such as a monitor, a file, or a pointer. When writing data to an output device, the device may not be ready to accept that data yet. For example, the printer may still be warming up when the program write data to it's output stream. The data will sit in the output stream until the printer begins consuming it.

Advantages of I/O stream : Althrough input/output are implemented with stream for both C and C++, the C++ I/O stream classes provide the same facilitites for input and output as C stdio.h. The I/O stream classes in the standard C++ library have the following advantages:

1. The input (>>) operator and output (<<) operator are type safe. These operators are easier to use than scanf ( ) and printf ( ).
2. You can overload the input and output operators to define input and output for your own types and classes. This makes input and output across types, including your own, uniform.

Q 27. Explain standard stream in C++ with example.

Ans. A standard stream in a pre-connected stream provided to a computer program by it's environment. C++ comes with four predefined standard stream objects that have already been set up for your use.

1. Cin : An istream_withassign class tied to the stand and input (typically the keyboard).
2. Cout : An ostream_withassign class tied to the standard output (typically the monitor).
3. Cerr : An ostream_withassign class tied to the standard error (typically the monitor).
4. Clog : An ostream_withassign class tied to the standard error (typically the monitor), providing buffered output.

Unbuffered output is typically handled immediately, whereas buffered output is typically stored and written out as a block. Because clog isn't used very often, it is often omitted from the list of standard streams.

For example

```
#include <iostream.h>
int main ( )
{
    using namespace std ;
    cout <<"Enter your age :" << end l ;
    int nage ;
    cin >> nage ;
```



```
    if (nage < = 0)
    {
        cerr << "OOPS, you entered an invalid age!" << end l ;
        exit (1) ;
    }
        cout <<"you entered" <<nage << "year old" << end l ;
        return 0 ;
}
```

Q 28. Name the various I/O stream class. Also explain the facilities provided by them.

Ans. In C++ there are number of stream classes for defining various streams related with files and for doing input output operations. All these class are defined in the file <iostream.h> The various stream classes are as follows :

1. **ios** class is the topmost class in the stream classes hierarchy. It is the base class for istream, ostream and streambuf class.
2. istream, ostream serves the base classes for iostream class. The class istream is used for input and ostream for output.
3. Class ios is indirectly to iostream class using istream and ostream. To avoid the duplicity of data and member functions of ios class, it is declared as virtual base class when inheriting in istream and ostream as :

   Class istream : virtual public ios
   {
   } ;
   Class ostream : virtual public ios
   {
   } ;
4. The_withassign classes are provided with extra functionality for the assignment operations that's why – withassign classes.

Facilities provided by stream classes :

1. The ios class : It is responsible for providing all input and output facilities to all other stream classes as it is the top most class in the hierarchy of stream classes seeing. This class provides number of functions for efficient handling of formatted output for strings and numbers.

2. The istream class : This class is responsible for handling input stream. It provides number of functions for handling chars, strings and objects, record etc. besides inheriting the properites from ios class. The istream class provides the basic capability for sequential and random access input. An **istream** object has a **streambuf** derived object attached, and the two classes work together. The istream class does the formatting and the streambuf class does the low level buffered input. The class provides number of methods for input handling such as get, getline, read, peek, gcount, ignore, eatwhile, putback etc. This class also contains overloaded extraction operator >> for handling all data types such as int, signed int, char, long, double, float, long double. The extraction operator is also overloaded for handling stream buf and istream types of objects.

3. The istream_withassign class is a variant istream that allows object assignment. The predefined object cin is an object of this class and thus may be reassigned at run time to a different istream class.

4. ostream_withassign class is a variant of ostream that allow object assignment. The predefined object cout, cerr and clog are objects of this class and thus may be reassigned at run time to a different ostream object.

Q 29. Write a program to overload << and >> operator to the object of a class.

Ans.

```
#include <iostream.h>
class complex
{
    double real, imag ;
Public :
    complex ( )
    {
    }
complex (double r, double i)
    {
    real = r ;
    imag = i ;
}
Friend ostream & operator << (ostream &S, complex &C) ;
Friend istream & operator >> (istream &S, complex &C) ;
} ;
ostream & operator <<ostream &S, complex &C)
{
S << " ("<<C. real <<", "<<C.imag <<")" ;
return S ;
}
istream & operator >> (istream &S, complex &C) ;
{
S >> C.real >> C.imag ;
return S ;
}
void main ( )
{
complex C1 (1.5, 2.5), C2 (3.5, 4,5), C3 ;
cout << end/ << "C1 = "<<C1 << end / << "C2 =" << C2 ;
cout << end/ <<"Enter a complex number : ;
cin >> C3 ;
cout <<"C3 = " <<C3 ;
}
```

Q 30. Explain the concept of designing our own manipulators.

Ans. We can design our own manipulators for certain special purpose. The general form for creating a manipulator without any argument is :

```
ostream & manipulator (ostream & output)
{
    .....
    ..... (code)
    .....
    return output ;
}
```


Here, the manipulator is the name of the manipulator under creation. The following functions defines a manipulator called unit that displays 'inches' :

```
ostream & unit (ostream &output)
{
    output << "inches" ;
    return output ;
}
```

The statement

```
cout <<36<< unit ;
```

will produce the following output

36 inches.

We can also create manipulator that could represent a sequence of operation. Example

```
ostream & show (ostream & output)
{
    output.setf (ios : : show point) ;
    output.setf (ios : : show pos) ;
    output << setw (10) ;
    return output ;
}
```

This function defines a manipulators called show that turns on the flag **show point** and **showpos** declared in the class ios and sets the field width to 10.

Q 31. Write the syntax of write ( ) and read ( ) functions.

Ans. Syntax of write function :

of stream_obj.write ((char*) & var, size of (var));

Syntax of read function :

if stream_obj.read ((char*) & var, size of (var));

Q 32. How getline () function is different gets ()?

Ans. The purpose of getline function for inputting multiple strings except that it removes the delimiter character from the input stream. This function is similar to gets () function but in getline we can specify the terminating character also which is not available in gets function e.g.

cin.getline (str, max, '*') ; this statement on execution can input multiple lines until either enter the terminating character '*' or until user exceeds the size of the array.

Q 33. What is a structure? How is it different from class?

Ans. A data structure is a group of data elements grouped together under one name. These data elements, known as members, can have different types and different lengths. Data structures are declared in C++ using the following syntax :

```
struct structure_name
{
    member_type1 member_name 1;
    member_type2 member_name 2;
    .
    .
} object_names;
```

Where structure_name is a name for the structure type, object_name can be a set of valid identifiers for objects that have the type of this structure. Within braces { } there is a list with the data members, each one is specified with a type and a valid identifier as its name.

For example :

```
struct product
{
    int weight;
    float price;
};
product apple;
product banana, melon;
```

The only difference between a structure and a class in C++ is that, by default the members of class are private while, by default the members of structure are public.

Q 34. Differentiate between << & >> operators.

Ans. << is called insertion and >> is called extraction. These operators are use to write information to or read information from respectively, ostream and istream objects and to all classes derived from these classes. By default white space is skipped when the insertion and extraction operator are used. Also insertion operator (<<) points to the ostream object wherein the information is inserted. The extraction operator points to the object receiving the information obtained from the istream object.

Q 35. Write a short note on file.

Ans. A file is a collection of related information normally representing programs, both source and object forms and data. Data may be numeric, alphabetic or alphanumeric. A file can also be defined as a sequence of bits, bytes, lines or records whose meaning is defined by the programmes operations such as create, open, read and close are performed on files.

Q 36. Define Streams. What are various types of streams? **(PTU, May 2007)**

Ans. The C++ I/O subsystem is designed to work with a wide variety of devices that includes keyboard, monitor etc. The C++ I/O subsystem is designed in such a way that it provides a uniform interface that is independent of the actual device being used. A programmer can write a C++ program that can receive input from and send output to any device without bothering about the characteristics of device being used. This interface is called stream. Following are the types of streams.

1. Standard input stream : Represented by an object cin of istream class, is the stream from which input is received.

2. Output stream : Represented by an object cout of ostream class, is the stream to which output is sent.

3. Error stream : Represented by an object cerr of ostream, class is the stream to which error messages are sent.

Q 37. List various I/O stream flags and syntactic rule governing them.

(PTU, May 2009)

Ans. Various I/O stream flags are :

	Flag	Purpose
1.	skipws	skip whitespace on input.
2.	left	left justify output
3.	right	right justify output
4.	internal	use padding after sign or base indicator
5.	dec	decimal conversion
6.	oct	octal conversion
7.	showbase	hexadecimal covnersion
8.	showbase	use base indicator on output
9.	showpoint	use decimal point in floating point output
10.	uppercase	uppercase hex output
11.	showpos	preface positive integers with'f'
12.	unitbuf	flush all streams after insertion
13.	stdio	flush stdout and stderr after insertion

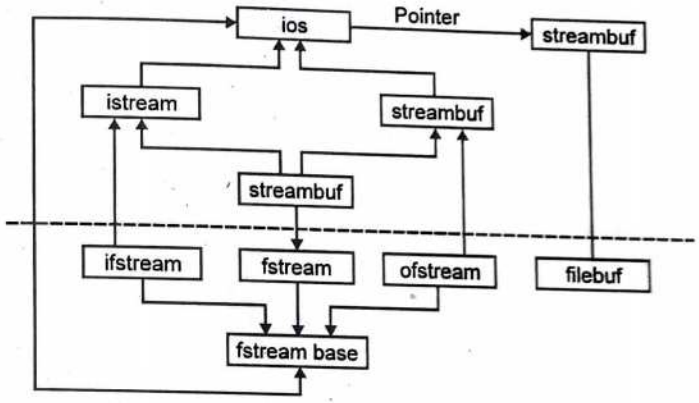
Rules : Following are the rules for governing them.

1. Reset the skipws flag when getting input from cin in REDIR because in normally skips over white space, which includes spaces, newlines and EOFs.

2. To redirect a program's input so that it comes from a file and not the keyboard, you can use <operator.

Q 38. Draw the hierarchical diagram of file stream classes.

Ans.



Hierarchy of file stream classes

Q 39. What do you mean by file positioning function? Discuss. **(PTU, Dec. 2007)**

Ans. Each file object has associated with it two integer values called the get pointer and the put pointer. These are also called the current get position and the current put position. These values specify the byte number in the file where writing or reading will take place.

Sometimes you want to start reading an existing file at the beginning and continue until the end. When writing, you may want to start at the begining, deleting and existing contents or at the end. For all these purposes file positioning functions work. Sometimes you take control of the file pointers so that you can read from and write to an arbitrary location in the file. Following are the file positioning functions :

1. Seekg ( ) and tellg ( ) : These functions allow you to set and examine the get pointer. The seekg ( ) function set it to the beginning of the file so that reading would start there. This form of seekg ( ) takes one argument, which represents the absolute position in the file. And the tellg ( ) function returns the current position to the get pointer. The program uses this function to return the pointer position at the end of the file. This is the length of the file in bytes.

2. Seekp ( ) and tellp ( ) : These functions perform these same actions on the put pointer.

Q 40. Explain the predefined stream objects cerr and clog. **(PTU, May 2007)**

Ans. cerr : If we want to display a particular message even when output of program has been redirected to a file, we can use cerr object. **e.g.** inserting a line

```
cerr <<ch ;
```

Each character ch will be sent to the screen by cerr, even though the output to cout is going to a file. The cerr object is normally used to display error messages, hence the name.

clog : This is another object. It is similar to cerr except that its output is buffered while cerrs is not.

Q 41. What do you mean by file pointer?

Ans. Each file has two associated pointers known as the file pointers. One of them is called the input pointer and the other is called the output pointer. We can use these pointers to move through the files while reading or writing. The input pointer is used for reading the contents of a given file location and the output pointer is used for writing to a given file location. Each time an input or output operation takes place, the appropriate pointer is automatically advanced.

Q 42. Explain the syntactic rules for the following random access member functions :

(i) seekg

(ii) seekp

(iii) tellg

(iv) tellp **(PTU, May 2005)**

Ans. (i) seekg : Its syntax is

```
ifstream & seekg (long offset, seek_dir origin = ios :: beg) ;
```

Where offset specifies number of bytes by which file pointer to be moved ; origin specifies the reference point from where the offset is to measured, default argument with default value beginning of the file as reference point.

(ii) seekp : It syntax is

```
ofstream & seekp (long offset, seek_dir origin = ios :: beg) ;
```

where offset, origin, default argument are same as seekg.

(iii) tellg :

```
long tellg ( ) ;
```

It return the current position as number of bytes from the beginning of the respective files.

(iv) tellp :

```
long tellp ( );
```

It also returns the current position as number of bytes from the beginning of the respective files.

Q 43. What do you mean by closing a file?

Ans. When reading and writing or consulting operations on a file are complete we must close it so that it becomes available again. In order to do that we shall call the member functions close ( ), that is in change of flusting the buffers and closing the file. Its form is quite simple :

```
void close ( ) ;
For example :
#include <iostream.h>
using namespace std ;
int main ( ) {
of stream outfile ;
outfile.open ("test.txt", ofstream : : out 1 ofstream : : app) ;
outfile << This sentence is appended to the file content \n" ;
outfile.close ( );
return 0 ;
}
```

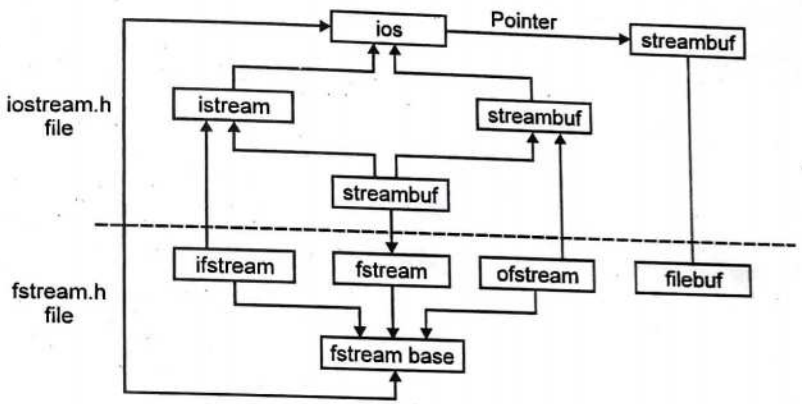

Q 44. Explain the hierarchy of file system classes.

An. The I/O system of C++ contains a set of classes the define the file handing method.s These including the following three classes.

1. ifstream – for handling input files.
2. ofstream – for handling output files.
3. fstream – for handling files on which both input and output can be performed.

These classes are derived from fstream base and from those declared in the header files iostream.h as shown in figure.

The classes ifstream, ofstream and fstream are designed exclusively to manage the disk files, and their declarations exist in the header file fstream.h.



Hierarchy of file stream classes

1. Fileup : Its purpose is to set the file buffers to read and write. Contains openprot constant used in the open ( ) of the stream classes. Also contains close ( ) and open ( ) as members.

2. Fstreambase : Provides operations common to the file streams. Serves as a base for stream, ifstream and ofstream classes. Contains open ( ) and close ( ) functions.

3. Ifstream : Provides input operations. Contains open ( ) with default input mode. inherits the functions get ( ) ; get line ( ), read ( ), seeks ( ) and tellg ( ) functions from istream.

4. Ofstream : Provides output operation. Contains open ( ) with default output mode. Inherits put ( ), seekp ( ), tellp (), and write ( ), function from ostream.

5. fstream : Provides support for simultaneous input and output operations. Contains open ( ) with default input mode. Inherits all the functions from istream and astream classes through iostream.

Q 45. What are the different modes in which files can be opened? Explain with example.
(PTU, May 2012, 2011, 2009 ; Dec. 2009)

OR

Describe different file opening modes in C++. **(PTU, Dec. 2011, 2008 ; May 2019, 2008)**

OR

What are the two methods of opening a file? Explain with examples. In which case, the two methods are used with the advantages. **(PTU, May 2006)**

Ans. A file can be opened in two ways :

1. Opening a file using constructor : Constructor is used to initialize an object while it is being

created. Here, filename is used as an argument for the constructor, which is then used to initialized the file stream object. The prototype of ifstream class is

```
ifstream (const char *path, int mode = ios : : in,
    int prot = file buf : : open prot) ;
```

and for ofstream class is

```
ofstream (const char * path, int mode = ios : : out,
    int prot = file buf : : open prot) ;
```

where path is filename with path information, mode is file opening mod, default argument with default value 'in' for input mode for ifstream class and cout for output mode, prot is access specifier

e.g.

```
#include <fstream.h>
void main ( )
{
    ofstream outfile ("ABC.TXT") ;
    outfile <<"This is object.oriented programming" ;
    outfile <<"This is written in file" ;
}
```

Advantages : In this type initialization sets aside various resources for the file and accesses the file of that name on the disk. When the program terminates, the outfile object goes out of scope. This calls destructor, which closes a file, so we don't need to close the file explicity.

2. Opening a file using open ( ) member function : The prototype of open ( ) functions is

```
void open const char * path, int mode,
    int prot = filebuf : : open prot) ;
```

and open ( ) function is invoked on a file stream object as

```
file-stream-object.open ("filename", mode) ;
```

e.g.

```
ifstream infile ;
infile.open ("ABC.dat" , ios : : in) ;
```

open a file ABC.dat in text mode for input only with read and write permissions.

Advantages : The open ( ) function allows several mode bits to specify file object that we are opening. In and out are used to perform input and output on the file. The vertical bars between the flags cause the bits representing these flags to be logically ORed together, so that several flags can apply simultaneously.

Q 46. Explain the error handling during file operations.

Ans. So far we have been opening and using the files for reading and writing on the assumption that everything is fine with the files. This may not be true always. For instance, one of the following things may happen when dealing with the files :

1. A file which we are attempting to open for reading doe not exist.
2. The file name used for a new file may already exist.
3. We may attempt or invalid operation such as reading past the end-of-file.
4. There may not be any space in the disk for storing more data.
5. We may use an invalid file name.
6. We may attempt to perform an operation when the file is not opened for that purpose.

The C++ file stream inherits a 'stream-state' member from the class ios. This member records information on the status of a file that is being currently used. The stream state member uses bit fields to store the status of the error conditions stated above.

The class ios supports several member functions that can be used to read the status recorded in a file stream. These functions are given below.

(i) eof ( ) : Returns true (non-zero value) if end-of-file is encountered while reading, otherwise return false (zero).

(ii) fail ( ) : Return true when an input as output operation has failed.

(iii) bad ( ) : Returns true if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is false it may be possible to recover from any other error reported and continue operation.

(iv) good ( ) : Returns true if error has occured. This mean, all the above functions are false. For instance, if file.good ( ) is true, all is well with the stream file and we can proceed to perform I/O operations. When it return false, no further operations can be carried out.

The functions may be used in the appropriate places in a program to locate the status of a file stream and there by to take the necessary corrective measures.

Q 47. Write a program that displays a text file. **(PTU, May 2009)**

Ans.

```
#include <iostream.h>
#include <fstream.h>
void main ()
{
    char ch, filename [12] ;
    cout <<"Enter name of file to read" ;
    cin >> file name ;
    ifstream infile (file name) ;
    if (in file. failed () ) {
    cout <<"unable to open file" << file name ;
    return ;
}
cout <<"contents of file :" <<file name ;
while (! in file. eof ( )) {
    infile.get (ch) ;
    cout << ch ;
    }
    infile. close ( ) ;
}
while (in file)
{
    infile.get (ch) ;
    cout << ch;
    ++ cout ;
}
    cout << "length of file" << cout ;
}
```

Q 48. Write a program in C++ to read a file and to display the contents of file on screen with line numbers. **(PTU, May 2009)**

Ans.

```
#include <iostream.h>
#include <fstream.h>
void main ( )
```

```
{
    char filename [12], strname [80] ;
    int i = 1 ;
    cout <<"Enter name of file to read" ;
    cin >> file name ;
    if stream infile (file name) ;
    if (infile. fail ( ))
{
    cout <<"unable to open file" << file name ;
    return ;
}
    cout <<"contents of file are" ;
    while (! infile. eof ( ) )
{
    infile.getline (str name, 80) ;
    cout <<"line" << i ;
    cout. write (str name, strlen (str name) ) ;
    i ++ ;
    cout <<end l ;
}
    infile. close ( ) ;
}
```

Q 49. Write a C++ program to copy the contents of one file to another file.
(PTU, Dec. 2009, 2008, 2005 ; May 2011, 2008, 2005)

Ans.
```
#include <iostream.h>
# include <fstream.h>
{
    char ch ;
    ifstream infile ("ABC. Txt") ;
    ofstream outfile ("BCD.Txt") ;
    while (infile)
    {
        infile.get (ch) ;
        outfile. put (ch) ;
    }
}
```

Q 50. Distinguish between binary and text files. **(PTU, May 2010)**

Ans. Binary files and text files both are used to storage and subsequent retrieval of data. These files are used to correct the incorrect data and to update the correct form of data. Binary files are : sequential access files and text files are random access files. In binary files, the data or text can be stored or read back sequentially. In text file, data can be accessed and processed randomly. Binary files are used to store program files. And text files are used as data files.

Q 51. What the text and binary modes have to do with files?

Ans. To use text mode of file we do not include the ios : : binary flag in their opening mode. These files are designed to store text and thus all values that we input or output from/to them can

suffer some formatting transformations, which do not necessarily correspond to their literal binary value. Whenever you open a file in textual mode and start to read it, the operating system converts each carriage-return and line-feed character pairs to a single line feed character. Whenever you write to a file opened in textual mode all line-feed characters produce two characters in the output file : a carriage-return and a line-feed character.

Binary files are simple. Bytes follow each other and when we read or write such a file the operating system reads or writes the bytes as they are without any conversion. Whenever you want to go to a certain position of a file using the command seek, you can without problem.

Q 52. What is the mode 'rb+' in file handling?

Ans. rb+ used for read write from binary file.

Q 53. Explain the error handling during file operations in C++.

Ans. Eof ( ) – used to check the end of file character. **(PTU, Dec. 2010)**
fail ( ) – used to check the status of file at opening for I/O.
bad ( ) – used to check for invalid file operations or unrecoverable error.
good ( ) – used to check whether the previous file operation has been successful.

Q 54. How random access is done for files?

Ans. To access a random file the following functions are used :
(a) seekp – it allows us to set the put pointer.
(b) ssekg – it allows us to set the get pointer.
(c) tellg – to enquire the current position of get pointer.
(d) tellp – it tells the current position of put pointer.

Q 55. Explain the difference between the mode "r" and the mode "r+".

Ans. The mode "r" opens the files for reading only, whereas the mode "r+" open the file fo reading and writing.

Q 56. Explain the difference between the mode "a" and mode "a+".

Ans. The mode "a" open the file for writing only (at the end of the files), whereas the mode "a+ opens the file for reading and writing (again at the end of the file).

Q 57. Write main () program that include everything necessary to call the functions give below :

Given function :

```
Int timer 2 (inta)
{
    return (a*2);
}
```

Ans.
```
main ()
{
    int times 2 (int); llPrototype
    int alpha = times 2 (37); ll Function call
}
```

Q 58. Write a function called () that has two integer arguments. The function will retu the value of the sum of these two arguments.

Ans.
```
add (x, y)
int x, y;
{
```

```
    return (x + y);
}
```

Q 59. Describe different file opening modes in C++.

Ans. We create the file in one statement and open it in another, using the open () function, which is a member of the fstream class.

In the open () function we include several mode bits to specify certain aspects of the file object we are opening. Following are the modes for opening file in C++ :

	Mode	Meaning
1.	in	open for reading (default for ifstream)
2.	out	open for writing (default for ofstream)
3.	app	start reading or writing at end of file (APPend)
4.	ate	erase file before reading or writing (trunc ATE)
5.	no create	error when opening if file does not already exists
6.	no replace	error when opening for output if file already exists, unless ate or app is set
7.	binary	open file in binary (not text) mode.

Q 60. Write a program to count non-vowels in a file of test.

Ans.

```
/*PRG TO COUNT NO OF NON VOWELS IN FILE .....*/
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
#include <stdio.h>
void main ()
{
   clrscr () ;
   int cnt = 0;
   char fname[15], line [80] ;
   cout <<"Enter file name :";
   cin >>fname ;
   ifstream fin;
   fin.open (fname, ios : : in);
   while (fin)
   {
      fin.getline (line,80);
      puts (line);
      for (int i=0; line[i]! = '0'!i++)
      if (line[i]!='a'&&line[i]!='e'&&line[i]!='i'&&line[i]!='o'&&line[i]!='u'&&line[i]!
      ='A'&&line[i]!='E'&&line[i]!='I'&&line[i]!='O'&&line[i]!='U')
         cnt++;
   }
      fin.close ( );
}
```


Q 61. Write a program to copy the content of one file to another in the text mode.

Ans.

```
/* TO COPY ONE FILE TO ANOTHER.....*/
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<stdio.h>
void main ()
{
    clrscr () ;
    char fname [15], line [80];
    cout <<"Enter file to copy : " ;
    cin>>fname ;
    ifstream fin;
    ofstream fout;
    fin.open(fname, ios : : in); //OPENING FILE IN READ ONLY MODE
    fout.open ("temp", ios : : app) ;
    while (fin)
    {
        fin. getline (line,80) ; //READING FILE LINE BY LINE .....
        fout <<line<< "\n" ;
    }
    fin.close ( ) ;
    fout. close () ;
    fin. open ("temp", ios : : in) ; //OPENING FILE IN READ ONLY MODE
    while (fin)
    {
        fin.getline (line, 80); //READING FILE LINE BY LINE .....
    }
    fin.close ( );
    getch ( );
}
```

Q 62. What are the various types of files? How are they handled in C++?

Ans. In the real world, computer applications need to store large amounts of data for extended period of time. To accomplish the task of storing large amounts of data, data files are used. There are two types of data files :

1. Sequential access files : These files must be accessed in the same order in which they were written. This process is analogous to audio cassette tapes where you must fast forward or rewind through the songs sequentially to get to a specific song. In order to access data from a sequential file, you must start at the beginning of the file and search through the entire file for the data that you want.

2. Random access files : These files are analogous to audio compact disks where you can easily access any song regardless of the order in which the songs were recorded. Random access files allow instant access any data in the file. Unfortunately, random access files often occupy more disk space than sequential access files.

File handling methods : The I/O system of C++ contains a set of classes that define the file

handling methods. These include ifstream, ofstream, and fstream. These classes are derived from fstreambase and from the corresponding iostream class. To open the files following methods are used.

1. Opening a file using constructor : To open the file create a file stream object to manage the stream using the appropriate class that is the ofstream is used to create the output stream and the class istream to create the input stream. After creating object initialise the file object with the desired filename.

For example the following statement opens a file named "results" for output.

```
Ofstream outfile ("results") ;
```

This creates outfile as an ofstream object that manages the output stream, similarly the following statements declares infile as an ifstream object and attaches it to the file "data" for reading

```
ifstream infile ("data") ;
```

The connection with file is closed automatically when the stream object expires.

2. Opening file using open ( ) : The function open can be used to open multiple file that use the same stream object. To open a file, first we create a single stream object as follows

```
File-stream-class stream-object ;
Stream-object.open ("filename") ;
```

For example

```
Ofstream outfile ;
Outfile.open ("DATA1") ;
.....
.....
outfile.close () ;
```

Q 63. What is the use of command line arguments? **(PTU, May 2012)**

Ans. Command line arguments are used to pass the name of a data file to an application. To read command line argument the main ( ) function must itself be given two arguments. The first, argc represent the total number of comand line arguments. The system stores the command line argument as strings in memory and creates an array of pointers to these strings.

```
#include <iostream.h>
#include <fstream.h>
void main (int argc, char*argv [ ])
{
ifstream infile ;
ofstream outfile ;
char ch ;
infile.open (argv [1]) ;
outfile.open (argv [2]) ;
if (in file)
{
cout <<"can't open" <<argv [1] ;
exit (0) ;
}
while (infile.get (ch)! = 0)
{
outfile.put (ch) ;
}
}
```


Q 64. What are various types of files? What are the various modes in which a file can be opened? Explain by giving examples. **(PTU, May 2012)**

Ans. In the real world, computer application used to store large amounts of data for extended period of time. To accomplish the task of storing large amount of data, data files are used. There are two types of data files :

1. Sequential access files : These files must be accessed in the same order in which they were written. This process is analogous to audio cassette tapes where you must fast forward or rewind through the songs sequentially to get to a specific song. In order to access data from a sequential file, you must start at the beginning of the file and search through the entire file for the data that you want.

2. Random access file : These files are analogous to audio compact disks where you can easily access any song regardless of the order in which the songs were recorded. Random access files allow instant access any data in file. Unfortunately, random access files often occupy more disk space than sequential access files.

File Opening Mode

Mode	Meaning
1. in	open for reading (default for reading)
2. out	open for writing (default for ofstream)
3. app	start reading or writing at end of file (APPend)
4. ate	erase file before reading or writing (trunc-ATE)
5. no create	error when opening if file does not already exist
6. no replace	error when oepning for output if file already exists, unless ate or app is set
7. binary	open file in binary (not text) mode

Q 65. Explain the various methods of reading a text file. **(PTU, May 2012)**

Ans. Following are the methods for reading a text file.

1. Reading character data : Data can be read using character input function get ( ) of ifstream class.

e.g.

```
#include <iostream.h>
#include <fstream.h>
void main ( )
{
char ch ;
char filename [12] ;
cout <<"Enter name of file to read" ;
cin >> filename ;
ifstream infile (file name) ;
if (infile.fail ( ))
{
cout <<"unable to open file" ;
return ;
}
cout <<"contents of file" ;
while (! infile.of (7)) ;
infile.get (ch) ;
cout <<ch ;
}
```

```
infile.close ( ) }
}
```

2. Reading string data : String data can be read using string input function getline ( ) of ifstream class.

e.g.

```
#include <iostream.h>
#include <fstream.h>
void main ( )
{
char filename [12], strname [80] ;
cout <<"Enter name of file to read" ;
cin >> filename ;
ifstream infile (file name) ;
if (infile.fail ( ))
{
cout <<"unable to open file" << file name ;
return ;
}
cout <<"contents of file" <<file name ;
while (! infile.eof ( ))
}
infile.getline (strname, 80);
cout.write (strname, strlen (strname)) ;
}
infile.close ( ) ;
}
```

Q 66. Write various I/O operations performed on files.

Ans. The file stream classes support a number of member functions for performing the input and output operations on files.

1. Put ( ) and get ( ) functions : The function put ( ) writes a single character to the associated stream. Similarly, the function get ( ) reads a single character from the associated stream.

Program to demonstrate I/O operators on characters

```
#include <iostream.h>
#include <fstream.h>.
#include <string.h>
int main ( )
{
char string [80];
cout <<"Enter a string";
Cin >> string;
int len = strlen (string);
stream file;
Cout <<"\n opening the 'Text' file and storing the string in if;
file.open ("Text", ios : : in/ios : : out);
for (int i = 0, i < len; i + +)
file.put (string [i]);
file.seek g (0);
char ch;
```



```
cout <<"Reading the file contents";
while (file)
{
file.get (ch);
cout <<ch;
}
return 0'
```

2. Write ( ) and read () functions : The functions write ( ) and read ( ) unlike the functions put ( ) and get ( ) handle the data in binary form. This means that the values are stored in the disk file in the same format in which they are stored in the internal memory.

Program to demonstrate I/O operations on binary files

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
const char * file name = "BINARY";
int main ( )
{
float height [4] = {175.5, 153.0, 167.25, 160.70};
ofstream outfile;
outfile.open (file name);
outfile.write ((char x) & height, size of (height));
outfile.close ( );
for (int i = 0; i < 4; i + +)
height [i] = 0;
ifstream infile;
infile.open (filename);
infile.read ((char x) & height, size of (height));
for (i = 0; i < 4; i + +)
{
cout.set f (ios : : showpoint);
cout << set w (10) <<set precision (2) <<height [i]};
}
infile. close ( );
return 0;
}.
```

3. Reading a writing or class object : The binary input and output functions read ( ) and write ( ) are designed to handle the entire structure of an object as a single unit, using the computer's internal representation of data. For instance, the function write ( ) copies a class object from memory bytes by byte with no conversion.

Q 67. How is the exception handling performed in C++ ? Write a program that throws an arithmetic exception as and when a number input is greater than 9999. **(PTU, Dec. 2017)**

Ans. Exception handling performed in C++ : *Refer to Q.No. 16*

Program :

```
#include <iostream.h>
#include <string>
using namespace std;
int main ( )
{
```

```
int num;
string str_bad = "wrong number used";
cout <<"input number.
Cin>> num;
try.
{
If (num = = 1)
{
throw 5;
}
If (num = = 2)
{
throw ;
}
If (num > 9999)
{
throw str_bad
}
}
}
catch (int a)
{cout <<"An exception occurred!" <<endl;
cout <<"Exception number is" <<a<<endl;
}
Catch (float b)
{
cout <<"An exception occured">> endl;
cout <<"exception number is"<<b>> endl;
}
Catch (...)
{
Cout <<"A default exception occurred "<<endl;
Cout <<"why?"<< str_bad <<endl;
}
return 0;
}
```

Q 68. What are the advantages of using Template functions? How are they different from the macros? **(PTU, Dec. 2014)**

Ans. Advantages of using Function Templates :

- Avoids writing the identical function again and again for the different data types.
- Reduce the storage space.
- Easy to debug, because only the small portion of the code (template) is debugged instead more number of functions.

Differences between templates and macros :

- Macros are not type safe; that is, a macro defined for integer operations cannot accept float data. They are expanded with no type checking.

- It is difficult to find error in macros.
- In case a variable is post-incremented or decremented, the operation is carried out twice.

Q 69. What is the use of Templates in C++? Explain the use of Standard template library with the help of a C++ Program. Also Explain how exceptions are caught in C++. **(PTU, Dec. 2014)**

OR

What are templates? Explain the need of templates. **(PTU, May 2019)**

Ans. Templates : Template is one of the features added to C++ recently. It is new concept which enable us to define generic classes and functions and thus provides support for generic programming. A template can be used to create a family of classes or functions. For example, a class template for an array class would enable us to create arrays of various data types such as int array and float array. Similarly, we can define a template for a function, say null (), that would help us to create various versions of null ( ) for multiplexing int, float and double type values.

Need for Templates : Consider the following function written for adding two integers.

```
Void add (int a, int b)
{
    int sum ;
    sum = a + b;
}
```

Addition of numbers of multiple data types may be required in any real program. When different data types like float or double are required to be added, one has to change the data types in the programs. A programmer usually rewrites the functions at the desired locations with appropriate data types using cut and paste. This cause unnecessary hassles and sometimes compile time errors. Another way of achieving addition of different data types is provided by function overloading. Function overloading requires coding of as many functions as there are combinations of data types. These hassless can be avoided by using function templates.

Program :

```
// C++ function template example, the base of class template
#include<iostream>
using namespace std;
// function template declaration and definition
template<class any_data_type>
any_data_type MyMax(any_data_type var1, any_data_type var2)
{
return var1>var2 ? var1:var2;
}
int main(void)
{
cout<<"MyMax(10,20)="<<MyMax(10,20)<<endl;
cout<<"MyMax('Z','p')="<<MyMax('z','p')<<endl;
cout<<"MyMax(1.234, 2.345)="<<MyMax(1.234, 2.345)<<endl;
//some logical error here ?
cout<<"\n Logical error, comparing pointers instead of string....."<<endl;
char*p="Function";
char*q="Template";
cout<<"Address of *p="<<&p<<endl;
```

```
cout<<"Address of *q="<<&q<<endl;
cout<<"MyMax(\"Function\",\"Template\")="<<MyMax(p,q)<<endl;
cout<<"Should use specialization, shown later....."<<endl;
return 0;
}
```

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords : try, catch and throw.

throw : A program throws an exception when a problem shows up. This is done using a throw keyword.

catch : A program catches an exception with an exception handler at the place in a program where you want to handle problem. The catch keyword indicates the catching of an exception.

try : A try block identifies a block of code for which particular exception will be activated. It's followed by one or more catch blocks.

The catch block following the try block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parenthesis following the keyword catch.

```
try
{
// protected code
}
catch (ExceptionName e)
{
// Code to handle ExceptionName exception.
}
```

Q 70. Write a function template to find maximum of three numbers. (PTU, May 2015)

Ans. Function template to find maximum of three numbers.

```
# include <iostream>
# include <cstring>
# include <string>
template <typename T>
inline T const &max (T const & a, T const & b)
{
return a <b ? b : a ;
}
inline char const* max (char const * a, char const *b)
{ return std : : strcmp (a,b) <0 ? b:a;
}
template <typename T>
inline T const &max (T const&a, T const&b, Tconst&c).
{
return max (max (a,b),c);
}
int main ( )
{
```



```
std : : cout << : : max (7, 42, 68);
const char *s1 = "frederic";
const char *s2 = "anica";
const char *s3 = "lucas";
}
```

Q 71. What is an exception ? Why it is necessary to handle an exception by special code? Develop a program containing possible exception and use a try block to throw it and a catch block to handle it properly. (PTU, May 2015)

Ans. Exception : An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumtance that arise while a program is running such as an attempt to divide by zero. Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords : try catch and throw. In general, an exception is handled by saving the current state of execution in a predefined place and switching the execution to a specific subroutine known as an exception handler. If exceptions are continuable, the handler may later resume the execution at the original location using the saved information.

Exception handling machanism : *Refer to Q.No. 13*

Program : The following is an example which throws a division by zero exception and we catch it in catch block.

```
#inlcude <iostream>
using namespace std;
double division (int a, int b)
{
If (b = = 0)
{
throw "Division by zero condition;"
}
return (a/b);
}
int main ( )
{
int x = 50;
int y = 0;
double Z = 0;
try {
Z = division (x,y);
cout <<z <<endl;
} catch (const char *msg) {
cerr << msg << endl;
}
return 0;
}
```

Because we are raising an exception of type const char* so while catching this exception, we have to use const char * in catch block. If we compile and run above code, this would produce the following result:

Division by Zero conditions !

Q 72. How you can open and close a file ? **(PTU, Dec. 2015)**

Ans. You can use the open function to open a file, the close function to close a file.

Q 73. Differentiate between function template and class template. **(PTU, Dec. 2015)**

Ans. Difference between class and function templates : Remember that for function template, the compiler can infer the template arguments :

```
int i = max (4,5);
int j = max <int> (7,2); //OK, but not needed.
```

For class templates, you always have to specify the actual template arguments; the compiler does not infer the template arguments :

```
List primes; //Error
primes.append (2);
```

Q 74. What are templates ? Write their syntax and usage. Design a function template in C++ to sort an array. **(PTU, Dec. 2015)**

Ans. Templates : Template is one of the features added to C++ recently. It is new concept which enable us to define generic classes and functions and thus provides support for generic programming. A template can be used to create a family of classes or functions. For example, a class template for an array class would enable us to create arrays of various data types such as int array and float array. Similarly, we can define a template for a function, say null (), that would help us to create various versions of null ( ) for multiplexing int, float and double type values.

Syntax for template declarations is :

```
>> – + ..... + ..... – >
'–export
> .....template .... <.....template_parameter_list.....>.....
declaration ..... ><
#include<iostream>
#include<cstdlib>
using namespace std;
template<classT> void quick sort (T a[ ], const int & leftarg, const int & rightarg)
{
if(leftarg <rightarg){
T pivotvalue = a[leftarg];
int left = leftarg – 1;
int right = rightarg + 1;
for (ii){
while (a [--right] > pivotvalue);
while (a [++left] < pivotvalue);
if(left > = right)break;
T temp = a[right];
a[right] = a [left];
a[left] = temp;
}
int pivot = right;
quick sort (a, leftarg, pivot);
quick sort (a, pivot +1, rightarg);
}
}
```



```
int main (void){
int sortone [10];
for (int i = 0; i<10; i++){
sortme [i] = rand ( );
cout <<sort one [i] <<" ";
};
cout <<endl;
quick sort <int> (sortone, 0, 10–1);
for (int i = 0; i < 10; i ++) cout <<sortone [i] <<n
";
cout <<endl;
return 0;
}
```

Q 75. Write a program to copy the content of a data file to another file. Make use of the exception handling conditions also. **(PTU, May 2016)**

Ans.

```
#include <iostream.h>
#include <fstream.h>
{
try
{
char ch;
ifstream infile ("ABC. Txt");
ofstream outfile ("BCD.Txt");
While (infile)
{
infile.get(ch);
outfile.put(ch);
}
}
Catch (exception e)
{
cout<<"Not Copied"
}
}
```

Q 76. Name various standard classes of C++. **(PTU, May 2017)**

Ans. The C + + standard library can be categorized into two parts. **The standard function library.** This library consists of general purpose, stand–alone functions that are not part of any class The function library is inherited from C. **The object orient class library.** This is a collection o classes and associated functions.

Q 77. Write a function template to find the minimum of three numbers. **(PTU, May 2017**

Ans. Function template to find minimum of three members :

```
#include <iostream>
#include <string>
#include <string>
template <typename T>
inline T const & max (T const & a, T const & b)
```

```
{
return a > b ? b : a.
}
inline  char const * min (char Const * a, char Const * b)
{
return std : : strcmp (a, b) > 0 ? b : a;
}
template <typename T>
inline T const & max (T const & a, T const & b, T const & c).
{
return min (min (a, b), c);
}
int main ( )
{
std : : cont < < : : min (7, 4, 2, 6, 8)
const char *S1 = "frederic";
const char *S2 = "ariea";
const char*S3 = "lucas";
}
```

❑❑❑

LORDS MODEL TEST PAPERS

(Unsolved)

LORDS MODEL TEST PAPER – 1

Instructions to candidates :

1. Section A is compulsory.
2. Attempt any four questions from section B.
3. Select any two questions from section C.

SECTION – A

Q 1. (a) What is data abstraction ? Discuss with an example.
(b) What is control statement ?
(c) What is friend function ?
(d) What are the uses of 'Thus' pointer ?
(e) What do you mean by visibility mode ?
(f) What are virtual constructors ? Give relevant examples to explain if.
(g) What do you mean by pure virtual function ?
(h) What is operator overloading ?
(i) What is the difference between get ( ) and put ( ) function.
(j) What do you mean by file pointer.

SECTION – B

Q 2. What are the various data type supported by Turbo C++ ? Give memory requirement of each type.
Q 3. What are the various types of constructors used with the classes ? Explain with examples.
Q 4. Differentiate between static and runtime polymorphism giving example.
Q 5. Discuss the exception handling features of C++ ?
Q 6. Write various I/O operations performed on files.

SECTION – C

Q 7. Explain in detail different operator in C++. Also give examples.
Q 8. What is the function overloading ? Explain the concert of constructor overloading. Can a template function be overloaded ?
Q 9. What are the various type of inheritance in C++ ? Give an example of each.

LORDS MODEL TEST PAPER – 2

Instructions to candidates :

1. Section A is compulsory.
2. Attempt any four questions from section B.
3. Select any two questions from section C.

SECTION – A

Q 1. (a) What do you mean by generic programming ?
(b) What is the purpose of Cin and Cout statement ?
(c) What are the characteristics of a constructor ?
(d) What is 'This' pointer ? Explain with example.
(e) What are virtual constructors.
(f) What is the difference between static and dynamic binding ?
(g) What are dangling pointer ? Give example.
(h) What are input/output stream in C++ ?
(i) Differentiate between << & >> operators.
(j) What is type casting ?

SECTION – B

Q 2. What is polymorphism ? Give difference between function overloading and overriding with example.
Q 3. Distinguish between single and multiple inheritance.
Q 4. Explain the exception handling mechanism with example.
Q 5. What is the use of command line arguments ?
Q 6. What is virtual function ? How we can declare the virtual function ? Also explain the need for virtual function.

SECTION – C

Q 7. Discuss the features of an object oriented programming in detail.
Q 8. What is operator overloading ? Explain binary operator overloading.
Q 9. Describe different file opening modes in C++.

❑❑❑