Code: AC11 **Subject: OBJECT ORIENTED PROGRAMMING**
PART - I, VOL – I

TYPICAL QUESTIONS & ANSWERS

OBJECTIVE TYPE QUESTIONS

Each Question carries 2 marks.

Choose correct or the best alternative in the following:

Q.1 The address of a variable temp of type float is

(A) *temp **(B)** &temp
(C) float& temp **(D)** float temp&

Ans: B

Q.2 What is the output of the following code

```
char symbol[3]={'a','b','c'};
for (int index=0; index<3; index++)
cout << symbol [index];
```

(A) a b c **(B)** "abc"
(C) abc **(D)** 'abc'

Ans: C

Q.3 The process of building new classes from existing one is called _____.

(A) Polymorphism **(B)** Structure
(C) Inheritance **(D)** Cascading

Ans: C

Q.4 If a class C is derived from class B, which is derived from class A, all through public inheritance, then a class C member function can access

(A) protected and public data only in C and B.
(B) protected and public data only in C.
(C) private data in A and B.
(D) protected data in A and B.

Ans: D

Q.5 If the variable count exceeds 100, a single statement that prints "Too many" is

(A) if (count<100) cout << "Too many";
(B) if (count>100) cout >> "Too many";
(C) if (count>100) cout << "Too many";
(D) None of these.

Ans: C

Q.6 Usually a pure virtual function
- **(A)** has complete function body.
- **(B)** will never be called.
- **(C)** will be called only to delete an object.
- **(D)** is defined only in derived class.

Ans: D

Q.7 To perform stream I/O with disk files in C++, you should
- **(A)** open and close files as in procedural languages.
- **(B)** use classes derived from ios.
- **(C)** use C language library functions to read and write data.
- **(D)** include the IOSTREAM.H header file.

Ans: B

Q.8 Overloading the function operator
- **(A)** requires a class with an overloaded operator.
- **(B)** requires a class with an overloaded [ ] operator.
- **(C)** allows you to create objects that act syntactically like functions.
- **(D)** usually make use of a constructor that takes arguments.

Ans: A

Q.9 In C++, the range of signed integer type variable is _____

(A) 0 to 2^{16} **(B)** -2^{15} to $2^{15}-1$

(C) -2^{7} to $2^{7}-1$ **(D)** 0 to 2^{8}

Ans: B

Q.10 If $x = 5, y = 2$ then $x \wedge y$ equals_____.

(where $\wedge$ is a bitwise XOR operator)

(A) 00000111 **(B)** 10000010

(C) 10100000 **(D)** 11001000

Ans: A

Q.11 If an array is declared as

int a[4] = {3, 0, 1, 2}, then values assigned to a[0] & a[4] will be _____

(A) 3, 2 **(B)** 0, 2

(C) 3, 0 **(D)** 0, 4

Ans: C

Q.12 Mechanism of deriving a class from another derived class is known as____

(A) Polymorphism **(B)** Single Inheritance
(C) Multilevel Inheritance **(D)** Message Passing

Ans: C

Q.13 RunTime Polymorphism is achieved by _____

(A) friend function **(B)** virtual function
(C) operator overloading **(D)** function overloading

Ans: B

Q.14 A function call mechanism that passes arguments to a function by passing a copy of the values of the arguments is _____

(A) call by name **(B)** call by value
(C) call by reference **(D)** call by value result

Ans: B

Q.15 In C++, dynamic memory allocation is accomplished with the operator ____

(A) new **(B)** this
(C) malloc( ) **(D)** delete

Ans: A

Q.16 If we create a file by 'ifstream', then the default mode of the file is _____

(A) ios :: out **(B)** ios :: in
(C) ios :: app **(D)** ios :: binary

Ans: B

Q.17 A variable defined within a block is visible

(A) from the point of definition onward in the program.
(B) from the point of definition onward in the function.
(C) from the point of definition onward in the block.
(D) throughout the function.

Ans: C

Q.18 The break statement causes an exit

(A) from the innermost loop only. **(B)** only from the innermost switch.
(C) from all loops & switches. **(D)** from the innermost loop or switch.

Ans: D

Q.19 Which of the following cannot be legitimately passed to a function

(A) A constant. **(B)** A variable.
(C) A structure. **(D)** A header file.

Ans: D

Q.20 A property which is not true for classes is that they
(A) are removed from memory when not in use.
(B) permit data to be hidden from other classes.
(C) bring together all aspects of an entity in one place.
(D) Can closely model objects in the real world.

Ans: C

Q.21 You can read input that consists of multiple lines of text using
(A) the normal cout << combination.
(B) the cin.get( ) function with one argument.
(C) the cin.get( ) function with two arguments.
(D) the cin.get( ) function with three arguments.

Ans: C

Q.22 The keyword *friend* does not appear in
(A) the class allowing access to another class.
(B) the class desiring access to another class.
(C) the private section of a class.
(D) the public section of a class.

Ans: C

Q.23 The process of building new classes from existing one is called
(A) Structure. **(B)** Inheritance.
(C) Polymorphism. **(D)** Template.

Ans: B

Q.24 If you wanted to sort many large objects or structures, it would be most efficient to

(A) place them in an array & sort the array.
(B) place pointers to them in an array & sort the array.
(C) place them in a linked list and sort the linked list.
(D) place references to them in an array and sort the array.

Ans: C

Q.25 Which statement gets affected when i++ is changed to ++i?
(A) i = 20; i++;
(B) for (i = 0; i<20; i++) { }

(C) a = i++;
(D) while (i++ = 20) cout <<i;

Ans: A

Q.26 A friend function to a class, C cannot access
(A) private data members and member functions.
(B) public data members and member functions.
(C) protected data members and member functions.
(D) the data members of the derived class of C.

Ans: D

Q.27 The operator that cannot be overloaded is
(A) ++ **(B)** ::
(C) ( ) **(D)** ~

Ans: B

Q.28 A struct is the same as a class except that
(A) there are no member functions.
(B) all members are *public*.
(C) cannot be used in inheritance hierarchy.
(D) it does have a *this* pointer.

Ans: C

Q.29 Pure virtual functions
(A) have to be redefined in the inherited class.
(B) cannot have *public* access specification.
(C) are mandatory for a virtual class.
(D) None of the above.

Ans: A

Q.30 Additional information sent when an exception is thrown may be placed in
(A) the throw keyword.
(B) the function that caused the error.
(C) the catch block.
(D) an object of the exception class.

Ans: C

Q.31 Use of virtual functions implies
(A) overloading. **(B)** overriding.
(C) static binding. **(D)** dynamic binding.

Ans: D

Q.32 *this* pointer

(A) implicitly points to an object.
(B) can be explicitly used in a class.
(C) can be used to return an object.
(D) All of the above.

Ans: D

Q.33 Within a *switch* statement

(A) *Continue* can be used but *Break* cannot be used
(B) *Continue* cannot be used but *Break* can be used
(C) Both *Continue* and *Break* can be used
(D) Neither *Continue* nor *Break* can be used

Ans:B

Q.34 Data members which are *static*

(A) cannot be assigned a value
(B) can only be used in *static* functions
(C) cannot be defined in a *Union*
(D) can be accessed outside the class

Ans:B

Q.35 Which of the following is false for cin?

(A) It represents standard input.
(B) It is an object of istream class.
(C) It is a class of which stream is an object.
(D) Using cin the data can be read from user's terminal.

Ans:C

Q.36 It is possible to declare as a *friend*

(A) a member function **(B)** a global function
(C) a class **(D)** all of the above

Ans:D

Q.37 In multiple inheritance

(A) the base classes must have only default constructors
(B) cannot have virtual functions
(C) can include virtual classes
(D) None of the above.

Ans:C

Q.38 Declaration of a pointer reserves memory space
(A) for the object.
(B) for the pointer.
(C) both for the object and the pointer.
(D) none of these.

Ans:B

Q.39 for (; ;)
(A) means the test which is done using some expression is always true
(B) is not valid
(C) will loop forever
(D) should be written as for( )

Ans:C

Q.40 The operator << when overloaded in a class
(A) must be a member function **(B)** must be a non member function
(C) can be both **(A)** & **(B)** above **(D)** cannot be overloaded

Ans:C

Q.41 A *virtual* class is the same as
(A) an abstract class **(B)** a class with a virtual function
(C) a base class **(D)** none of the above.

Ans:D

Q.42 Identify the operator that is NOT used with pointers
(A) -> **(B)** &
(C) * **(D)** >>

Ans:D

Q.43 Consider the following statements

```
char *ptr;
ptr = "hello";
cout << *ptr;
```

What will be printed?
(A) first letter **(B)** entire string
(C) it is a syntax error **(D)** last letter

Ans:A

Q.44 In which case is it mandatory to provide a destructor in a class?
(A) Almost in every class

(B) Class for which two or more than two objects will be created
(C) Class for which copy constructor is defined
(D) Class whose objects will be created dynamically

Ans:D

Q.45 The members of a class, by default, are
(A) public **(B)** protected
(C) private **(D)** mandatory to specify

Ans:C

Q.46 Given a class named ***Book,*** which of the following is not a valid constructor?
(A) Book ( ) { } **(B)** Book (Book b) { }
(C) Book (Book &b) { } **(D)** Book (char* author, char* title) { }

Ans:B

Q47 Which of the statements is true in a protected derivation of a derived class from a base class?
(A) Private members of the base class become protected members of the derived class
(B) Protected members of the base class become public members of the derived class
(C) Public members of the base class become protected members of the derived class
(D) Protected derivation does not affect private and protected members of the derived class.

Ans:C

Q48 Which of the following statements is NOT valid about operator overloading?
(A) Only existing operators can be overloaded.
(B) Overloaded operator must have at least one operand of its class type.
(C) The overloaded operators follow the syntax rules of the original operator.
(D) none of the above.

Ans:D

Q.49 Exception handling is targeted at
(A) Run-time error **(B)** Compile time error
(C) Logical error **(D)** All of the above.

Ans:A

Q.50 A pointer to the base class can hold address of
(A) only base class object
(B) only derived class object
(C) base class object as well as derived class object
(D) None of the above

Ans:C

Q.51 Function templates can accept
- **(A)** any type of parameters
- **(B)** only one parameter
- **(C)** only parameters of the basic type
- **(D)** only parameters of the derived type

Ans:C

Q.52 How many constructors can a class have?
- **(A)** 0
- **(B)** 1
- **(C)** 2
- **(D)** any number

Ans:D

Q.53 The new operator
- **(A)** returns a pointer to the variable
- **(B)** creates a variable called new
- **(C)** obtains memory for a new variable
- **(D)** tells how much memory is available

Ans:C

Q.54 Consider the following statements:

```
int x = 22,y=15;
x = (x>y) ? (x+y) : (x-y);
```

What will be the value of x after executing these statements?
- **(A)** 22
- **(B)** 37
- **(C)** 7
- **(D)** Error. Cannot be executed

Ans:B

Q.55 An exception is caused by
- **(A)** a hardware problem
- **(B)** a problem in the operating system
- **(C)** a syntax error
- **(D)** a run-time error

Ans:D

Q.56 A template class
- **(A)** is designed to be stored in different containers
- **(B)** works with different data types
- **(C)** generates objects which must be identical
- **(D)** generates classes with different numbers of member functions.

Ans:B

Q.57 Which of the following is the valid class declaration header for the derived class **d** with base classes **b1** and **b2**?

(A) class **d** : public **b1**, public **b2** **(B)** class **d** : class **b1**, class **b2**
(C) class **d** : public **b1**, **b2** **(D)** class **d** : **b1**, **b2**

Ans:A

Q.58 A library function exit() causes an exit from

(A) the loop in which it occurs **(B)** the block in which it occurs
(C) the function in which it occurs **(D)** the program in which it occurs

Ans:D

Q.59 RunTime polymorphism is achieved by _____

(A) friend function **(B)** virtual function
(C) operator overloading **(D)** function overloading

Ans:B

Q.60 Declaration of a pointer reserves memory space

(A) for the object.
(B) for the pointer.
(C) both for the object and the pointer.
(D) none of these.

Ans:B

Q.61 An array element is accessed using

(A) a FIFO approach **(B)** an index number
(C) the operator **(D)** a member name

Ans:B

Q.62 If there is a pointer p to object of a base class and it contains the address of an object of a derived class and both classes contain a virtual member function abc(), then the statement p->abc(); will cause the version of abc() in the _____class to be executed.

(A) Base Class **(B)** Derived class
(C) Produces compile time error **(D)** produces runtime error

Ans:B

Q.63 A pure virtual function is a virtual function that

(A) has no body **(B)** returns nothing
(C) is used in base class **(D)** both **(A)** and **(C)**

Ans:D

Q.64 A static function

(A) should be called when an object is destroyed.

(B) is closely connected with and individual object of a class.
(C) can be called using the class name and function name.
(D) is used when a dummy object must be created.

Ans:C

Q.65 We can output text to an object of class *ostream* using the insertion operator<< because
(A) the *ostream* class is a stream
(B) the insertion operator works with all classes.
(C) we are actually outputting to cout.
(D) the insertion operator is overloaded in *ostream.*

Ans:D

Q.66 The statement f1.write((char*)&obj1, sizeof(obj1));
(A) writes the member function of obj1 to f1.
(B) Writes the data in obj1 to f1.
(C) Writes the member function and the data of obj1 to f1.
(D) Writes the address of obj1 to f1.

Ans:B

Q.67 To convert from a user defined class to a basic type, you would most likely use.
(A) A built-in conversion function.
(B) A one-argument constructor.
(C) A conversion function that's a member of the class.
(D) An overloaded '=' operator.

Ans:C

Q.68 Which of the following is not the characteristic of constructor.
(A) They should be declared in the public section.
(B) They do not have return type.
(C) They can not be inherited.
(D) They can be virtual.

Ans:D

Q.69 Name the header file to be included for the use of built in function isalnum()
(A) string.h **(B)** process.h
(C) ctype.h **(D)** dos.h

Ans:C

Q.70 What is the output of given code fragment?

```
int f=1, i=2;
while(++i<5)
```

```
f*=i;
cout<<f;
```

(A) 12 **(B)** 24
(C) 6 **(D)** 3

Ans:A

Q.71 A class defined within another class is:

(A) Nested class **(B)** Inheritance
(C) Containership **(D)** Encapsulation

Ans:A

Q.72 What will be the values of x, m and n after the execution of the following statements?

```
int x, m, n;
m = 10;
n = 15;
x = ++m  +  n++;
```

(A) x=25, m=10, n=15 **(B)** x=26, m=11, n=16
(C) x=27, m=11, n=16 **(D)** x=27, m=10, n=15

Ans:B

Q.73 Which of the following will produce a value 10 if x = 9.7?

(A) floor(x) **(B)** abs(x)
(C) log(x) **(D)** ceil(x)

Ans:D

Q74 The major goal of inheritance in c++ is:

(A) To facilitate the conversion of data types.
(B) To help modular programming.
(C) To extend the capabilities of a class.
(D) To hide the details of base class.

Ans:C

Q.75 Consider the following class definitions:

```
class a
{
};
class b: protected a
{
};
```

What happens when we try to compile this class?

(A) Will not compile because class body of a is not defined.
(B) Will not compile because class body of b is not defined.

(C) Will not compile because class a is not public inherited.
(D) Will compile successfully.

Ans:D

Q.76 Which of the following expressions is illegal?

(A) (10|6). **(B)** (false && true)
(C) bool (x) = (bool)10; **(D)** float y = 12.67;

Ans:C

Q.77 The actual source code for implementing a template function is created when

(A) The declaration of function appears.
(B) The function is invoked.
(C) The definition of the function appears.
(D) None of the above.

Ans:B

Q.78 An exception is caused by

(A) a runtime error.
(B) a syntax error.
(C) a problem in the operating system.
(D)a hardware problem.

Ans:A

Q.79 Which of the following statements are true in c++?

(A) Classes can not have data as public members.
(B) Structures can not have functions as members.
(C) Class members are public by default.
(D) None of these.

Ans:B

Q.80 What would be the output of the following program?

```
int main()
{
int x,y=10,z=10;
x = (y = =z);
cout<<x;
return 0;
}
```

(A) 1 **(B)** 0
(C) 10 **(D)** Error

Ans:A

Q.81 What is the error in the following code?

```
class t
{
virtual void print();
}
```

(A) No error.
(B) Function print() should be declared as static.
(C) Function print() should be defined.
(D) Class t should contain data members.

Ans:A

Q.82 What will be the output of following program?

```
#include<iostream.h>
void main()
{
float x;
x=(float)9/2;
cout<<x;
}
```

(A) 4.5 **(B)** 4.0
(C) 4 **(D)** 5

Ans:A

Q.83 A white space is :
(A) blank space **(B)** new line
(C) tab **(D)** all of the above

Ans:D

Q.84 The following can be declared as friend in a class
(A) an object **(B)** a class
(C) a public data member **(D)** a private data member

Ans:B

Q.85 What would be the output of the following?

```
#include<iostream.h>
void main()
{
char *ptr="abcd"
char ch;
ch = ++*ptr++;
```

```
cout<<ch;
}
```

(A) a **(B)** b
(C) c **(D)** d

Ans:B

Q.86 A copy constructor takes
(A) no argument **(B)** one argument
(C) two arguments **(D)** arbitrary no. of arguments

Ans:B

Q87 Overloading a postfix increment operator by means of a member function takes
(A) no argument **(B)** one argument
(C) two arguments **(D)** three arguments

Ans:A

Q88 Which of the following ways are legal to access a class data member using **this** pointer?
(A) this.x **(B)** *this.x
(C) *(this.x) **(D)** (*this).x

Ans:D

Q.89 If we store the address of a derived class object into a variable whose type is a pointer to the base class, then the object, when accessed using this pointer.
(A) continues to act like a derived class object.
(B) Continues to act like a derived class object if virtual functions are called.
(C) Acts like a base class object.
(D) Acts like a base class, if virtual functions are called.

Ans:B

Q.90 Which of the following declarations are illegal?
(A) void *ptr; **(B)** char *str = “hello”;
(C) char str = “hello”; **(D)** const *int p1;

Ans:C

Q.91 What will be the result of the expression 13 & 25?
(A) 38 **(B)** 25
(C) 9 **(D)** 12

Ans:C

Q.92 Which of the following operator can be overloaded through friend function?
(A) ->
(B) =
(C) ( )
(D) *

Ans:D

Q.93 To access the public function fbase() in the base class, a statement in a derived class function fder() uses the statement.fbase();

(A) fbase();

(B) fder();
(C) base::fbase();
(D) der::fder();

Ans:A

Q.94 If a base class destructor is not virtual, then
(A) It can not have a function body.
(B) It can not be called.
(C) It can not be called when accessed from pointer.
(D) Destructor in derived class can not be called when accessed through a pointer to the base class.

Ans:D

Q.95 Maximum number of template arguments in a function template is
(A) one
(B) two
(C) three
(D) many

Ans:D

Q 96 In access control in a protected derivation, visibility modes will change as follows:

(A) private, public and protected become protected
(B) only public becomes protected.
(C) public and protected become protected.
(D) only private becomes protected.

Ans:C

Q 97 Which of the following statement is valid?
(A) We can create new C++ operators.
(B) We can change the precedence of the C++ operators.
(C) We can change the associativity of the C++ operators.
(D) We can not change operator templates.

Ans:D

Q.98 What will be the output of the following program?

```
#include<iostream.h>
void main()
{
 float x=5,y=2;
int result;
result=x % y;
cout<<result;
}
```

(A) 1 **(B)** 1.0
(C) Error message **(D)** 2.5

Ans:C

Q.99 Which can be passed as an argument to a function?
(A) constant **(B)** expression
(C) another function **(D)** all of the above.

Ans:A

Q.100 Member functions, when defined within the class specification:
(A) are always inline.
(B) are not inline.
(C) are inline by default, unless they are too big or too complicated.
(D) are not inline by default.

Ans:A

Code: AC-11 **Subject:OBJECT ORIENTED PROGRAMMING**
PART – II, VOL – I

DESCRIPTIVES

Q.1 Explain the following. **(16)**

(i) Conversion from Class to Basic Type.
(ii) File Pointers.
(iii) Function Prototyping.
(iv) Overload resolution.

Ans:(i) **Conversion from Class to Basic Type**: We know that conversion from a basic to class type can be easily done with the help of constructors. The conversion from class to basic type is indeed not that easy as it cannot be done using constructors. for this conversion we need to define an overloaded casting operator. This is usually referred to as a conversion function. The syntax for this is as follows:
The above function shall convert a class type data to typename.
A conversion function must follow the following 3 rules:
a. It cannot have a return type
b. It has to be declared inside a class.
c. It cannot have any arguments.
(ii)**File pointers**: we need to have file pointers viz. input pointer and output pointer. File pointers are required in order to navigate through the file while reading or writing. There are certain default actions of the input and the output pointer. When we open a file in read only mode, the input pointer is by default set at the beginning. When we open a file in write only mode, the existing contents are deleted and the file pointer is attached in the beginning.
C++ also provides us with the facility to control the file pointer by ourselves. For this, the following functions are supported by stream classes: seekg(), seekp(), tellg(), tellp().
(iii) **Function prototyping** is used to describe the details of the function. It tells the compiler about the number of arguments, the type of arguments, and the type of the return values. It some what provides the compiler with a template that is used when declaring and defining a function. When a function is invoked, the compiler carries out the matching process in which it matches the declaration of the function with the arguments passed and the return type. In C++, function prototype was made compulsory but ANSI C makes it optional. The syntax is as follows:
type function-name(argument-list);
example int func(int a, int b, int c);
(iv)**Overload resolution**: When we overload a function, the function prototypes are matched by the compiler. The best match is found using the following steps.
a. The compiler first matches that prototype which has the exactly same types of actual arguments.
b. If an exact match is not found by the first resolution, the integral promotions to the actual arguments are used like char to int, float to double.
c. When both the above methods fail, the built in conversions to the actual arguments are used like long square (long n).
d. If all the above steps do not help the compiler find a unique match, then the compiler uses the user defined convergence, in combination with integral promotions and built in conversions.

Q.2 Write a C++ program that prints **(6)**

```
1    1.0000e+00    1.000
2    5.0000e-01    1.500
3    3.3333e-01    1.833
```

Ans:
```
#include<iostream.h>
#include<iomanip.h>
void main()
{
 float x,y,z;
 x=1.0000e+00;
 y=5.0000e-01;
 z=3.3333e-01;
 cout.setf(ios::showpoint);
 cout.precision(3);
 cout.setf(ios::fixed,ios::floatfield);
 cout<<x<<'\n';
 cout<<y<<'\n';
 cout<<z<<'\n';
}
```

Q.3 What are friend functions? Explain there characteristics with a suitable example. **(6)**

Ans: We generally have a belief that a non member function cannot access the private data member of the class. But this is made possible using a friend function. A friend function is that which does not belong to the class still has complete access to the private data members of the class. We simply need to declare the function inside the class using the keyword "friend". The syntax for declaring the friend function is as follows:

```
class demo
{
public:
 friend void func (void);
}
```

The characteristics of a friend function are as follows.

- It can be declared both in the private and the public part of the class without altering the meaning.
- It is not called using the object of the class.
- To access the data members of the class, it needs to use the object name, the dot operator and the data member's name. example obj.xyz
- It is invoked like a normal function.
- It does not belong to the class

An example program is as follows:

```
#include<iostream.h>
using namespace std;
class one;
```

```
class two
{
 int a;
   public:
 void setvalue(int n){a=n;}
 friend void max(two,one);
};
class one
{
 int b;
   public:
 void setvalue(int n){b=n;}
 friend void max(two,one);
};

void max(two s,one t)
{
 if(s.a>=t.b)
            cout<<s.a;
 else
            cout<<t.b;
}

int main()
{
 one obj1;
 obj1.setvalue(5);
 two obj2;
 obj2.setvalue(10);
 max(obj2,obj1);
 return 0;
          }
```

Q.4 write a program to overload the operator '+' for complex numbers. **(7)**

Ans:

To perform complex number addition using operator overloading.

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

class complex                    {
```

```
        float r, i;
    public:
    complex()
    {
            r=i=0;
    }
    void getdata()
    {
        cout<<"R.P";
        cin>>r;
        cout<<"I.P";
        cin>>i;
    }
    void outdata (char*msg)
    {
        cout<<endl<,msg;
        cout<<"("<<r;
        cout<<"+j" <<i<<")";
    }
    Complex operator+(Complex);
};
Complex complex::Operator+(Complex(2))
{
    complex temp;
    temp.r=r+c2.r;
    temp.i=I=c2.i;
    return(temp);
}
void main()
{
    clrscr();
    complex c1, c2, c3;
    cout<<"Enter 2 complex no: "<<endl;
    cl.getdta();
    c2.getdata();
    c3=c1+c2;
    c3.outdata ("The result is :");
    getch();
}
```

OUTPUT

Enter 2 complex no: R.P: 2 I.P: 2 R.p: 2 I.P:2

The result is: 4+j4

RESULT

Thus the complex number addition has been done using operator overloading

Q.5 Write a complete C++ program to do the following :

(i) 'Student' is a base class, having two data members: entryno and name;

entryno is integer and name of 20 characters long. The value of entryno is 1 for Science student and 2 for Arts student, otherwise it is an error.

(ii) 'Science' and 'Arts' are two derived classes, having respectively data items marks for Physics, Chemistry, Mathematics and marks for English, History, Economics.

(iii) Read appropriate data from the screen for 3 science and 2 arts students.

(iv) Display entryno, name, marks for science students first and then for arts students. **(14)**

Ans:

```
 #include<iostream.h>
class student
{
protected:
 int entryno;
 char name[20];
public:
 void getdata()
 {
            cout<<"enter name of the student"<<endl;
      cin>>name;
 }
 void display()
 {

 cout<<"Name of the student is"<<name<<endl;
 }

};
class science:public student
{

  int pcm[3];
public:
 void getdata()
 {
                        student::getdata();
                        cout<<"Enter marks for Physics,Chemistry and Mathematics"<<endl;
              for(int j=0;j<3;j++)
              {
                        cin>>pcm[j];
              }

 }
  void display()
```

```
 {
   entryno=1;
          cout<<"entry no for Science student is"<<entryno<<endl;
                student::display();
                cout<<"Marks in Physics,Chemistry and Mathematics are"<<endl;
            for(int j=0;j<3;j++)
            {
                cout<<pcm[j]<<endl;;
            }

 }
};
class arts:public student
{

 int ehe[3];
public:
 void getdata()
 {
                student::getdata();
                cout<<"Enter marks for English,History and Economics"<<endl;
            for(int j=0;j<3;j++)
            {
                cin>>ehe[j];
            }

 }
 void display()
 {
   entryno=2;
 cout<<"entry no for Arts student is"<<entryno<<endl;;
                student::display();
                cout<<"Marks in English,History and Economics are"<<endl;
            for(int j=0;j<3;j++)
  {
                cout<<ehe[j]<<endl;;
            }

 }
};
void main()
{
 science s1[3];
 arts a1[3];
 int i,j,k,l;
 cout<<"Entry for Science students"<<endl;
 for(i=0;i<3;i++)
```

```
 {
 s1[i].getdata();
 }
 cout<<"Details of three Science students are"<<endl;
 for(j=0;j<3;j++)
 {
 s1[j].display();
 }

cout<<"Entry for Arts students"<<endl;
 for(k=0;k<3;k++)
 {
 a1[k].getdata();
 }
 cout<<"Details of three Arts students are"<<endl;
 for(l=0;l<3;l++)
 {
 a1[l].display();
 }

}
```

Q.6 An electricity board charges the following rates to domestic users to discourage large consumption of energy :

For the first 100 units – 50 P per unit
Beyond 300 units – 60 P per unit

If the total cost is more than Rs.250.00 then an additional surcharge of 15% is added on the difference. Define a class Electricity in which the function Bill computes the cost. Define a derived class More_Electricity and override Bill to add the surcharge. **(8)**

Ans:

```
#include<iostream.h>
class electricity
{
protected:
 float unit;
 float cost;
public:
 void bill()
 {
        cout<<"\n enter the no. of units"<<endl;
        cin>>unit;
        if(unit<=100)
        {
                cost=0.50*unit;
```

```
                cout<<"cost up to 100 unit is Rs."<<cost<<endl;
        }
        else
        {
                if(unit>300)
                {
                        cost=0.60*unit;
                        cout<<"Beyond 300 units is Rs"<<cost;
                }
        }
 }
};
class more_electricity:public electricity
{
 float surcharge,diff,total_cost;
public:
 void bill()
 {
        electricity::bill();
        if(cost>250.00)
        {
                diff=cost-250;
                surcharge=diff*0.15;
                total_cost=cost+surcharge;
                cout<<" Bill amount with surcharge is Rs"<<total_cost;
        }
        else
        {
                cout<<"Bill amout is Rs."<<cost;
        }}};
void main()
{
 more_electricity me;
 me.bill();
}
```

Q.7 Write a program to open a file in C++ “Hello.dat” and write
“This is only a test”
“Nothing can go wrong”
“All things are fine…”
into the file. Read the file and display the contents. **(6)**

Ans:

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

```
{
 ofstream fout;
 fout.open("Hello.dat");
 fout<<"This is only a test\n";
 fout<<"Nothing can go wrong\n";
 fout<<"All things are fine....\n";
        fout.close();
        const int N=100;
        char line[N];
 ifstream fin;
 fin.open("Hello.dat");
 cout<<"Contents of the Hello.dat file\n";
 while(fin)
 {
        fin.getline(line,N);
        cout<<line;
 }
 fin.close();}
```

Q.8 Develop a program in C++ to create a database of the following items of the derived class. Name of the patient, sex, age, ward number, bed number, nature of illness, date of admission. Design a base class consisting of data members : name of the patient, sex and age ; and another base class consisting of the data members : bed number and nature of the illness. The derived class consists of the data member ,date of admission.
Program should carry out the following methods
(i) Add a new entry.
(ii) List the complete record. **(10)**

Ans:.

```
#include<conio.h>
#include<iostream.h>
class A{
 public:
 char name[20];
 char sex[10];
 int age;
 void get_data();
 void disp_data();
};
void A::get_data(){
 cout<<"enter d name:";
 cin>>name;
cout<<"enter d age:";
 cin>>age;
 cout<<"enter d sex:";
 cin>>sex;}
void A::disp_data(){
 cout<<"name:"<<name<<'\n';
```

```
 cout<<"age:"<<age<<'\n';
 cout<<"sex:"<<sex<<'\n';}
class B{
 public:
 int bed_number;
 char nature_illness[40];
 void get_data();
void disp_data();
};
void B::get_data(){
 cout<<"enter d bed_number:";
 cin>>bed_number;
 cout<<"enter d nature_illness:";
 cin>>nature_illness;
}
void B::disp_data(){
 cout<<"bed_number:"<<bed_number<<'\n';
 cout<<"nature_illness:"<<nature_illness<<'\n';
}
class C:public A,public B{
 int date_admission;
 public:
void get_data();
 void disp_data();
 void record();
};
void C::get_data(){
        A::get_data();
        B::get_data();
        cout<<endl<<"Enter Data of Admission:-> ";
        cin>>date_admission;
}
void C::disp_data(){
        A::disp_data();
        B::disp_data();
        cout<<endl<<"Date of Admission\t"<<date_admission;
}
void main(){
 clrscr(); C c1;
 cout<<endl<<"Adding a new record to database\n";
 getch();
 c1.get_data();
 cout<<endl<<"Displaying the added record to database\n";
 c1.disp_data();
 getch();
}
```

Q.9 How many times is the copy constructor called in the following code? **(4)**

```
Apple func(Apple u)
{
Apple w=v;
Return w;
}
void main()
{
Apple x;
Apple y = func (x);
Apple z = func (y);
}
```

Ans: 2 times

Q.10 Write a program code which throws an exception of type char* and another of type int. Write a try ---- catch block which can catch both the exception. **(5)**

Ans:

```
#include <iostream>
    using namespace std;
 int main()
{
 double Operand1, Operand2, Result;

 // Request two numbers from the user
 cout << "This program allows you to perform a division of two numbers\n";
 cout << "To proceed, enter two numbers: ";
 try {
            cout << "First Number: ";
            cin >> Operand1;
            cout << "Second Number: ";
            cin >> Operand2;

// Find out if the denominator is 0
            if( Operand2 == 0 )
                        throw "Division by zero not allowed";

// Perform a division and display the result
            Result = Operand1 / Operand2;
```

```
            cout << "\n" << Operand1 << " / " << Operand2 << " = " << Result << "\n\n";
 }
 catch(const char* Str) // Catch an exception
 {
            // Display a string message accordingly
            cout << "\nBad Operator: " << Str;
 }

 return 0;
#include <iostream.h>

int main()
{
 double Operand1, Operand2, Result;
 const char Operator = '/';

 // Request two numbers from the user
 cout << "This program allows you to perform a division of two numbers\n";
 cout << "To proceed, enter two numbers\n";

 try {
            cout << "First Number: ";
            cin >> Operand1;
            cout << "Second Number: ";
            cin >> Operand2;
               // Find out if the denominator is 0
               if( Operand2 == 0 )
                        throw 0;
            // Perform a division and display the result
            Result = Operand1 / Operand2;
            cout << "\n" << Operand1 << " / " << Operand2 << " = " << Result << "\n\n";
 }
 catch(const int n) // Catch an exception
 {
            // Display a string message accordingly
            cout << "\nBad Operator: Division by " << n << " not allowed\n\n";
 }

 return 0;
}
```

Q.11 Create a class Time that has separate int member data for hours, minutes and seconds. One constructor should initialize this data to zero and another constructor initialize it to fixed values. Write member function to display time in 12 hour as well as 24 hour format. The final member function should add two objects of class Time.
A main() program should create three objects of class time, of which two are initialized to specific values and third object initialized to zero. Then it should add the two

initialized values together, leaving the result in the third. Finally it should display the value of all three objects with appropriate headings. **(14)**

Ans:

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

class time24
{

 int hours,minutes,seconds;
public:
 time24()
   {
        hours=minutes=seconds=0;
 }
 time24(int h,int m,int s)
 {
        hours=h;
        minutes=m;
        seconds=s;
 }
  void display()
 {
        if(hours<10)
                cout<<'0';
        cout<<hours<<":";
        if(minutes<10)
                cout<<'0';
        cout<<minutes<<":";

        if(seconds<10)
                cout<<'0';
        cout<<seconds;
 }
};
class time12
{
bool pm;
 int hour,minute;

public:
 time12()
   {
        pm=true;
```

```
            hour=minute=0;
 }
 time12(bool ap,int h,int m)
 {
            pm=ap;
            hour=h;
            minute=m;
 }
 time12(time 24);
 void display()
 {
            cout<<hour<<":";
            if(minute<10)
                        cout<<'0';
            cout<<minute<<" ";

 char *am_pm=pm ? "p.m." : "a.m.";
 cout<<am_pm;
 }
};

time12::time12(time24 t24)
{
 int hrs24=hours;
 bool pm=hours<12 ? false:true;
            int min=seconds<30 ? minutes:minutes+1;
            if(min==60)
            {
                        min=0;
                        ++hrs24;
                        if(hrs24==12 || hrs24==24)
                                    pm=(pm==true)? false:true;
            }
            int hrs12=(hrs24<13) ? hrs24 : hrs24-12;
            if(hrs12==0)
            {
                        hrs12=12;
                        pm=false;
            }
            return time12(pm,hrs12,min);
 }

int main()
{
 int h1,m1,s1;
```

```
while(true)
{
        cout<<"enter 24-hour time:\n";
        cout<<"Hours(0-23):";
        cin>>h1;
if(h1>23)
        return(1);
cout<<"Minutes:";
cin>>m1;
cout<<"Seconds:";
cin>>s1;

time24 t24(h1,m1,s1);
cout<<"you entered:";
t24.display();
cout<<endl;
time12 t12=t24;
cout<<"\n12-hour time:";
t12.display();
cout<<endl;
}
return 0;
}
```

Q.12 In the following output, the following flag constants are used with the stream member function setf. What effect does each have

(i) ios::fixed
(ii) ios::scientific
(iii) ios::showpoint
(iv) ios::showpos
(v) ios::right
(vi) ios::left **(6)**

Ans: A two argument version of setf() is used to set flag which is a member of a group of flags. The second argument specifies the group and is a bitwise OR of all the flags in the group. The setfO, a member function of the ios class, can provide answers to these and other formatting questions. The setfO *(setf* stands for set flags) function can be us follows:

cout.setf(argl,arg2)

The *argl* is one of the formatting *flags* defined in the class ios. The formatting flag Spi the format action required for the output. Another ios constant, *arg2,* known as bi specifies the group to which the formatting flag belongs.

There are three bit and each has a group of format flags which are mutually exclusive

Examples:

cout.setf(ios::1eft, ios::adjustfied);
cout.setf(ios::scientific, ios::floatfield);
Note that the first argument should be one of the group members of the second argument.
The Setf() can be used with the flag ios::showpoint as a single argument to achieve this form of output. For example
Cout.setf(ios::showpoint);
Would cause cout to display trailing zeros and trailing decimal point.
Similarly, a plus sign can be printed a positive number using the following statement:
Cout.setf(ios::showpos);set to selection

selection	ios::ingroup	Meaning
ios::left	ios::adjustfield	left alignment
ios::right	ios::adjustfield	right alignment
ios::fixed	ios::floatfield	fixed form of floating point
ios::scientific	ios::floatfield	force exponential format
ios::showpoint //		Always show a point
ios::showpos //		If positive still show sign

Q.13 Define a class Distance with feet and inch and with a print function to print the distance. Write a non-member function max which returns the larger of two distance objects, which are arguments. Write a main program that accepts two distance objects from the user, compare them and display the larger. **(8)**

Ans:

```
#include<iostream.h>
class distance
{
 int feet;
 float inches;
public:
 distance()
 {
        feet=0;
        inches=0.0;
 }

 void getdist()
 {
        cout<<"Enter feet:"<<endl;
        cin>>feet;
        cout<<"Enter inches:"<<endl;
        cin>>inches;
 }
 void showdist()
 {
        cout<<feet<<endl<<inches<<endl;
 }
 friend void maxdist(distance d1, distance d2)
 {
 d1.feet=d1.feet+ d1.inches/12;
```

```
 d2.feet=d2.feet+ d2.inches/12;
 if(d1.feet>d2.feet)
 {
            cout<<"first distance is greater";
 }
 else
 {
 cout<<"second distance is greater";
 }
 }
};
void main()
{
 distance d3,d4;
 d3.getdist();
 d3.showdist();
 d4.getdist();
   d4.showdist();
 maxdist(d3,d4);
}
```

Q.14 a) Define the class Person which has name (char*) and age (int). Define the following constructors

a. default
b. with name as argument (assume age is 18)
c. with name and age as argument

Also define the copy constructor. **(7)**

Ans:

```
#include<iostream.h>
#include<string.h>
class person
{
char *name;
int age;
public:
person()
{
 name=NULL;
//strcpy(name,NULL);
age=0;
}
person(char *n)
{
 strcpy(name,n);
age=18;
}
person(char *n1,int a)
```

```
{
 strcpy(name,n1);
age=a;
}
person(person &p)
{
strcpy(name,p.name);
age=p.age;
}
void disp()
{
cout<<"name is"<<name;
cout<<"age is"<<age;
}
};
void main()
{
person p1("ram"),p2("sita",20),p3=p2;
cout<<"hello";
p1.disp();
p2.disp();
p3.disp();
}
```

Q.14 b). Using the class above, define two subclasses Student and Professor. Student subclass displays the name and CGPA (grade points in float) and Professor subclass displays the name and number of publications (int). Write a main program using polymorphism to display the data of one student and one professor. **(7)**

Ans:

```
 #include<iostream.h>
class person
{
protected:
 char name[40];
public:
 void getname()
 {
 cout<<"Enter name:";
 cin>>name;
 }
 void disp()
 {
 cout<<"Name is:"<<name<<endl;
 }
 virtual void getdata()=0;
 virtual bool outstanding()=0;
};
```

```
class student:public person
{
float gpa;
public:
 void getdata()
 {
            person::getname();
            cout<<"Enter student's GPA:";
            cin>>gpa;
 }
 bool outstanding()
 {
            return(gpa>3.5) ? true:false;
 }
};
class professor:public person
{
 int numpubs;
public:
 void getdata()
 {
            person::getname();
            cout<<"enter number of professor's publication:";
            cin>>numpubs;
 }
   bool outstanding()
 {
            return(numpubs>100) ? true:false;
 }
};
void main()
{
 person *perptr[10];
 int n=0;
 char choice;
 do
 {
            cout<<" Enter student or professor(s/p):";
            cin>>choice;
            if(choice=='s')
              perptr[n]=new student;
                        else
                                    perptr[n]=new professor;
                        perptr[n++]->getdata();
                        cout<<"enter another(y/n)?";
                        cin>>choice;
 }while (choice=='y');
 for(int j=0;j<n;j++)
 {
            perptr[j]->disp();
            if(perptr[j]->outstanding())
```

```
                cout<<"this person is outstanding\n";
    }
}
```

Q.15 Write a global function which uses the above operator and prints the number and the number of times it appears (frequency) .

Ans:

```
void frequency_calc()
{
        bag b;
        for(int j=0;j<50;j++)
        {
                int temp;
                cin>>temp;
                b[j]=temp;
        }
         for(int i=0;i<50;i++)
        {
                 int k=0;
                  for(j=j+1;j<50;j++)
                 {
                        if (b[i]==b[j])
                           k++;
                  }
                cout<<b[i]<<"occurred"<<k<<"times";
        }
}
```

Q.16 How are template functions overloaded? Explain with a suitable example. **(8)**

Ans: A template function can be overloaded using a template functions. It can also be overloaded using ordinary functions of its name. The matching of the overloaded function is done in the following manner.

a. The ordinary function that has an exact match is called.
b. A template function that could be created with an exact match is called.
c. The normal matching process for overloaded ordinary functions is followed and the one that matches is called.

If no match is found, an error message is generated. Automatic conversion facility is not available for the arguments on the template functions. The following example illustrates an overloaded template function:

```
#include<iostream.h>
#include<string.h>
using namespace std;
template<class T>
void display(T x)
{
 cout<<"Template display: "<<x<<"\n";
}
void display(int x)
{
 cout<<"Explicit display: "<<x<<"\n";
}

int main()
{
 display(20);
 display(3.6);
 display('A');
 return 0;
}
```

Q.17 Give a function template SEARCH that can be used to search an array of elements of any type and returns the index of the element, if found. Give both the function prototype and the function definition for the template. Assume a class template Array <T> is available. **(5)**

Ans:

```
#include<iostream.h>
template<class atype>

//function returns index number of item, or -1 if not found
int find(atype *array,atype value,int size)
{

 for(int j=0;j<size;j++)
            if(array[j]==value)
                        return j;
            return -1;
```

```
}
char chrarr[]={1,3,5,9,11,13};
char ch=5;
int intarr[]={1,3,5,9,11,13};
int in=6;
long lonarr[]={1L,3L,5L,9L,11L,13L};
long lo=11L;
double dubarr[]={1.0,3.0,5.0,9.0,11.0,13.0};
double db=4.0;
int main()
{
 cout<<"\n 5 in chrarr: index="<<find(chrarr,ch,6);
 cout<<"\n 6 in intarr: index="<<find(intarr,in,6);
 cout<<"\n 11 in lonarr: index="<<find(lonarr,lo,6);
   cout<<"\n 4 in dubarr: index="<<find(dubarr,db,6);

cout<<endl;
return 0;

}
```

Q.18 Answer briefly :

(i) What happens in a while loop if the control condition is false initially ?

(ii) What is wrong with the following loop

```
while (n <= 100)
Sum += n*n;
```

(iii) What is wrong with the following program

```
int main( )
        {
        const double PI;
        int n;
        PI = 3.14159265358979;
        n = 22;
        }
```

(iv) When a multidimensional array is passed to a function, why does C++ require all but the first dimension to be specified in parameter list?

(v) Why can't ** be overloaded as an exponentiation operator?

(vi) How many constructors can a class have, and how should they be distinguished.

(2x6=12)

Ans:

i) If the condition in while loop initially false the loop does not enter into the statements inside the loop and terminates initially.

ii) In the given statement of while loop n is not initialized. Although there is no initialization expression, the loop variable n must be initialized before the loop begins.

iii) The given program code is wrong as const declared PI must be initialized at the time of declaration. Whereas initialization of int n is correct the correct code is
Int main()
{Const double PI=3.1459265358979;
Int n=22;}

iv) Multidimensional array is an array of arrays the function first take the argument as an array of parameter first. It does not need to know how many values in parameter there are, but it does need to know how big each parameter element is, so it can calculate where particular element is. So we must tell it the size of each element but not how many there are.

v) Only existing operator symbols may be overloaded. New symbols, such as ** for exponentiation, cannot be defined.

vi) A class can have Default constructor, constructor with an argument, constructor with two argument and copy constructor.

Q.19 What are the various ways of handling exceptions? Which one is the best? Explain. **(6)**

Ans: Exception handling is a mechanism that finds errors at run time. It detects an error and reports so that an appropriate action can be taken. It adopts the following ways:

a. Hit the exception
b. Throw the exception
c. Catch the exception
d. Handle the exception

The three blocks that is used in exception handling are try throw and catch. A further advancement that is made is of multiple catch handlers.

A program may have more than one condition to throw an exception. For this we can use more than one catch block in a program. An example is shown below illustrate the concept:

```
#include<iostream.h>
using namespace std;
void test(int x){
 try        {
            if(x==1) throw x;
            else
                        if(x==0) throw 'x';
            else
                        if(x==-1) throw 1.0;
            cout<<"End of try-block\n";
 }
 catch(char c)      {
            cout<<"Caught a character \n";
 }
 catch(int m)       {
            cout<<"Caught an integer\n";
 }
 catch(double d)  {
            cout<<"Caught a double\n";
 }
 cout<<""End of try-catch system\n\n;}
```

```
int main(){
 cout<<"Testing multiple catches\n";
 cout<<"x==1\n";
 test(1);
 cout<<"x==0\n";
 test(0);
 cout<<"x==-1\n";
 test(-1);
 cout<<"x==2\n";
 test(2);  return 0;}
```

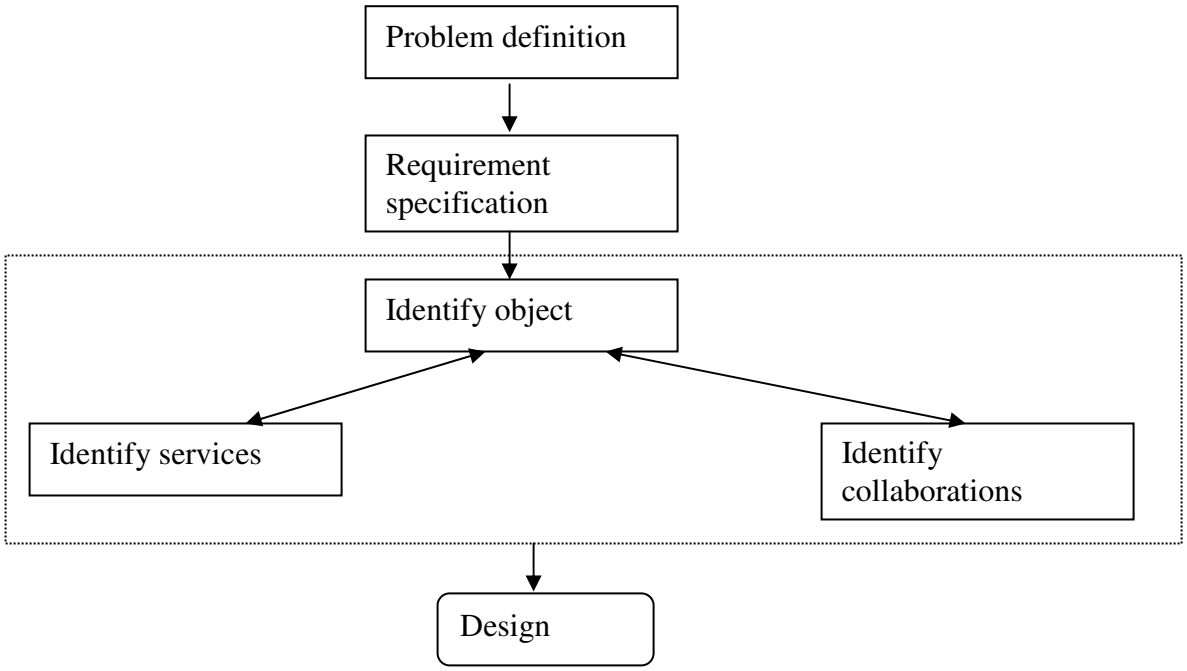
Q.20 Describe, in brief, the steps involved in object oriented analysis. **(6)**

Ans: The steps involved in object oriented analysis.

Object-oriented analysis provides us with a simple, yet powerful, mechanism for identifying objects, the building block of the software to be developed. The analysis is basically concerned with the decomposition of a problem into its component parts and establishing a logical model to describe the system functions.

The object-oriented analysis (OOA) approach consists of the following steps:

1. Understanding the problem.
2. Drawing the specifications of requirements of the user and the software.
3. Identifying the objects and their attributes.
4. Identifying the services that each object is expected to provide (interface).
5. Establishing inter-connections (collaborations) between the objects in terms of services required and services rendered.



Problem Understanding

The first step in the analysis process is to understand the problem of the user. The problem statement should be refined and redefined in terms of computer system engineering that could suggest a computer-based solution. The problem statement should be stated, as far as possible, in a single, grammatically correct sentence. This will enable the software engineers to have a highly focused attention on the solution of the problem. The problem statement provides the basis for drawing the requirements specification of both the user and the software.

Requirements Specification

Once the problem is clearly defined, the next step is to understand what the proposed. System is required to do. It is important at this stage to generate a list of user requirement A clear understanding should exist between the user and the developer of what is require Based on the user requirements, the specifications for the software should be drawn. Developer should state clearly

What outputs are required?
What processes are involved to produce these outputs?
What inputs are necessary?
What resources are required?

These specifications often serve as a reference to test the final product for its performance the intended tasks.

Identification of Objects

Objects can often be identified in terms of the real-world objects as well as the abstra~

objects. Therefore, the best place to look for objects is the application itself. The application may be analyzed by using one of the following two approaches:

1.Data flow diagrams (DFD)
2.Textual analysis (TA)

Identification of Services

Once the objects in the solution space have been identified, the next step is to identify set services that each object should offer. Services are identified by examining all the verbs are verb phrases in the problem description statement. *Doing verbs* and *compare verbs* usually give rise to services (which we call as functions C++). *Being verbs* indicate the existence of the classification structure while *having* verbs' rise to the composition structures.

Establishing interconnection

This step identifies the services that objects provide and receive. We may use an information flow diagram(IED) or an entity-relationship (ER) diagram to enlist this information.

Q.21 Write a program in C++ which calculates the factorial of a given number. **(6)**

Ans: Program to calculate factorial of a given number

```
#include <iostream.h>

        long factorial (long a)
{
 if (a > 1)
  return (a * factorial (a-1));
 else
  return (1);
```

```
}

int main ()
{
 long ln;
 cout << "Type a number: ";
 cin >> ln;
 cout << "!" << ln << " = " << factorial (ln);
 return 0;
}
```

Q.22 Define a class Queue with the following data members

(i) An array of numbers that the Queue can hold

(ii) Indices 'front' and 'rear' to specify the positions in Queue.
Initially set front = rear = -1
and the following member functions

(i) Delete – Remove front element.

(ii) Add – Add an element at rear. **(8)**

Ans:

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

struct Node{
int data;
Node *next;
}*rear,*front;
void Create();
void AddItem();
void DeleteItem();
void Display();
void main(){
int ch=0;
Create();
clrscr();
while(ch!=4){
cout<<"\n1. Add  item";
cout<<"\n2. Delete item";
cout<<"\n3. Display";
cout<<"\n4. Quit";
cout<<"\n Enter choice";
cin>>ch;
switch(ch){
case 1:
AddItem();
break;
case 2:
```

```
DeleteItem();
break;
case 3:
Display();
break;
}
}
}
void Create(){
front=rear=NULL;
}
void AddItem(){
Node *p=new Node;
cout<<"\n Enter the data:";
cin>>p->data;
p->next=NULL;
if(front==NULL){
 front=rear=p;
 }
else{
 rear->next=p;
 rear=p;
    }
}
void DeleteItem(){
if(front==NULL){
 cout<<"\nQueue is empty";
 return;
 }
if(front==rear){
 cout<<endl<<front->data<<" is deleted";
 delete front;
 front=rear=NULL;
 }
else{
 Node *p=front;
 cout<<endl<<front->data<<" is deleted";
 front=front->next;
 delete p;
 }
}
void Display(){
Node *p=front;
while(p!=NULL){
cout<<"\n"<<p->data;
p=p->next;
}}
```

Q.23 What are the two methods of opening a file? Explain with examples. What is the difference between the two methods? **(10)**

Ans: For opening a file, we first create a file stream which we can link to the file name. we can define a file stream using 3 classes namely ifstream, ofstream and fstream that are all contained in the header file fstream. Which class to use shall depend upon the mode in which we want to open the file viz, read or write. There are 2 modes to open a file.

- Using the constructor function of the class
- Using the member function "open()" of the class.

When using the first method that is constructor function of the class, we initialize the file stream object by file name. for example the following statement opens a file named "results" for input.

ifstream infile ("results");

When using the second method of function "open()", multiple files can be opened that use the same stream object for example when we wish to process a number of files in sequential manner, we shall create a single stream object and use it to open each file in turn. The syntax is as follows:

file-stream-class stream-object;

stream-object.open("filename");

The basic difference between the two methods is that the constructor function of the class is used when we wish to open just one file in the stream. The open function is used when we wish to open multiple files using one stream.

Q.24 What is containership? How does it differ from inheritance? **(4)**

Ans: Inheritance, one of the major features of OOP, is to derive certain properties of the base class in the derived class. A derived class can make use of all the public data members of its base classes.

Containership is different from Inheritance. When one class is defined inside another class, it is known as containership or nesting. We can thus imply that one object can be a collection of many different objects. Both the concepts i.e. inheritance and containership have a vast difference. Creating an object that contains many objects is different from creating an independent object. First the member objects are created in the order in which their constructors occur in the body. Then the constructor of the main body is executed. An initialization list of the nested class is provided in the constructor.

Example:

```
class alpha{....};
class beta{....};

class gamma
{
 alpha a;
 beta b;
 public:
```

```
gamma(arglist): a(arglist1), b(arglist2)     {
        //constructor body
}};
```

Q.25. How do the properties of following two derived classes differ?

(i) class X : public A{//..}

(ii) class Y : private A{//..} **(5)**

Ans: The properties of the following two derived class differ:-

(i) class X:public A{//..}

In this class A is publicly derived from class X. the keyword public specifies that objects of derived class are able to access public member function of the base class.

i) class Y:private A{//..}

In this class A is privately derived from class Y. the keyword private specifies that objects of derived class cannot access public member function of the base class. Since object can never access private members of a class.

Q.26 Identify and remove error from the following, if any :-

```
array_ptr = new int [10] [10] [10];
array_ptr = new int [20] [ ] [ ];
```

(2)

Ans: There is an error we cannot assign more than one block sizes to one variable.

Correct array_ptr new int[10];

Q.27 What is the task of the following code :-

```
cout << setw (4) << XYZ << endl
```

(2)

Ans: The task of the given code is to display the value of XYZ after setting field width 4 (3) if XYZ=3; set causes the number or string that follows it in the stream to be printed within a field n characters wide.

Q.28 What is dynamic initialization of objects? Why is it needed? How is it accomplished in C++? Illustrate. **(8)**

Ans: Dynamic initialization of objects means to provide the initial values to an object at run time.

Using dynamic initialization of objects we can provide different initialization formats of data for an object at run time. We can use various data types like int, float or char depending upon our need for the same object created. This provides the flexibility of using different format of data at run time depending upon the situation.

It can be accomplished using the concept of constructor overloading. For example:

```
#include<iostream.h>
class loan
{ long principle;
  int year;
```

```
 float rate;
 public:
 loan(){}
 loan(long p,int y,float r=0.15);
 loan(long p,int y,int r);
};
```

This class uses three overloaded constructors. The parameter values to these constructors are provided at run time. One can input in any of the form to create object of this class.

Q.29 State, with reason, whether the following statement is true or false
$!(4<5)\|\|(6>7)$ is equal to false **(2)**

Ans: The following statement is true as condition !(4<5)||(6>7) is equal to false.

Q.30 The following macro is invoked as $f(x+3)$. What is the output when x = 5
$\#\text{define}\, f(x)\ \ x * x - x$ **(3)**

Ans: The output is 21. when the macro #define f(x) x*x-3 as f(x+3) when x=5

Q.31 List at least three C++ operators which cannot be overloaded. **(3)**

Ans: The operators ::, .*, . and ?: cannot be overloaded.

Q.32 Write a template function that swaps the values of two arguments passed to it. In main( ), use the function with integers and characters. **(4)**

Ans:

```
#include<iostream.h>
using namespace std;
template <class T>
void swap(T &a,T &b)
{
 T temp=a;
 a=b;
 b=temp;
}
void func(int x,int y,char w,char z)
{
 cout<<"x and y before swap: "<<x<<y<<"\n";
 swap(x,y)
 cout<<"x and y after swap: "<<x<<y<<"\n";

 cout<<"w and z before swap: "<<w<<z<<"\n";
 swap(w,z)
 cout<<"w and z after swap: "<<w<<z<<"\n";
```

```
}
int main()
{
 fun(10,20,'s','S');
 return 0;
}
```

Q.33 What is multiple inheritance? Discuss the syntax and rules of multiple inheritance in C++. How can you pass parameters to the constructors of base classes in multiple inheritance? Explain with suitable example. **(12)**

Ans: C++ provides us with a very advantageous feature of multiple inheritance in which a derived class can inherit the properties of more than one base class. The syntax for a derived class having multiple base classes is as follows:

```
Class D: public visibility base1, public visibility base2
{
   Body of D;
}
```

Visibility may be either 'public' or 'private'.
In case of multiple inheritance, the base classes are constructed in the order in which they appear in the declaration of the derived class. However the constructors for virtual base classes are invoked before any non-virtual base classes. If there are multiple virtual base classes, they are invoked in the order in which they are declared. An example program illustrates the statement.

```
#include<iostream.h>
using namespace std;
class M
{
    protected:
            int m;
    public:
            M(int x)
            {
                    m=x;
                    cout<< "M initialized\n";
            }
};
class N
{
    protected:
            int n;
    public:
            N(int y)
            {
```

```
                    n=y;
                    cout<< "N initialized\n";
                }
};
class P: public N, public M
{       int p,r;
    public:
      P(int a,int b,int c, int d):M(a),N(b)
          {
             p=c;
        r=d;
           cout<< "P initialized\n";
    }

};
void main()
{
    P p(10,20,30,40);

}
```

Output of the program will be:
N initialized
M initialized
P initialized

Q.34 Write a program in C++ to sort a list of given numbers in nondecreasing order. **(7)**

Ans:

```
#include<iostream.h>
void main()
{           int *num,i,j,temp,n,flag;
 num=new int[10];
 cout<<"how many number to sort(limit)"<<endl;
 cin>>n;
 cout<<"enter numbers to sort in nondecreasing order"<<endl;
 for(i=0;i<n;i++)
 {
           cin>>num[i];
 }
 for(i=0;i<n-1;i++)
 {
           flag=1;
           for(j=0;j<(n-1-i);j++)
           {
```

```
                    if(num[j]>num[j+1])
                    {
                            flag=0;
                    temp=num[j];
                    num[j]=num[j+1];
                    num[j+1]=temp;
                    }
            }
            if(flag)
                    break;
 }
            for(i=0;i<n;i++)
 {
            cout<<num[i]<<endl;
 }
}
```

Q.35 What do you mean by static class members? Explain the characteristics of static class members with suitable examples. **(8)**

Ans: Static class member variables are used commonly by the entire class. It stores values. 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 variables.

It has the following characteristics:

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

An example is given below:

```
#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<<"count: "<<c<<"\n";
 }
};
int num :: c;
```

```
int main()
{
 num o1,o2,o3;
 o1.getc();
 o2.getc();
 o3.getc();

 o1.getd(1);
 o2.getd(2);
 o3.getd(3);
 cout<<"After reading data"<<"\n";
 o1.getc();
 o2.getc();
 o3.getc();
 return 0;
 }
```

Q.36 Create a class Patient that stores the patient name (a string) and the disease (a string) of the patient. From this class derive two classes : In_patient which has a data member roomrent (type float) and Out_patient which has a data member OPD_charges (float). Each of these three classes should have a nondefault constructor and a putdata() function to display its data. Write a main() program to test In_ patient and Out_patient classes by creating instances of them and then displaying the data with putdata(). **(10)**

Ans:

```
#include<iostream.h>
class patient
{
protected:
 char *name,*disease;
public:
patient(char *n,char *d)
{
 name=n;
 disease=d;
}
 void putdata()
 {
  cout<<"patient's name is"<<name<<endl;
  cout<<"Disease is"<<disease<<endl;
 }
};
class in_patient:public patient
{
 float roomrent;
public:
 in_patient(char *n,char *d,float rr):patient(n,d)
```

```
 {
            roomrent=rr;
 }
 void putdata()
 {
            patient::putdata();
            cout<<"Room rent is"<<roomrent;
            cout<<endl;
 }
};
class out_patient:public patient
{
 float opd_charges;
public:
 out_patient(char *n,char *d,float opd):patient(n,d)
 {
            opd_charges=opd;
 }

            void putdata()
 {

      patient::putdata();
            cout<<"OPD charges is"<<opd_charges;
 }
};
void main()
{
 char *nm,*dis;
 float rr1,op1;
 nm=new char;
 dis=new char;
    cout<<"Enter patient's name"<<endl;
  cin>>nm;
  cout<<"Enter disease"<<endl;
  cin>>dis;
  cout<<"Enter room rent"<<endl;
            cin>>rr1;
            cout<<"Enter opd charges"<<endl;
            cin>>op1;
 in_patient ip(nm,dis,rr1);
 out_patient op(nm,dis,op1);
 cout<<"Details of In_patient"<<endl;
 ip.putdata();
 cout<<"Details of Out_patient"<<endl;
 op.putdata();
}
```

Q.37 Consider the following specifications: **(8)**

Structure name	data	type	size
Name	First	array of characters	40
	Mid	array of characters	40
	Last	array of characters	60
Phone	Area	array of characters	4
	Exch	array of characters	4
	Numb	array of characters	6
Class name	data	type	
N_rec	Name	Name	
	Phone	Phone	

(i) Declare structures in C++ for Name and Phone.

(ii) Declare a class N_rec with the following members.

1. Define the constructor (outside the class N_rec) that gathers the information from the user for Name and Phone.
2. Define the display_rec(outside the class N_rec) that shows the current values of the data members.

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
struct Name
{
 char First[40], Mid[40], Last[60];
};

struct Phone
{
 char Area[4], Exch[4], Numb[6];
};
class N_Rec
{
 struct Name name;
 struct Phone phone;
 public:
        N_Rec();
        Display_Rec();

};

N_Rec :: N_Rec()
{
 strcpy(name.First,"Shailesh");
```

```
 strcpy(name.Mid," ");
 strcpy(name.Last,"Saurabh");
 strcpy(phone.Area,"011");
 strcpy(phone.Exch,"64");
 strcpy(phone.Numb,"608284");
}

N_Rec :: Display_Rec()
{
 cout<<name.First<<" " <<name.Mid<<" "<<name.Last;
 cout<<endl<<phone.Area<<" "<<phone.Exch<<" "<<phone.Numb;

};

main()
{
 clrscr();
 N_Rec obj1;
 obj1.Display_Rec();
 getch();
}
```

Q.38 When do we use multi catch handlers? Explain with an example. **(8)**

Ans: Multiple Catch handlers:

Just like we use conditions in a switch statement, we can use multi catch statements with one try block when a program has to throw an exception based on more than one condition.

The multi handlers are searched in sequence of which they occur. The first match that is found is executed.If no match is found then the program terminates.

An example program is given below:

```
#include<iostream.h>
using namespace std;
void test(int x)
{
 try
 {
        if(x==1) throw x;
        else
                if(x==0) throw 'x';
        else
                if(x==-1) throw 1.0;
        cout<<"End of try-block\n";
 }
 catch(char c)
 {
        cout<<"Caught a character \n";
 }
```

```
 catch(int m)       {
            cout<<"Caught an integer\n";
 }
 catch(double d)  {
            cout<<"Caught a double\n";
 }
 cout<<""End of try-catch system\n\n;
}
int main()
{
 cout<<"Testing multiple catches\n";
 cout<<"x==1\n";
 test(1);
 cout<<"x==0\n";
 test(0);
 cout<<"x==-1\n";
 test(-1);
 cout<<"x==2\n";
 test(2);
 return 0;}
```

The output:

```
Testing Multiple Catches
x==1
Caught an integer
End of try-catch system
x==0
Caught a character
End of try-catch system
x==-1
Caught a double
End of try-catch system
x==2
End of try-block
End of try-catch system
```

Q.39 Find errors in the following code: **(6)**

```
#include<iostream.h>
class A { int a1;
        public: int a2;
        protected : int  a3;    };
class B :public A
        { public:
        void  func( )
        { int b1, b2, b3;
              b1 = a1;
```

```
        b2 = a2;
        b3 = a3; } };
class C : A
        { public:
                void f( )
                { int c1, c2, c3;
                        c1 = a1;
                        c2 = a2;
                        c3 = a3;   }   };
int main( )
{ int p, q, r, i, j, k;
        B   O1;
        C   O2;
        p = O1.a1;
        q = O1.a2;
        r = O1.a3;
        i = O2.a1;
        j = O2.a2;
        k = O2.a3;
        }
```

Ans:

In the given code:

- Class B cannot have access over the private variable a1 of class A as a private variable is never inherited. Thus b1=a1 cannot be used.
- Class C also cannot have access over the private member a1 of class A as C inherits class B which does not contain a1 as its inherited member. Thus c1=a1 cannot be used.
- Inside main() function, object O1 of class B cannot access a1. Thus the statement O1.a1 is not valid.
- Similarly object O2 of class C also cannot access a1. Thus statement O2.a1 is also not valid.

Inside main() function, the protected member a3 of class B is not accessible.So the statements "r = O1.a3; and k = O2.a3; " is invalid

Q.40 Write short notes on any **TWO** :-

(i) Procedure oriented vs object oriented programming.
(ii) Object oriented design.
(iii) User defined data types in C++. **(2 x 7)**

Ans:

(i) Procedure oriented vs object oriented programming

Procedure-oriented programming basically consists of writing a list of instructions (or actions) for the computer to follow, and organizing these instructions into groups known as *functions* . We normally use a *flowchart* to organize these actions and represent the flow of control from one action to another. While we concentrate on the development of functions, very little

attention is given to the data that are being used by various functions. What happens to the data? How are they affected by the functions that work on them? In a multi-function program, many important data items are placed as *global* so that they may be accessed by all the functions. Each function may have its own *local data.* Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides an opportunity for bugs to creep in.
Another serious drawback with the procedural approach is that it does not model real world problems very well. This is because functions are action-oriented and do not really correspond to the elements of the problem.

Some characteristics exhibited by procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions. Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs *top-down* approach in program design.

Object-Oriented programming

The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside 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 be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.

Some of the striking features of object-oriented programming are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions. .
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom-up approach in program design.

Object-oriented programming is the most recent concept among programming paradigms and still means different things to different people. It is therefore important to have a working definition of object-oriented programming before we proceed further. We define "object-oriented programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand." Thus, an object is considered to be a partitioned area of computer memory that stores data and set of operations that can access that data. Since

the memory partitions are independent, the objects can be used in a variety of different programs without modifications.

ii) Object oriented design

Design is concerned with the mapping of objects in the problem space into objects in the solution space, and creating an overall structure (architectural model) and computational models of the system. This stage normally uses the bottom-up approach to build the structure of the system and the top-down functional decomposition approach to design :the class member functions that provide services. It is particularly important to construct structured hierarchies, to identify abstract classes, and to simplify the inter-object communications. Reusability of classes from the previous designs, classification of the objects into subsystems and determination of appropriate protocols are some of the 'considerations of the design stage. The object oriented design (OOD) approach may involve the following steps:

1. Review of objects created in the analysis phase.
2. Specification of class dependencies.
3. Organization of class hierarchies.
4. Design of classes.
5. Design of member functions.
6. Design of driver program.

Review of Problem Space Objects

The main objective of this review exercise is to refine the objects in terms of their attributes and operations and to identify other objects that are solution specific. Some guidelines that might help the review process are:

1. If only one object is necessary for a service (or operation), then it operates only on that object.
2. If two or more objects are required for an operation to occur, then it is necessary to identify which object's private part should be known to the operation.
3. If an operation requires knowledge of more than one type of objects, then the operation is not functionally cohesive and should be rejected.

Class Dependencies

It is important to identify appropriate classes to represent the objects in the solution space and establish their relationships. The major relationships that are important in the context of design are:

1. Inheritance relationship
2. Containment relationship.
3. Use relationships

Organization of Class Hierarchies

This involves identification of common attributes and functions among a group of related classes and then combining them to form new class. The new class serves as the super class and the other as a subordinate classes.

Design of Class

Given below are some guidelines which should be considered while designing a class:

1. The public interface of a class should have only functions of the class.
2. An object of one class should not send a message directly to a member of another class.
3. A function should be declared public only when it is required to be used by the objects of the class.
4. Each function either accesses or modifies some data of the class it represents.

5. A class should be dependent on as few (other) classes as possible.
6. Interactions between two classes must be explicit.
7. Each subordinate class should be designed as a specialization of the base class with the sole purpose of adding additional features.
8. The top class of a structure should represent the abstract model of the target concept.

Design of Member Functions

The member functions define the operations that are performed on the object's data.

These functions behave like any other C function and therefore we can use the top-down functional decomposition technique to design them.

Design of the Driver Program

The driver program is mainly responsible for:

1. Receiving data values from the user
2 Creating objects from the class definitions
3. Arranging communication between the objects as a sequence of messages for invoking the member functions, and
4. Displaying output results in the form required by the user.

(iii) User defined data types in C++

Structure and Classes

Data types such as struct and class is user defined data type in c++

example

```
struct{
   char title[50];
  char author[50];
} book;
class student{
char title[50];
  char author[50];
public:
void get()}
student s1;
```

Enumerated data type

The enum keyword automatically enumerates a list of words by assigning them values 0,1,2 and so on enum colors_t {black, blue, green, cyan, red, purple, yellow, white};

Q.41 When does ambiguity arise in multiple inheritance? How can one resolve it? Develop a program in C++ to create a derived class having following items: name, age, rollno, marks, empcode and designation. Design a base class student having data members as rollno, marks and the another base class is employee having data members as empcode and designation. These both base classes are inherited from a single base class person with data members name and age. The program should carry out the required input( ) and output( ) member functions for all. **(12)**

Ans:

A class can be derived such that it inherits the properties of two or more base classes. This is called multiple inheritance.

Ambiguity occurs when the base classes have functions with same name. for example if two base classes have the function get_data().then which function will be used by the derived class? This problem can be solved be defining a named instance with the derived class, using the class resolution operator with the function name.

Program:

```
#include<conio.h>
#include<iostream.h>
class person{
 public:
 char name[20];
 int age;
 void get_details();
 void disp_details();
};
 void person::get_details() {
 cout<<"enter name:";
 cin>>name;
 cout<<"enter age:";
 cin>>age;
 }
 void person::disp_details() {
 cout<<"name:"<<name;
 cout<<'\n';
 cout<<"age:"<<age;
 cout<<'\n';
 }
 class student: public person {
 public:
 int roll_no;
 int marks;
 void get_details(int a,int b) {
  roll_no = a;
  marks = b;
 }
 void disp_data()
 {
  cout<<"roll_no:"<<roll_no;
  cout<<'\n';
  cout<<"marks:"<<marks; }
 };
 class employee:public person {
 public:
 int emp_code;
 char desig[20];
 void get_details(int a,char *d) {
  emp_code = a;
  strcpy(design, d); }
```

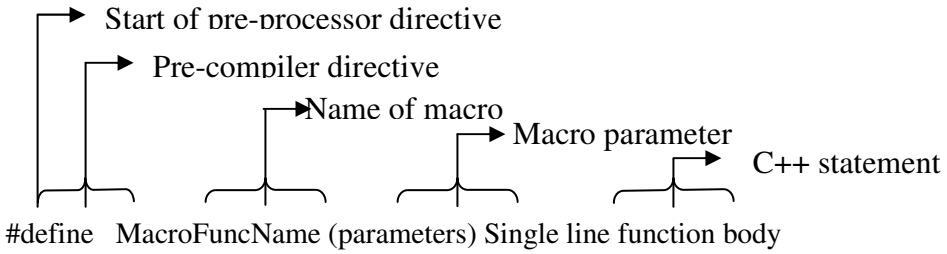
```
void disp_data() {
  cout<<"emp_code:"<<emp_code;
  cout<<'\n';
  cout<<"designation:"<<desig; }
 };
void main(){
 int r,m,c;char d[20];
 cout<<"\n Enter roll_no. and marks :";
 cin>>r,m
 cout<<"\n Enter employee code:";
 cin>>c;
 cout<<"\n Enter designation: ";
 getline(d);
 student ob1;
 employee ob2;
 ob1.get_details(r,m);
 ob2.get_details(c,d);
 ob1.disp_data();
 ob2.disp_data();getch();
}
```

Q.42 Write two advantages of using include compiler directive. **(4)**

Ans: The include compiler directive is a preprocessor directive. Means, it is a command to the preprocessor. A preprocessor is a program that performs the preliminary processing with the purpose of expanding macro code templates to produce a C++ code. By including this header file, the function that we are using is replaced by the entire function definition. This is very helpful in case of small modules which are to be called several times in a program. It reduces the overhead cost of calling the function each time.

Q.43 Explain the difference between macro & function. **(5)**

Ans:Difference between macro and function macro The preprocessor will replace all the macro function used in the program by their function body before the compilation. The distinguishing feature of macro function are that there will be no explicit function call during execution, since the body is substituted at the point of macro call during compilation. Thereby the runtime overhead for the function linking or context-switch time is reduced. The directive # define indicates the start of a macro function as shown in the figure.



Examples:
#define inc(a) a+1
#define add(a, b) (a+b)

Function:A function is a set of program statements that can be processed independer.tly. A function can be invoked which behaves as though its code is inserted at the point of the function call. The communication between a *caller* (calling function) and *callee* (called function) takes place through parameters. The functions can be designed, developed, and implemented independently by different programmers. The independent functions can be grouped to form a software library. Functions are independent because variable names and labels defined within its body are local to it.
The use of functions offers flexibility in the design, development, and implementation of the program to solve complex problems. The advantages of functions includes the following:

- Modular programming
- Reduction in the amount of work and development time
- Program and function debugging is easier
- Division of work is simplitied due to the use of divide-and-conquer principle .
- Reduction in size of the program due to code reusability
- Functions can be accessed repeatedly without redevelopment, which in turn promotes reuse of code .
- *Library of functions* can be implemented by combining well designed, tested and proven function.

Function Components
Every function has the following elements associated with it:
1. Function declaration or prototype.
2. Function parameters (formal parameters)
3. Combination of function declaration and its definition.
4. Function definition (function declaration and a function body).
5. return statement.
6. Function call.

A function can be executed using *a function call* in the program. The various components associated with the function are shown in Figure

```
Void func(int a, int b);    Function declaration
Prototype
                    parameter
……
void func(int a, int b) {   Function definition
……                                Body
}
func(x,y);                  Function Call
```

Q.44 Write a C++ program to find the sum of digits of a number reducing it to one digit. **(10)**

Ans:
```
#include<iostream.h>
void main(){
 int num,n,s,s1,limit;s=s1=0;
 cout<<"enter limit of no. of digits you want to enter";
```

```
cin>>limit;
cout<<"Enter (limit)digit number";
cin>>num;
for(int i=0;i<limit;i++)
{
        n=num%10;
                s=s+n;
        num=num/10;
}
 cout<<"sum of digits is"<<s;
 if(s>1)
 {
  while(s>0)
{
   n=s%10;
   s1=s1+n;
   s=s/10;
  }
cout<<"reduced to one digit is"<<s1;
  }
}
```

Q.45 Explain the use of break and continue statements in switch case statements. **(4)**

Ans: The switch Statement

The **switch** and **case** statements help control complex conditional and branching operations. The **switch** statement transfers control to a statement within its body.

Syntax

selection-statement :
switch (*expression* **)** *statement*
labeled-statement :
case *constant-expression* **:** *statement*
default : *statement*

Control passes to the statement whose **case** *constant-expression* matches the value of **switch (** *expression* **)**. The **switch** statement can include any number of **case** instances, but no two case constants within the same **switch** statement can have the same value. Execution of the statement body begins at the selected statement and proceeds until the end of the body or until a **break** statement transfers control out of the body.

Use of the **switch** statement usually looks something like this:

switch (*expression*)
{
declarations
.
. .
case *constant-expression* **:**

statements executed if the expression equals the

```
   value of this constant-expression
   .
   .
   .    break;
  default :
   statements executed if expression does not equal
   any case constant-expression
}
```

We can use the **break** statement to end processing of a particular case within the **switch** statement and to branch to the end of the **switch** statement. Without **break**, the program continues to the next case, executing the statements until a **break** or the end of the statement is reached. In some situations, this continuation may be desirable.

The **default** statement is executed if no **case** *constant-expression* is equal to the value of **switch** (*expression*). If the **default** statement is omitted, and no **case** match is found, none of the statements in the **switch** body are executed. There can be at most one **default** statement. The **default** statement need not come at the end; it can appear anywhere in the body of the **switch** statement. In fact it is often more efficient if it appears at the beginning of the **switch** statement. A **case** or **default** label can only appear inside a **switch** statement.

The type of **switch** *expression* and **case** *constant-expression* must be integral. The value of each **case** *constant-expression* must be unique within the statement body.

The **case** and **default** labels of the **switch** statement body are significant only in the initial test that determines where execution starts in the statement body. Switch statements can be nested. Any static variables are initialized before executing into any **switch** statements.

The following examples illustrate **switch** statements:

```
switch( c )
{
   case 'A':
      capa++;
   case 'a':
      lettera++;
   default :
      total++;
}
```

All three statements of the **switch** body in this example are executed if c is equal to 'A' since a **break** statement does not appear before the following case. Execution control is transferred to the first statement (capa++;) and continues in order through the rest of the body. If c is equal to 'a', lettera and total are incremented. Only total is incremented if c is not equal to 'A' or 'a'.

```
switch( i )
{
   case -1:
      n++;
```

```
        break;
    case 0 :
        z++;
        break;
    case 1 :
        p++;
        break;
}
```

In this example, a **break** statement follows each statement of the **switch** body. The **break** statement forces an exit from the statement body after one statement is executed. If i is equal to 1, only n is incremented. The **break** following the statement n++; causes execution control to pass out of the statement body, bypassing the remaining statements. Similarly, if i is equal to 0, only z is incremented; if i is equal to 1, only p is incremented. The final **break** statement is not strictly necessary, since control passes out of the body at the end of the compound statement, but it is included for consistency.

Q.46 What is Object Oriented Programming (OOP)? What are the advantages of Object Oriented Programming? **(6)**

Ans:

Ans: Object Oriented Programming (OOP) is an approach that provides a way of modulating programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

The **advantages** of OOP are as follows:

- Function and data both are tied together in a single unit.
- Data is not accessible by external functions as it is hidden.
- Objects may communicate with each other only through functions.
- It is easy to add new data and functions whenever required.
- It can model real world problems very well.

Q.47 Differentiate between recursion and iteration. **(4)**

Ans: Difference between recursion and iteration

Recursion and iteration are two very commonly used, powerful methods of solving complex problems, directly harnessing the power of the computer to calculate things very quickly. Both methods rely on breaking up the complex problems into smaller, simpler steps that can be solved easily, but the two methods are subtlety different. Iteration, perhaps, is the simpler of the two. **In iteration, a problem is converted into a train of steps that are finished one at a time, one after another.** For instance, if you want to add up all the whole numbers less than 5, you would start with 1 (in the 1st step), then (in step 2) add 2, then (step 3) add 3, and so on. In

each step, you add another number (which is the same number as the number of the step you are on). This is called "iterating through the problem." The only part that really changes from step to step is the number of the step, since you can figure out all the other information (like the number you need to add) from that step number. This is the key to iteration: using the step number to find all of your other information. The classic example of iteration in languages like BASIC or C++, of course, is the *for* loop.

If iteration is a bunch of steps leading to a solution, **recursion is like piling all of those steps on top of each other and then quashing them all into the solution.** Recursion is like holding a mirror up to another mirror: in each image, there is another, smaller image that's basically the same.

Example: **recursion**

```
int factorial(int number)
 {
        if (number < 0)
        {
                cout << "\nError - negative argument to factorial\n";
        exit(1);
        }
        else if (number == 0)

                return 1;

        else
                return (number * factorial(number - 1));
 }
```

Iteration :

```
int factorial(int number)
    {
        int product = 1;

        if (number < 0)
        {
                cout << "\nError - negative argument to factorial\n";
                exit(1);
        }
        else if (number == 0)
                return 1;
        else
        {
                for ( ; number > 0 ; number--)
```

```
                product *= number;
                return product;
        }

 }
```

So, the difference between iteration and recursion is that **with iteration, each step clearly leads onto the next, like stepping stones across a river**, while **in recursion, each step replicates itself at a smaller scale, so that all of them combined together eventually solve the problem.** These two basic methods are very important to understand fully, since they appear in almost every computer algorithm ever made.

Q.48 What are the different storage classes in C++. **(5)**

Ans: STORAGE CLASSES

C++ provides 4 storage class specifiers:
AUTO, REGISTER, EXTERN, and STATIC
An identifier's storage class specifier helps determine its storage class, scope, and linkage.
Storage class - determines the period during which that identifier exists in memory
Scope - determines if the identifier can be referenced in a program
Linkage - determines for a mulitple-source-file program whether an identifier is known only in the current source file or in any source file with proper declarations.

Automatic storage class

The **auto** and **register** keywords are used to declare variables of the automatic storage class. Such variables are created when the block in which they are declared is entered, they exist while the block is active, and they are destroyed when the block is exited.
Example:
auto float x, y;
declares that **float** variables **x** and **y** are local variables of automatic storage class, they exist only in the body of the function in which the definition appears.
register int counter = 1;
declares that the integer variable **counter** be placed in one of the computer's register and be initialized to 1.
Either write **auto** or **register** but not both to an identifier.

Static Storage Class

The keywords **extern** and **static** are used to declare identifiers for variables and functions of the static storage class. Such variables exist from the point at which the program begins execution.

There are two types of identifiers with static storage class: external identifiers (such as global variables and function names) and local variables declared with the storage class specifier **static**. Global variables and function names default to storage class specifier **extern**. Global variables are created by placing variable declarations outside any function definition. They retain their values throughout the execution of the program.

Q.49 Write a program to overload the unary minus operator using friend function. **(8)**

Ans:

```
#inlcude<iostream.h>
using namespace std;
class space{
 int x;
 int y;
 int z;
   public:
 void getdata(int a,int b.int c);
 void display(void);
 friend void operator-(space &s);
};
void space :: getdata(int a,int b,int c){
 x=a;
 y=b;
 z=c;
}
void space :: display(void){
 cout<<x<<" ";
 cout<<y<<" ";
 cout<<z<<"\n";
}
void  operator-( space &s){
 s.x=-s.x;
 s.y=-s.y;
 s.z=-s.z;
}
int main(){
 space s;
 s.getdata(10,-20,30);
 cout<<"S : ";
 s.display();
 -s;
 cout<<"S :";
 s.display();
 return 0;
}
```

Q.50 Write a program in which a class has three data members: name, roll no, marks of 5 subjects and a member function Assign() to assign the streams on the basis of table given below: **(9)**

Avg. Marks	**Stream**
90% or more	Computers
80% - 89%	Electronics
75% - 79%	Mechanical
70% - 74%	Electrical

Ans:

```
#include<iostream.h>
class student
{
 char name[10];
 int rollno,marks[5],sum,avg;
public:
 void assign()
 {
   cout<<"enter name of the student";
 cin>>name;
 cout<<"enter rollno";
 cin>>rollno;
 cout<<"enter marks of five subjects";
 for(int i=0;i<5;i++)
 {
          cin>>marks[i];
 }
 sum=avg=0;
 for(int j=0;j<5;j++)
 {
          sum=sum+marks[j];
 }
 //cout<<sum;
 avg=sum/5;
 cout<<"Average marks is"<<avg;
 cout<<endl;
 if(avg>=90)
 {
          cout<<"stream is Computer"<<endl;
 }
          else
          {
                  if(avg>=80 && avg<=89)
                  {
                          cout<<"stream is Electronics";
```

```
        }
                if(avg>=75 && avg<=79)
                {
                        cout<<"stream is Mechanical";
                }
                if(avg>=70 && avg<=74)
                {
                        cout<<"stream is Electrical";
                }
        }
 }
 };

 void main()
 {
        student s1;
        s1.assign();
 }
```

Q.51 Write a program to convert the polar co-ordinates into rectangular coordinates. (hint: polar co-ordinates(radius, angle) and rectangular co-ordinates(x,y) where x = r*cos(angle) and y = r*sin(angle)). **(8)**

Ans:

```
#include<iostream.h>
#include<math.h>
using namespace std;
class coord
{
 float radius,angle,x,y;
 public:
    polar(float r, float a);
    void rectangular(float x, float y);
    void display();
}
void coord :: polar(float r,float a)
{
 cout<<"\n Enter the radius and angle : ";
 cin>>radius>>angle;

}
void coord :: rectangular(float x,float y)
{
 x=radius*cos(angle);
 y=radius*sin(angle);
}
```

```
void display()
{
 cout<<"\n The rectangular coordinates are : "<<x<<" and "<<y;
}
void main()
{
 coord obj;
 obj.polar();
 obj.rectangular();
 obj.display();
}
```

Q.52 Is the following code correct? Justify your answer

```
int intvar = 333;
int * intptr;
cout << *intptr;
```

(3)

Ans: The following code is not correct as it will display nothing the given code is

```
 int intvar=333;
 int *intptr;
cout<<*intptr;
```

since the pointer variable is not assigned any address so it will display nothing. But if we add one more line that is intptr=&intvar; then it will display 333.

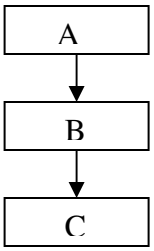
Q.53 What is multilevel inheritance? How is it different from multiple inheritance? **(5)**

Ans: Multilevel Inheritance

We can built hierarchies that contain as many layers of inheritance as we like. For example, given three classes called **A**, **B** and **C**, **C** can be derived from **B**, which can be derived from **A**. When this type of situation occurs, each derived class inherits all of the traits found in all of its base classes. In this case, **C** inherits all aspects of **B** and **A**.

```
class A{…….};   //Base class
class B: public A{…..}; // B derived from A
class C:public B{……}; // C derived from B
```

This process can be extended to any numbers of levels



Base

Intermediate Base Class

Derived class

Multiple Inheritance.
When the number of base classes Inherited exceeds one then the type of inheritance is known as **Multiple Inheritance**.
For example in C++, The syntax:

```
Class D: public A, public B
{
 //code
}
```

Indicates that **D** inherits the attributes of the two classes **A** and **B**.

Though Multiple Inheritance sounds exciting and very useful it has its own drawbacks, owing to which most OOP languages deny support for this feature. These drawbacks include complexities involved when inheriting from many classes and the difficulty in managing such codes.

Q.54 Write a C++ program to prepare the mark sheet of an university examination with the following items from the keyboard :

Name of the student
Roll no.
Subject name
Subject code
Internal marks
External marks
Design a base class consisting of data members Name of the student and Roll no. The derived class consists of the data members Subject name, Subject code, Internal marks and External marks. **(9)**

Ans:

```
#include<iostream.h>
class student
{
 char name[10];
 int rollno;
public:
 void get()
 {
        cout<<"enter student's name"<<endl;
        cin>>name;
        cout<<"enter student's rollno"<<endl;
        cin>>rollno;
 }
 void disp()
 {
```

```
 cout<<"Name of the student is : "<<name<<endl;
            cout<<"Rollno is : "<<rollno<<endl;
 }
};

class marks:public student
{
 char subjectname[15];
 int subcode,intmarks,extmarks;

public:
 void get()
 {
            student::get();
            cout<<"enter subject name";
            cin>>subjectname;
            cout<<"enter subject code";
            cin>>subcode;
            cout<<"enter Internal marks";
            cin>>intmarks;
            cout<<"enter External marks";
            cin>>extmarks;
 }
 void disp()
 {
            student::disp();
            cout<<"Subject name is : "<<subjectname<<endl;
            cout<<"Subject code is : "<<subcode<<endl;
            cout<<"Internal marks : "<<intmarks<<endl;
            cout<<"External marks : "<<extmarks<<endl;
 }
};

void main()
{
 marks m;
 m.get();
 m.disp();
}
```

Q.55 Explain the following:
(i) Non public constructors
(ii) Inline function
(iii) Virtual functions
(iv) Types of Inheritance. **(4x4=16)**

Ans:

i.) **Non-public constructors:** If a constructor is protected or private, it is called non-public constructor. We cannot create objects of a class that have non-public constructors. Only member function and friend function can create objects in this case.

ii.) **Inline functions**: A function which is expanded in line is called an inline function. Whenever we write there are certain costs associated with it such as jumping to the function, saving registers, pushing arguments into the stack and returning to the calling function. To remove this overhead we write an inline function. It is a sort of request from the compiler. If we declare and define a function within the class definition then it is automatically inline.

Advantage of inline functions:

- They have no overhead associated with function cal or return mechanism

Disadvantage of inline functions:

- If the function made inline is too large and called regularly our program grows larger.

There are certain restrictions in which the compiler may apply like:- some compilers will not inline a function if it contains a static variable or if the function contains a loop, a switch , etc..

iii.) **Virtual function:** virtual functions are important because they support polymorphism at run time. A virtual function is a member function that is declared within the base class but redefined inside the derived class. To create a virtual function we precede the function declaration with the keyword virtual. Thus we can assume the model of "one interface multiple method". The virtual function within the base class defines the form of interface to that function. It happens when a virtual function is called through a pointer.

The following is an example that illustrates the use of a virtual function:

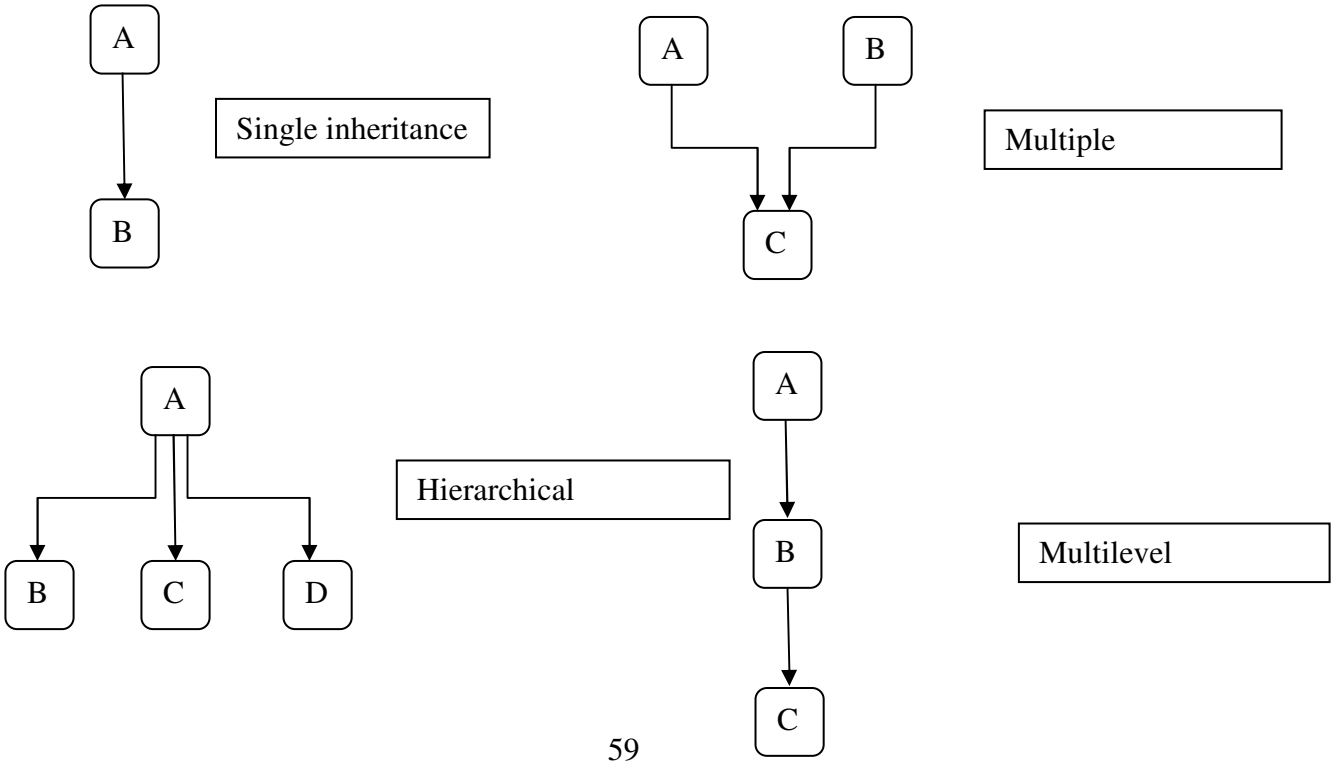
```
#include<iostream.h>
using namespace std;
class base
{
```

```
  public:
 int i;
 base(int x)
 {
            i=x;
 }
 virtual void func()
 {
            cout<<"\n using base version of function";
            cout<<i<<endl;
 }
};
class derived: public base
{
  public:
 derived(int x):base(x){}
 void func()
 {
            cout<<"\n sing derived version";
            cout<<i*i<<endl;
 }
};
class derived1: public base
{
  public:
 derived1(int x): base(x){}
 void func()
 {
            cout<<"\n using derived1 version";
            cout<<i+i<<endl;
 }
```

```
};
int main()
{
 base *p;
 base ob(10);
 derived d_ob(10);
 derived1 d_ob1(10);
 p=&ob;
 p->func();
 p=&d_ob;
 p->func();
 p=&d_ob1;
 p->func();
 return 0;
}
```

iv.) **Inheritance:** the mechanism of deriving a new class from an old one is called inheritance. The old class is called the base class and the new one is known as the derived class or sub class.
Types of inheritance

a. A derived class with only one base class is called single inheritance
b. A derived class with several base classes is called multiple inheritance.
c. When one base class is inherited by more than one class is called hierarchical inheritance.
d. When one derived class inherits another derived class, it is called multilevel inheritance.



Q.56 Write a program that shows the use of read() and write() to handle file I/O involving objects. **(8)**

Ans:

```
#include<iostream.h>
#include<fstream.h>
#include<iomanip.h>
class INVENTORY
{
 char name[10];
 int code;
 float cost;
    public:
 void readdata(void);
 void writedata(void);
};
void INVENTORY :: readdata(void)
{
 cout<<"\n Enter name:";cin>>name;
 cout<<"\n Enter code:";cin>>code;
 cout<<"\n Enter cost:";cin>>cost;
}
void INVENTORY :: writedata(void)
{
 cout<<setiosflags(ios::left)
     <<setw(10)<<name
     <<setiosflags(ios::right)
     <<setw(10<<code)
     <<setprecision(2)
     <<setw(10)<<cost<<endl;
}
```

```
int main()
{
 INVENTORY item[3];
 fstream file;
 file.open("Stock.dat",ios::in|ios::out);
 cout<<"\n Enter details for three  items \n";
 for(int i=0;i<3;i++)
 {
        item[i].readdata();
        file.write((char *) & item[i],sizeof(item[i]));
 }
 file.seekg(0);
 cout<<"\nOutput\n";
 for(i=0;i<3;i++)
 {
        file.readdata((char *) &item[i],sizeof(item[i]));
        item[i].writedata();
 }
 file.close();
 return 0;
}
```

Q.57 Define a function area() to compute the area of objects of different classes – triangle, rectangle, square. Invoke these in the main program. Comment on the binding (static or dynamic) that takes place in your program. **(10)**

Ans:

```
#include <iostream.h>    //  For input/output
class triangle
{
float base, height, areat;
public:
 void area()
 {
  cout << "Enter the base of the triangle: ";
  cin  >> base;
```

```
  cout << endl;   // Go to a new line
   cout << "Enter the height of the triangle: ";
   cin  >> height;
   cout << endl;  //  Go to a new line

   areat = 0.5 * base * height;

    cout << "\n\n\n\n\n\n\n\n\n\n";    //  What is this line doing?

   cout << base   << " is the base length of the triangle. \n\n";
   cout << height << " is the height of the triangle. \n\n";
   cout << areat   << " is the area of the triangle. \n\n";
  }
  };
class rectangle
{
float breadth, length, arear;
public:
  void area()
  {
   cout << "Enter the breadth of the rectangle: ";
   cin  >> breadth;
   cout << endl;   // Go to a new line
   cout << "Enter the length of the rectangle: ";
   cin  >> length;
   cout << endl;  //  Go to a new line

   arear = breadth * length;

    cout << "\n\n\n\n\n\n\n\n\n\n";    //  What is this line doing?

   cout << breadth   << " is the length of the rectangle. \n\n";
   cout << length << " is the height of the rectangle. \n\n";
   cout << arear   << " is the area of the rectangle. \n\n";
  }
};

class square
{
float side,areas;
public:
  void area()
  {

   cout << "Enter the height of the square: ";
   cin  >> side;
```

```
  cout << endl;  //  Go to a new line

  areas = side * side;

   cout << "\n\n\n\n\n\n\n\n\n\n";    //  What is this line doing?

     cout << side << " is the side of the square:. \n\n";
  cout << areas   << " is the area of the square:. \n\n";
 }
};

main()
{
triangle t;
rectangle r;
square s;
t.area();
r.area();
s.area();
  return(0);
}
```

Q.58 What are the basic differences between manipulators and ios member functions in implementation? Give examples. **(8)**

Ans:

The basic difference between manipulators and ios member functions in implementation is as follows:

The ios member function returns the previous format state which can be of use later on but the manipulator does not return the previous format state. Whenever we need to save the old format states, we must use the ios member functions rather then the manipulators. Example: an example program illustrating the formatting of the output values using both manipulators and ios functions is given below.

```
#include<iostream.h>
#include<iomanip.h>
using namespace std;
int main()
{
 cout.setf(ios::showpoint);
 cout<<setw(5)<<"\n"
    <<setw(15)<<"Inverse_of_n"
    <<setw(15)<<"Sum_of_terms\n\n";
```

```
        double term,sum=0;
        for(int n=1;n<=10;n++)
        {
                term=1.0/float(n);
                sum+=term;
                cout<<setw(5)<<n
                   <<setw(14)<<setprecision(4)
                   <<setiosflags(ios::scientific)<<term
                   <<setw(13)<<resetiosflags(ios::scientific)
                   <<sum<<endl;
        }
        return 0;
}
```

Q.59 Class Y has been derived from class X. The class Y does not contain any data members of its own. Does the class Y require constructor. If yes, why? **(4)**

Ans: Yes, it is mandatory for class Y to have a constructor. It is the responsibility of the derived class constructor to invoke the base class constructor by passing required arguments to it.Unless the constructor of class Y is invoked the data members of class X that have been inherited by class Y will not be given any values. Although class Y does not have any data member of its own, yet to use the data members inherited from X, Y needs a constructor.

Y(type1 x, type2 y,…):

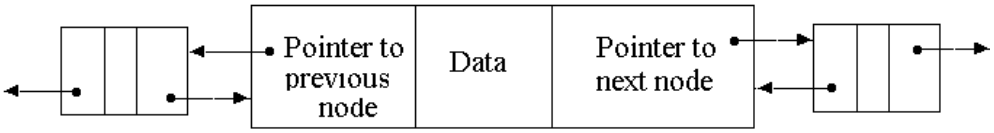
Q.60 Write a program using function template to find the cube of a given integer, float and double number. **(6)**

Ans:

```
//Using a function template

#include <iostream.h>

template < class T >

T cube( T value1)
{
```

```
  return value1*value1*value1;

  }
int main()
{
 int int1;

 cout <<"Input integer value: ";
 cin  >> int1;
 cout << "The cube of integer value is: "<< cube( int1);
 double double1;
 cout << "\nInput double value; ";
 cin  >> double1;
 cout << "The cube of double value is: "<< cube( double1);
 float float1;
 cout     << "\nInput float value";
 cin      >> float1;
 cout     << "The cube of float value is: "<< cube(float1);
cout<< endl;
 return 0;
}
```

Q.61 Write a C++ program to insert and delete a node from the double linked list. The list must be arranged in ascending order. Structure of node is **(14)**



Ans:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
struct Node{
int data;
Node *next;
}*start;
void Create();
void AddItem();
void DeleteItem();
```

```
void Display();

main(){
int ch=0;
Create();
clrscr();
while(ch!=4){

cout<<"\n1. Add  item";
cout<<"\n2. Delete item";
cout<<"\n3. Display";
cout<<"\n4. Quit";
cout<<"\n Enter choice";
cin>>ch;
switch(ch){
case 1:
AddItem();
break;
case 2:
DeleteItem();
break;
case 3:
Display();
break;
}
}
}

void Create(){
Node *h=new Node;
h->data=9999;
h->next=NULL;
start=h;
}

void AddItem(){
Node *p=new Node;
Node *loc, *old;
cout<<"Enter the Item :";
cin>>p->data;
old=start;
loc=start->next;
while(loc!=NULL){
 if(loc->data>p->data){
            p->next=old->next;
            old->next=p;
```

```
            return;
            }else{
                    old=loc;
                    loc=loc->next;
                    }
 }//End while\
//Insert at end
p->next=NULL;
old->next=p;
}

void DeleteItem(){
 Node *p;
 Node *loc,*old;
 int item;
 if(start->next==NULL){
 cout<<"\nList is empty";
 return;
 }
 cout<<"\nEnter the item to delete :";
 cin>>item;
 old=start;
 loc=start->next;
 while(loc!=NULL){
 if(loc->data==item){
            old->next=loc->next;
            cout<<"\n"<<loc->data<<" is deleted";
            delete loc;
            return;
            }else{
            old=loc;
            loc=loc->next;
            }
 }
 cout<<"\n Item not found";
}

void Display(){
Node *p=start;
while(p!=NULL){
cout<<"\n"<<p->data;
p=p->next;
}
}
```

Q.62 How can a common friend function to two different classes be declared? **(4)**

Ans:

Friend functions are not considered class members; they are normal external functions that are given special access privileges. Friends are not in the classs scope, and they are not called using the member-selection operators (**.** and **>**) unless they are members of another class.

Any data which is declared **private** inside a class is not accessible from outside the class. A function which is not a member or an external class can never access such private data. But there may be some cases, where a programmer will need need access to the private data from non-member functions and external classes. C++ offers some exceptions in such cases.

A class can allow non-member functions and other classes to access its own private data, by making them as **friends**. This part of C++ tutorial essentially gives two important points.

Once a non-member function is declared as a friend, it can access the private data of the class similarly when a class is declared as a friend, the friend class can have access to the private data of the class which made this a friend.

Q.63 Write a C++ program that copies the contents of a text file (i.e. any CPP file) to another file. Invoke the program with two command line arguments-the source file and the destination file – like this. **(10)**

C> ccopy scfile.cpp dstfile.cpp

Ans:

```
#include<iostream.h>
#include<fstream.h>
#include<process.h>
void main(int argc,char *argv[])
{
 if(argc!=3)
 {                  cout<<"\n argument is not passed";
          exit(-1);
 }
 char ch;
 ifstream inf;
 ofstream otf;
 inf.open(argv[1]);
 otf.open(argv[2]);
 if(!inf)
 {                  cout<<"\n can't open"<<argv[1];
          exit(-1);
 }
 while(inf)
 {                  inf.get(ch);
          otf<<ch;
}}
```

Q.64 Explain any three drawbacks of procedure oriented languages. Give an example each in C language to highlight the drawback. **(6)**

Ans: Drawbacks of procedure oriented languages.

Previously, programs were written in a procedural fashion i.e. the statements were written in the form of a batch. But as the requirements grew, it was seen that the programs were getting larger and larger and it became difficult to debug. So functions were introduced to reduce the size of the programs and improve readability in them. Still that was not enough.

One of the major problems with the "Procedural Paradigm" was that data was treated as a stepson and functions were given more priority. Where as, it is the other way. In this procedure the original data could easily get corrupted, as it was accessible to all the functions, even to those which do not have any right to access them. Before OOP, the programmer was restricted to use the predefined data types such as integer, float and character. If any program required handling of the x-y coordinates of some point then it is quite a headache for the programmer. Where as, in OOP this can be handled very easily as the programmer can define his own data types and the corresponding functions.

Q.65 Write a C++ program which reads in a line of text and counts the number of occurrences of the word 'the' in it. It outputs the line of text on one line, 'The number of times *the* appears is' and the count on the next line using a single cout statement. **(8)**

Ans:

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

void main()
{
clrscr();
int countch=0;
int countwd=1;
cout << "Enter your sentence in lowercase: " << endl;
char ch='a';
while(ch!='\r')
{
ch=getche();
if(ch==' ')
countwd++;
else
countch++;
}
cout << "\n Words = " << countwd << endl;
cout << "Characters = " << countch-1 << endl;
getch();
}
```

This program takes in a sentence as a screen input from the user.

Q.66 Explain the difference between constructor and copy constructor in C++. Which of these is invoked in the following statement
Date D1 (D2); where D2 is also an object of class Date. **(3)**

Ans: *Constructors* are special `member functions' of which exactly one is called automatically at creation time of an object. Which one depends on the form of the variable declaration. Symmetrically, there exists a destructor, which is automatically called at destruction time of an object. Three types of constructors are distinguished: default constructor, copy constructor, and all others (user defined). A constructor has the same name as the class and it has no return type. The parameter list for the default and the copy constructor are fixed. The user-defined constructors can have arbitrary parameter lists following C++ overloading rules.

```
class A {
   int i; // private
public:
   A();            // default constructor
   A( const A& a);  // copy constructor
   A( int n);       // user defined
   ~A();           // destructor
};
int main() {
   A a1;  // calls the default constructor
   A a2 = a1;  // calls the copy constructor (not the assignment operator)
   A a3(a1);   // calls the copy constructor (usual constructor call syntax)
   A a4(1);    // calls the user defined constructor
}  // automatic destructor calls for a4, a3, a2, a1 at the end of the block
```

The compiler generates a missing default constructor, copy constructor, or destructor automatically. The default implementation of the default constructor calls the default constructors of all class member variables. The default constructor is **not** automatically generated if other constructors are explicitly declared in the class (except an explicit declared copy constructor). The default copy constructor calls the copy constructor of all class member variables, which performs a bitwise copy for built-in data types. The default destructor does nothing. All default implementations can be explicitly inhibited by declaring the respective constructor/destructor in the private part of the class.
Constructors initialize member variables. A new syntax, which resembles constructor calls, allows to call the respective constructors of the member variables (instead of assigning new values). The constructor call syntax extends also to built-in types. The order of the initializations should follow the order of the declarations, multiple initializations are separated by comma.

```
class A {
   int i; // private
public:
   A() : i(0) {}              // default constructor
A( const A& a) : i(a.i) {}  // copy constructor, equal to the compiler default
   A( int n) : i(n) {}        // user defined
```

```
   ~A() {}                    // destructor, equal to the compiler default
};
```

Usually, only the default constructor (if the semantics are reasonable), and some user defined constructors are defined for a class. As soon as the class manages some external resources, e.g., dynamically allocated memory, the following four implementations have to work together to avoid resource allocation errors: default constructor, copy constructor, assignment operator, and destructor. Note that the compiler would create a default implementation for the assignment operator if it is not defined explicitly

Q.67 Define a class Rectangle which has a length and a breadth. Define the constructors and the destructor and member functions to get the length and the breadth. Write a global function which creates an instance of the class Rectangle and computes the area using the member functions. **(8)**

Ans:

```
#include <iostream.h>
class CRectangle {
   int width, height;
  public:
    CRectangle (int,int);
    ~CRectangle ();
    friend int area(CRectangle);
};

CRectangle::CRectangle (int a, int b) {

  width = a;
  height = b;
}
CRectangle::~CRectangle () {
// delete width;
  //delete height;
}
int area(CRectangle r)
{
 return r.height * r.width;
}
int main () {
 int h,w;
 cout<<"enter length and breadth of rectangle";
 cin>>h>>w;
  CRectangle rect (h,w);
  cout << "area: " <<area(rect)<< endl;
  return 0;
}
```

Q.68 Why is destructor function required in class? What are the special characteristics of destructors? Can a destructor accept arguments? **(8)**

Ans: Destructor: A destructor is used to destroy the objects that have been created by a constructor. It has the same name as that of the class but is preceded by a tilde. For example, ~integer () { }

Characteristics:

- A destructor is invoked implicitly by the compiler when we exit from the program or block. It releases memory space for future use.
- A destructor never accepts any argument nor does it return any value.
- It is not possible to take the address of a destructor.
- Destructors can not be inherited.
- They cannot be overloaded

For example, the destructor for the matrix class will defined as follows:

```
matrix :: ~matrix()
{
 for(int i=0;i<d1;i++)
 delete p[i];
 delete p;
}
```

Q.69 Define a class Deposit which has a principal, a rate of interest which is fixed for all deposits and a period in terms of years. Write member functions to
(i) alter(float) which can change the rate of interest.
(ii) interest() which computes the interest and returns it.
(iii) print() which prints data as shown below. Note that the width of each column is 20. **(10)**

Principal	Year	Interest
1100.00	1	100.00

Ans:

```
#include<iostream.h>

class deposit
```

```
{
 float p,r,t,i;
public:
float interest()
{
cout << "Enter the principal, rate & time : " << endl;
cin>>p>>r>>t;
i=(p*r*t)/100;
return i;
}
 void print()
 {
cout << "Principal = Rs" << p << endl;
cout << "Rate = " << r << "%" << endl;
cout << "Time = " << t << " years" << endl;
cout << "Simple Interest = Rs" << i << endl;
}
 };
void main()
{
 deposit d;
 d.interest();
 d.print();
}
```

Q.70 Differentiate and give examples to bring out the difference between

(i) private and public inheritance.
(ii) instantiation and specialization of a template class.
(iii) static and dynamic binding.
(i) a class and a struct. **(3.5 x 4)**

Ans:

i) private and public inheritance

In private inheritence the public members of the base class become private members of the derived class. Therefore the object of derived class cannot have access to the public member function of base class.

In public inheritance the derived class inherit all the public members of the base class and retains their visibility. Thus the public member of the base class is also the public member of the derived class.

```
Class a
{
int x;
public:
```

```
void disp()
{
}
};
class y : private a
{
int c;
public:
void disp()
{
}
};
so when d.disp() // disp() is private;
```

ii) instantiation and specialization of template class

A template specialization allows a template to make specific implementations when the pattern is of a determined type. For example, suppose that our class template **pair** included a function to return the result of the module operation between the objects contained in it, but we only want it to work when the contained type is **int**. For the rest of the types we want this function to return **0**. This can be done the following way:

```
// Template specialization
#include <iostream.h>
template <class T>
class pair {
   T value1, value2;
  public:
   pair (T first, T second)
     {value1=first; value2=second;}
   T module () {return 0;}
};
template <>
class pair <int> {
   int value1, value2;
  public:
   pair (int first, int second)
     {value1=first; value2=second;}
   int module ();
};
template <>
int pair<int>::module() {
  return value1%value2;
}
int main () {
  pair <int> myints (100,75);
  pair <float> myfloats (100.0,75.0);
  cout << myints.module() << '\n';
```

```
  cout << myfloats.module() << '\n';
  return 0;
}
```

the process of creating a specific classs from a class template is called instantiation. The compiler will perform the error analysis only when an instantiation takes place.

```
// class templates
#include <iostream.h>
template <class T>
class pair {
   T value1, value2;
  public:
    pair (T first, T second)
      {value1=first; value2=second;}
    T getmax ();
};
template <class T>
T pair<T>::getmax ()
{
  T retval;
  retval = value1>value2? value1 : value2;
  return retval;
}
int main () {
  pair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

iii) Static and Dynamic binding

The information is known to the compiler at the compile time and, therefore compiler is able to select the appropriate function for a particular call at the compile time itself. This is called early binding or static binding.

The linking of function with a class much later after the compilation, this process is termed as late binding or dynamic binding because the selection of the appropriate function is done dynamically at run time. This requires the use of pointers to objects.

iv)a class and a struct

A class is a logical method to organize data and functions in the same structure. They are declared using keyword class, whose functionality is similar to that of the C keyword struct, but with the possibility of including functions as members, instead of only data.

Its form is:

```
class class_name {
  permission_label_1:
    member1;
  permission_label_2:
    member2;
```

... } object_name;

where ***class_name*** is a name for the class (user defined *type*) and the optional field ***object_name*** is one, or several, valid object identifiers. The body of the declaration can contain ***members***, that can be either data or function declarations, and optionally ***permission labels***, that can be any of these three keywords:

private:, **public:** or **protected:**

In struct all members declared are public by default.

Struct student{ char name[10]; int age;}stud;

Both struct and class is used as user defined data type.

Q.71 Explain the order in which constructors are invoked when there is multiple inheritance. **(4)**

Ans: The order in which constructor is invoked when there is multiple inheritance is order in which the class is derived that is constructor of base class will invoke first and then the constructor of derived class and so on. The destructor will invoke in reverse order than constructor. That is destructor of derived class will invoke first and then base class. Example

```
Class a
{
a()
{
cout<<"constructor a";
}
~a()
{
cout<<"destructor a";
}
};
class b : public a
{
b()
{
cout<<"constructor b";
}


~b()
{
cout<<"destructor b";
}
};
void main()
{
b b1;
}
```

output
constructor a
constructor b
destructor b
destructor a

Q.72 Define a class Publication which has a title. Derive two classes from it – a class Book which has an accession number and a class Magazine which has volume number. With these two as bases, derive the class Journal. Define a function print() in each of these classes. Ensure that the derived class function always invokes the base(s) class function. In main() create a Journal called IEEEOOP with an accession number 681.3 and a volume number 1. Invoke the print() function for this object. **(10)**

Ans:

```
#include<iostream.h>
class publication
{
 char title[40];
 float price;
public:
 void getdata()
 {
            cout<<"Enter publication:";
                    cin>>title;
            cout<<"Enter price";
            cin>>price;
 }
 void print()
 {
            cout<<"\tPublication ="<<title<<endl;
            cout<<"\tPrice="<<price<<endl;
 }
};
 class book:public publication
 {
            int accessionno;
 public:
     void getdata()
  {
              publication::getdata();
              cout<<"Enter Accession number";
              cin>>accessionno;
  }
```

```
  void print()
  {
            publication::print();
            cout<<"Accession number is"<<accessionno;
  }
};
class magazine:public publication
{
          int volumeno;
public:
    void getdata()
  {
            publication::getdata();
            cout<<"Enter Volume number of magazine";
            cin>>volumeno;
  }
  void print()
  {
            publication::print();
            cout<<"Volume number is"<<volumeno<<endl;
  }
};
class journal:public book,public magazine
{
          char name[20];
public:
          void getdata()
          {

                    book::getdata();
                    magazine::getdata();
                    cout<<"enter name of journal";
                    cin>>name;
          }
          void print()
          {
                    cout<<"name of journal is"<<name<<endl;
                    book::print();
                    magazine::print();
          }
};
void main()
{
          journal j;
          j.getdata();
          j.print();
}
```

Q.73 With an example highlight the benefit of operator overloading. **(2)**

Ans:
The operator facilitates overloading of the c++ operators. The c++ operator overloaded to operate on member of its class.

Q.74 Explain the difference between operator member functions and operator non member functions. **(2)**

Ans:
The operator overloaded in a class is known as operator member function. Using friend functions in operator overloading then it becomes operator non member function.

Q.75 Define a class Date. Overload the operators += and <<. **(10)**

Ans:
Date class

```
#include<iostream.h>
#include<conio.h>
class time
    {
     private:  int hr,min,sec;

     public:
            void getdata();
 void operator +=(time a1);
 friend ostream & operator << (ostream &out,time &t)
            {
                    out<<t.hr<<":";
                            out<<t.min<<":";
                            out<<t.sec;
                    return out;
            }
   };

 void time::getdata()
    {
 cout<<"ENTER THE HOURS"<<endl;
 cin>>hr;
 cout<<"ENTER THE MINUTES"<<endl;
 cin>>min;
 cout<<"ENTER THE SECONDS"<<endl;
 cin>>sec;
    }
```

```
void time::operator +=(time a1)
    {

 hr=hr+a1.min;
 min=min+a1.min;
 sec=sec+a1.sec;
 if(sec>=60)
 {
          min++;
          sec=sec-60;
 }
 if(min>=60)
 {
          hr++;
          min-=60;
 }

 }


 void main()
   {
    clrscr();
    time t1,t2;
    t1.getdata();
    t2.getdata();
    t1+=t2;
    cout<<"\n RESULT OF ADDITION"<<endl;
    cout<<t1;
    getch();
   }
```

Q.76 What are abstract classes? How do you define one and how is it useful in a typical library? **(4)**

Ans: A class that contains at least one pure virtual function is considered an abstract class. Classes derived from the abstract class must implement the pure virtual function or they, too, are abstract classes. class Account

```
{
public:
            Account( double d );  // Constructor.
   virtual double GetBalance();         // Obtain balance.
   virtual void   PrintBalance() = 0;   // Pure virtual function.
```

```
private:
   double _balance;
};
```

Abstract classes act as expressions of general concepts from which more specific classes can be derived. You cannot create an object of an abstract class type; however, you can use pointers and references to abstract class types.

Q.77 Define a class City which has name and pincode. Define another class Address which has house no and city of type City. Define the constructor, the destructor and a function print() which prints the address. Create an object of type Address in main() and print it. **(10)**

Ans:

```
#include<iostream.h>
class city
{
public:
 char *name;
 int pin;
public:
 city()
 {
 }
 city(char nm[],int pn)
 {
     name=nm;
     pin=pn;
 }
};
class address
{
public:
 city houseno,cty;
 address(char ct[],int hn)
 {
            cty.name=ct;
            houseno.pin=hn;
 }

   void print()
 {
    cout<<"city name is"<<cty.name;
  cout<<"pincode is"<<houseno.pin;
 }
};
```

```
void main()
{
 address ad("delhi",23),ad1("Rohini",85);
ad.print();
ad1.print();
}
```

Q.78 a. Define a class template Queue with put() and get() operations. Using this create a Queue of integers in main() and add two elements to the queue. **(7)**

b. Implement exception handling for handling both when the queue is empty and when the queue is full. **(7)**

Ans:

```
#include<iostream.h>
const int max=3;
template<class type>
class queue
{
 type qu[max];
 int head,tail,count;
 public:
        class full
        {
        };
        class empty
        {
        };
        queue()
        {
                head=-1;
                tail=-1;
                count=0;
        }
        void put(type var)
        {
                if(count>=max)
                        throw full();
                qu[++tail]=var;
                ++count;
                if(tail>=max-1)
                        tail=-1;
        }
```

```
        type get()
        {
                if(count<=0)
                        throw empty();
                type temp=qu[++head];
                --count;
                if(head>=max-1)
                        head=-1;
                return temp;
        }
};
int main()
{
 queue<float> q1;
 float data;
 char choice='p';
 do
 {
        try
        {
                cout<<"\enter 'x' to exit, 'p'for put,'g'for get:";
                cin>>choice;
                if(choice=='p')
                {
                        cout<<"Enter data value:";
                        cin>>data;
                        q1.put(data);
                }
                if(choice=='g')
                        cout<<"Data="<<q1.get()<<endl;
        }
        catch(queue<float>::full)
        {
                cout<<"Error:queue is full."<<endl;
        }
        catch(queue<float>::empty)
        {
                cout<<"Error: queue is empty"<<endl;
        }
 }while(choice!='x');
 return 0;
}
```

Q.79 Describe the different modes in which files can be opened in C++. **(4)**

Ans: Different modes in which files can be opened in C++.

ios::in	Open file for reading
ios::out	Open file for writing
ios::ate	Initial position: end of file
ios::app	Every output is appended at the end of file
ios::trunc	If the file already existed it is erased
ios::binary	Binary mode

Q.80 Define a class Car which has model and cost as data members. Write functions
(i) to read the model and cost of a car from the keyboard and store it a file CARS.
(ii) to read from the file CARS and display it on the screen. **(10)**

Ans:

```
#include<iostream.h>
#include<fstream.h>
class car
{
 char model[10];
 float cost;
public:
 void read()
 {
            cout<<"\n Enter model of car";
            cin>>model;
            cout<<"\n Enter cost of car";
            cin>>cost;
                    ofstream of("car.dat",ios::app);
        of<<model;
             of<<cost;
             of.close();        }
 void read_file()
 {
            ifstream ifs("car.dat");
            while(ifs)
            {                               ifs>>model>>cost;

            }
            ifs.close();
            cout<<"Model"<<model<<endl;
```

```
            cout<<"Cost"<<cost;
 }};
void main()
{           car c;
 c.read();
 c.read_file();}
```

Q.81 What is the output of the following program segment?

```
float pi = 3.14167234; int i=1, j=2;
cout.fill('$'); cout.ios::precision(5); cout.ios::width(10);
cout<<i*j*pi<<'\n';
```

(2)

Ans: The output is $$$$6.2833.

Q.82 Given the following definition

```
class X { public: int a; };
class Y1 : public X { };
class Y2 : protected X { };
class Y3 : private X { };
```

Point out the errors

```
void f(Y1* py1, Y2* py2, Y3* py3)
{
  X* px = py1;
  py1->a = 7;
  px = py2;
  py2->a = 7;
  px = py3;
  py3->a = 7;
}
```

(2)

Ans:

There ia an error class y3 derived privately from X so it cannot access the public or private members;

Q.83 Given

```
class Confusion { };
```

What is the difference between Confusion C; and Confusion C( ); **(2)**

Ans:

The difference between Confusion C; and Confusion C() is that Confusion C() invoke constructor with an argument whereas Confusion C invoke default constructor;

Q.84 What are the shortcomings of procedure oriented approach? **(8)**

Ans: Programming, using languages such as COBOL, FORTRAN and C is known as procedure oriented programming (POP).The problem with POP is that in this approach things are done in a sequential manner. A number of functions are written for this. The drawbacks of POP are as follows:

i. In POP groups of instructions are written which are executed by the compiler in a serial manner.
ii. POP contains many functions, and important data items are made global so that they are easily accessible by all the functions. Global data are more prone to undesirable changes which can be accidently made by a function
iii. Another drawback is that it "does not model real world problems" very well.
iv. In POP, data moves openly around the system from one function to another.

Functions can transform data into different forms.

Q.85 Given

```
const char *s = "This";
char * const ptr = "That";
```

Point out the errors, if any in the following statements

```
*s = 'A';
*ptr = 'A';
ptr++;
s++;
```

(2)

Ans: the error is l-value specifies const object

Q.86 The output of the following piece of code is

```
for (x=6,y=3; x && x!=y; x--)
        y++;
cout<< "x = "<< x << "y = "<< y;
```

(2)

Ans:

The output is z=3,y=3

Q.87 Write expressions to represent the following:

(i) p is a function whose argument is a pointer to an array of characters and which returns a pointer to an integer.

(ii) p is a function whose argument is a pointer to character and which returns a pointer to an array of ten integers. **(2)**

Ans:

i) int *p(char **arr[])

ii) int *p(char *ch)

Q88 What is encapsulation? What are its advantages? How can encapsulation be enforced in C++? **(6)**

Ans: Encapsulation: The wrapping up of data and functions into a single unit is called encapsulation. The data can only be accessed by the function with which they have been tied up and not by the outside world. It creates an interface between the object's data and the program.
A common use of information hiding is to hide the physical storage layout for data so that if it is changed, the change is restricted to a small subset of the total program. For example, if a three-dimensional point (x, y, z) is represented in a program with three floating point scalar variables and later, the representation is changed to a single array variable of size three, a module designed with information hiding in mind would protect the remainder of the program from such a change.
In object-oriented programming, information hiding reduces software development risk by shifting the code's dependency on an uncertain implementation (design decision) onto a well-defined interface. Through encapsulation, we can provide security to our data function.
Encapsulation can be enforced in C++ using the concept of classes which binds data and function together. Private data and member functions will not be visible to outside world. For example, in the following class two members x, y declared as private can be accessed only by the member functions, not even in main( ) we can call these data members.

```
class XY
 {
        int x,y;
   public:
        void getdata();
      void dispdata();
};
 void  XY::getdata()
{   cout<< "Enter two values\n";
    cin>>a>>b;
     }
  void XY::dispdata()
{  cout<<a<<b;
}

   void main()
     {    XY  xy1,xy2;
          xy1.getdata();
          xy2.dispdata();
     }
```

Q.89 Write a C++ program which reads in a line of text word by word in a loop using *cin* statement and outputs on the screen the words in the reverse order using *cout* statement. For example, if the input is 'Jack Prat could eat no fat', then the output is 'fat no eat could prat Jack'. **(8)**

Ans: A C++ program to read a text and output words of the text in reverse order:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
 char words[25][15],wo[15];
 int len=0,i=0;
 cout<<endl<<"\nEnter words separated by space and type XXX to terminate \n";
 while (1)
 {
            cin>>wo;
            if (strcmp(wo,"XXX")==0)
                        break;
            strcpy(words[i++],wo);
            if (i==24)break;
 }
 cout<<endl<<"Output in Reverse order\n";
 while (1)
 {
            cout<<words[--i]<<' ';
            if (i==0)break;
 }
 getch();
}
```

Q 90 How are member functions different from other global functions? **(2)**

Ans: A member function and a non –member function are entirely different from each other.
A member function has full access to the private data members of a class. A member function can directly access the data members without taking the help of an object. For calling member function in main(), dot operator is required along with an object. A member function cannot be called without an object outside the class.
A non member function cannot access the private members of a class. Non-member function are called in main( ) without using an object of a class.

Q.91 Define a class Word which can hold a word where the length of the word is specified in the constructor. Implement the constructor and the destructor. **(6)**

Ans:

```
#include<iostream.h>
 #include<conio.h>
 #include<stdio.h>
 #include<string.h>
 class Word
```

```
{
  char *word;
  int length;
  public:
            Word()
            {
               length=0;
               word=new char[length+1];
            }
            Word(char *s)
            {
                     length=strlen(s);
                     word=new char[length+1];
                     strcpy(word,s);

            }
            void display()
            { cout<<word<<"\n";
            }
};
void main()
{
  char *w1="Girl";
  Word word1(w1);
  Word word2("Good");
  word1.display();
  word2.display();
}
```

Q.92 Differentiate and give examples to bring out the difference between

(i) function overloading and function overriding
(ii) function template and inline functions
(iii) default constructor and copy constructor
(iv) public and *private* access specifiers **(3.5 x 4 = 14)**

Ans:

(i). **Function overloading and function overriding:**

Function overloading means the use of the same function name to create functions that perform a variety of different tasks. The functions would perform different operations depending on the argument list in the function call. The correct function to be invoked is selected by seeing the number and type of the arguments but not the function type. For example an overloaded function volume() handles different types of data as shown below:

```
int  volume(int s);
double volume(double r, int h);
long volume(long l, long b,int h);
```

Function overridding is used to describe virtual function redefinition by a derived class. This is useful when we want to have multiple derived classes implementing a base class function. We can put the common code to all of the derived classes in the base class function, and then in each of the derived class functions we can add the code specific to each one, and then just invoke the parent method. It differs from function overloading in the sense that all aspects of the parameters should be same when a overridden function is redefined in derived class.

(ii)**Function template and inline functions**

Function templates are those functions which can handle different data types without separate code for each of them. For a similar operation on several kinds of data types, a programmer need not write different versions by overloading a function. It is enough if he writes a C++ template based function. This will take care of all the data types. An example that finds the maximum value of an array using template is given below:

```
#include<iostream.h>
template<class T>
void max(T a[],T &m,int n)
{
 for(int i=0;i<n;i++)
            if(a[i]>m)
                        m=a[i];
}
int main()
{
 int x[5]={10,50,30,40,20};
 int m=x[0];
 max(x,m,5);
 cout<<"\n The maximum value is : "<<m;
 return 0;
}
```

Inline Functions: An inline function is one for which the compiler copies the code from the function definition directly into the code of the calling function rather than creating a separate set of instructions in memory. Instead of transferring control to and from the function code segment, a copy of the function body may be substituted directly for the function call. In this way, the performance overhead of a function call is avoided. A function is declared inline by using the inline function specifier or by defining a member function within a class or structure definition. The inline specifier is only a suggestion to the compiler that an inline expansion can be performed; the compiler is free to ignore the suggestion.

The following code fragment shows an inline function definition.

```
inline int add(int i, int j) { return i + j;}
```

(iii) **Default constructor and copy constructor**

Dafault constructor: A constructor that accepts no argument is called default constructor. This default constructor takes no parameters, or has default values for all parameters. The default constructor for the class A is A::A(). If no such constructor is defined, then the compiler supplies a default constructor. A default parameter is one where the parameter value is supplied in the definition. For example in the class A defined below 5 is the default value parameter.

```
Class A
{    int value;
  Public:
    A(int param=5)
   {
     value = param;
   }
};
```

A copy constructor is a special constructor in the C++ programming language used to create a new object as a copy of an existing object. This constructor takes a single argument: a reference to the object to be copied. Normally the compiler automatically creates a copy constructor for each class (known as an implicit copy constructor) but for special cases the programmer creates the copy constructor, known as an explicit copy constructor. In such cases, the compiler doesn't create one.

For example the following will be valid copy constructors for class A.

```
A(A const&);
A(A&);
A(A const volatile&);
A(A volatile&);
```

(iv)**Public and Private access specifiers**:

The keywords public and private are visibility labels. When the data members of a class are declared as private, then they will be visible only within the class. If we declare the class members as public then it can be accessed from the outside world also. Private specifier is used in data hiding which is the key feature of object oriented programming. The use of the keyword private is optional. When no specifier is used the data member is private by default. If all the data members of a class are declared as private then such a class is completely hidden from the outside world and does not serve any purpose.

```
class class-name
{
   private:
 int x,y;//No entry to private area
```

```
  private:
 int a,b;//entry allowed to public area
}
```

Q.93 The keyword 'virtual' can be used for functions as well as classes in C++. Explain the two different uses. Give an example each. **(6)**

Ans: Virtual function: virtual functions are important because they support polymorphism at run time. A virtual function is a member function that is declared within the base class but redefined inside the derived class. To create a virtual function we precede the function declaration with the keyword virtual. Thus we can assume the model of "one interface multiple method". The virtual function within the base class defines the form of interface to that function. It happens when a virtual function is called through a pointer.The following example illustrates the use of a virtual function:

```
#include<iostream.h>
using namespace std;
class base
{
  public:
  int i;
  base(int x)
  {
            i=x;
  }
  virtual void func()
  {
            cout<<"\n using base version of function";
            cout<<i<<endl;
  }
};
class derived: public base
{
  public:
  derived(int x):base(x){}
  void func()
      {
            cout<<"\n using derived version";
            cout<<i*i<<endl;
      }
};
class derived1: public base
{
  public:
  derived1(int x): base(x){}
  void func()
```

```
  {
        cout<<"\n using derived1 version";
        cout<<i+i<<endl;
  }
};
int main()
{
  base *p;
  base ob(10);
  derived d_ob(10);
  derived1 d_ob1(10);
  p=&ob;
  p->func();
  p=&d_ob;
  p->func();
  p=&d_ob1;
  p->func();
  return 0;
}
```

Virtual class: The concept of virtual class is very important as far as inheritance is concerned. Basically, the need for a making a class virtual arises when all the three basic types of inheritance exist in the same program- Multiple, multilevel and hierarchical inheritance.

For instance suppose we create a class child which inherit the properties of two other classes parent1 and parent2 and these two parent classes are inherited from a grandparent class. The grandfather class is the indirect base class for child class. Now the class child will have properties of the grandfather class but inherited twice, once through both the parents. The child can also inherit the properties straight from its indirect base class grandfather. To avoid the ambiguity caused due to the duplicate sets of inherited data we declare the common base class as virtual. Those classes that directly inherit the virtual class will be declared as follows:

```
class grandparents
{
  ....
  ....
};
class parent1: virtual public grandparents
{
  ....
  ....
};
class parent2: public virtual grandparents
{
  ....
  ....
};
class child: public parent1,public parent2
{
```

```
    ....//only one copy of grandparents
    ....//will be inherited.
};
```

Q.94 Define a class Array which can hold 10 elements of type int. Define a member function
int Get(int index);
which returns the indexth element of the array if index is between 0 to 9 and throws an exception if index is out of bounds. Catch the exception in the main program and print an error. **(8)**

Ans:

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

class Array
{
 int no[10],i;
 public:
     Array();
     void getdata();
     int get (int index)
          {
               if (index>=0 && index<10)
                  return no[index];
               else
                 throw index;
          }
};
Array::Array()
{
  for (i=0;i<10;i++)
  no[i]=0;

}
void Array::getdata()
{
               cout<<endl<<"Accepting Input from User"<<endl;
               for (i=0;i<10;i++)
               {

                    cout<<"please enter an element:-> ";
                    cin>>no[i];
               }
          }

void main()
```

```
{
 class Array obj;
 obj.getdata();
 //char ch='y';
 try
 {

            cout<<endl<<"Enter the index;-> ";
            int ind,value=0;
            cin>>ind;
            value=obj.get(ind);
            cout<<endl<<"Value at that index = "<<value;
            getch();
  }
  catch(int m)
  {
                        cout<<endl<<"U have opted for wrong index";
                        getch();
  }
}
```

Q 95 Define a class Bag(duplicate values permitted) of 50 integers. The values of the integers is between 1 and 10. Overload the subscript operator[]. **(6)**

Ans:

```
#include <iostream.h>
class Bag
{
  int val[50];
  int size;
   public:
  Bag();
  Bag(int *x);
  int &operator[](int i)
  { return val[i];
  }
};
Bag::Bag()
{    size=50;
  for(int i=0;i<50;i++)
       val[i]=0;
}
Bag::Bag(int *x)
{  int size=sizeof(x);
  for(int i=0;i<size;i++)
       val[i]=x[i];
}
```

```
void main()
{  Bag b1;
   int x[50],n;
   cout<<"How many values u want to put in bag";
   cin>>n;
   cout<<"Enter values between 1 to 10";
   for(int i=0;i<n;i++)
    cin>>x[i];
    b1=x;
   cout<<"values in the bags\n";
   for(i=0;i<n;i++)
   cout<<x[i]<<"\n";
}
```

Q.96 What is the output of the following program segment?

```
void main(){
      char buffer[80]; cout<< "Enter a line with *: "<<endl;
      cin.get(buffer, 8, '*');
      cout << "The buffer contains " << buffer << endl;}
```

Input : Hello World*

Ans: output:- The buffer contains HELLO W

Q.97 What does the poem.txt contain?

```
void main()
{
ofstream outfile("poem.txt");
outfile << "Mary had a lamb";
outfile.seekp(11, ios::beg);
outfile << "little";
                }
```

Ans: poem.txt contain : Mary had a little lamb

Q.98 Given

```
int * entry = new int [10];
```

Write code to fill the array with 10 numbers entered through the keyboard.

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr();
 int *entry=new int[10];
 cout<<"\n Enter ten numbers; ";
```

```
for(int i=0;i<10;i++)
        cin>> entry[i];
for(i=0;i<10;i++)
        cout<<entry[i]<<"\n";
getch();
}
```

Q.99 What is the output of the following program segment?

```
void main() {
   int i = 6;
   double d = 3.0;
   char * str = "OOP";
   cout.setf(ios::left);cout.setf(ios::fixed);
   cout << setfill('#') << setw(5) << str;
   cout << setprecision(1) << i + d;
}
```

Ans: output:- OOP##9

Q.100 What is the output of the following program segment?

```
void main ( )
        { int xyz = 333;
         cout << endl <<
         "The output is \n" << endl
          ; cout << "x\
            y\
        z\
        = "
        << xyz;
}
```

Ans: output:- the output is
xyz=333

Q.101 When do you rethrow any (caught) exception? Give the syntax.

Ans: An exception handler can rethrow an exception that it has caught without even processing it. It simply invokes the throw without passing any arguments.

```
throw;
```

This exception is then thrown to the next try/catch and is caught by a catch statement listed after that enclosing try block.

Q.102 Point out the error and correct it

```
class X( static int a; char b; int star() {return a * b } };
```

(2 x 7)

Ans: the error in:

```
class X( static int a; char b;int star{return a*b;}};
```

is that class X should be followed by an opening curly bracket and not a curve bracket.

Q.103 Assume that the cell users are two kinds – those with a post paid option and those with a prepaid option. Post paid gives a fixed free talk time and the rest is computed at the rate of Rs.1.90 per pulse. Prepaid cards have a fixed talk time.

Define a class Cell_user as a base class and derive the hierarchy of classes. Define member functions and override them wherever necessary to

(i) retrieve the talk time left for each user.

(ii) print the bill in a proper format containing all the information for the post paid user. **(14)**

Ans:

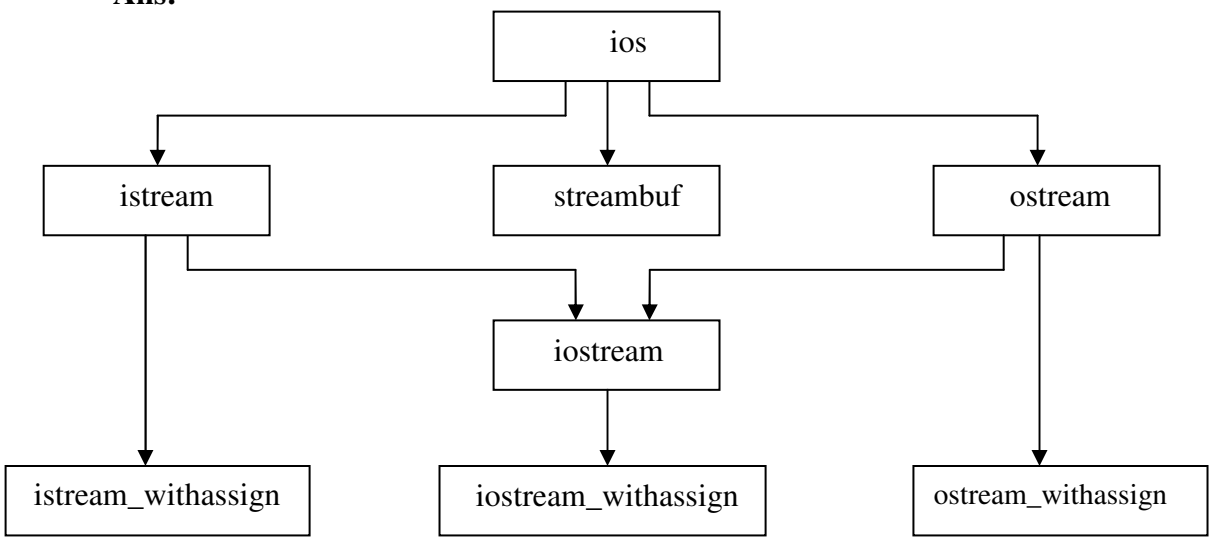
```
class Cell_User
{       protected:
 char cellno[15];
 char name[25], add[50];
 public:
  char plan;
  void getdata();
  void showdata();
};
void Cell_User::getdata()
 {
            cout<<endl<<"Enter your Name:-> ";
            cin.getline(name,25);
            cout<<endl<<"Enter your Address:-> ";
            cin.getline(add,50);
            cout<<endl<<"Enter Cell Number:-> ";
            cin>>cellno;
            cout<<endl<<"Enter Plan ('O' for Post Paid and 'R' for Pre Paid:-> ";
            cin>>plan;
}
void Cell_User::showdata()
{             cout<<"\nName:"<<'\t'<<name;
            cout<<"\nCellno:"<<'\t'<<cellno;
            cout<<"\nAddress:"<<'\t'<<add<<"\n\n";
            cout<<"Plan:"<<'\t'<<plan;
}
class post : public Cell_User
{
 int freepulse, usedpulse,extra;
 public:
  post()
  {
   freepulse=50;
   usedpulse=0;
```

```
  }
  void getdata()
  {cout<<"\nEnter used Time for this user in Pulses:-> ";
   cin>>usedpulse;
  }
  void showdata();
};
void post::showdata()
  {
                cout<<"\nPostpaid User\n";
                cout<<"\nFreepulse="<<freepulse;
                cout<<"\nUsedpulse="<<usedpulse;
                extra=usedpulse-freepulse;
                cout<<"\n Extra pulses:"<<'\t'<<extra;
                if (extra>0)
                {
                     cout<<"\nMonthly Rent="<<600;
                          cout<<"\nCall charges="<<extra*1.90;
                          cout<<"\nTotal="<<600+extra*1.90;
                }
              else
              {  cout<<"\nMonthly Rent="<<600;
                    cout<<"\nNo extra charges\n";
              }
        }
class pre : public Cell_User
{
 int talktime, usedtime,lefttime;

 public:

          pre()
          {
                talktime=50;
                usedtime=0;
          }
          void getdata()
          {
                cout<<"\nEnter used Time for this user in Pulses:-> ";
                cin>>usedtime;
          }
          void showdata()
          {
                cout<<"\nFree talk time="<<talktime;
                cout<<"\nUsed talk time"<<usedtime;
                lefttime=talktime-usedtime;
                if(lefttime>0)
```

```
        cout<<"\n Time left:"<<'\t'<<lefttime;
                        else
                        cout<<"\n Please recharge your coupan\n";
                }
        };
        void main()
        {       clrscr();
                Cell_User c;
         c.getdata();
         if(c.plan=='O')
          {
           post obj1;
           obj1.getdata();
           c.showdata();
                obj1.showdata();
          }
          if(c.plan=='R')
          {       pre obj2;
                obj2.getdata();
                c.showdata();
                obj2.showdata();
          }
          getch();
}
```

Q.104 Explain the I/O stream hierarchy in C++. **(4)**

Ans:



The above hierarchies of classes are used to define various streams which deal with both the console and disc files. These classes are declared in the header file iostream. ios is the base for istream, ostream and iostream base classes. ios is declared as a virtual base class so that only one copy of its members are inherited by the iostream. The three classes istream_withassign,

ostream_withassign and iostream_withassign are used to add assignment operators to these classes.

Q.105 Define a class License that has a driving license number, name and address. Define constructors that take on parameter that is just the number and another where all parameters are present. **(6)**

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<fstream.h>
#include<stdlib.h>
class License
{
 long dlno;
 char name[25],add[50];
 public:
        License()
        {
                dlno=0;
                strcpy(name, " ");
                strcpy(add, " ");

        }
        License(long no)
        {
                dlno=no;
                strcpy(name," ");
                strcpy(add, " ");
        }
        License(long no, char n[25], char ad[50])
        {
                dlno=no;
                strcpy(name,n);
                strcpy(add,ad);

        }

        void getdata()
        {
                cout<<endl<<"Enter your name:-> ";
                cin>>name;
                cin.get();
                cout<<endl<<"Enter your address:-> ";
                cin.getline(add,50);
```

```
                cout<<endl<<"Enter License Number:-> ";
                cin>>dlno;
        }
        void showdata()
        {
                cout<<endl<<dlno<<'\t'<<name<<'\t'<<add;
        }
};
```

Q.106 Create a file and store the License information. **(4)**

Ans:

```
void main()
{
 class License obj1(101),obj2(102,"Ramesh","Laxminagar");
 fstream file;
 file.open("List.TXT",ios::in | ios::out);
 if (file==NULL)
 {
        cout<<endl<<"Memory Allocation Problem.";
        getch();
        exit(1);
 }
 class License obj;
 obj.getdata();
 file.write((char *) & obj, sizeof(obj));
 file.seekg(0);
 cout<<endl<<"Following informations are stored in the file:\n";
 while(1)
 {
        file.read((char *) &obj, sizeof(obj));
        if (file)
                obj.showdata();
        else
                break;
 }
}
```

Q.107 Define a class template Pair which can have a pair of values. The type of the values is the parameter of the template. Define a member function Add() which adds the two values and returns the sum. **(10)**

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
template <class T>
```

```
class pair
{
 T val1, val2, val3;
 public:
        void Add()
        {
                val3=val1 + val2;
        }
        void getdata()
        {
                cout<<endl<<"Enter two input:-> ";
                cin>>val1>>val2;
        }
        void showdata()
        {
                cout<<val3;
        }
};

void main()
{
 pair <int> obj1;
 cout<<endl<<"Enter two integer input.\n";
 obj1.getdata();
 obj1.Add();
 obj1.showdata();
 pair <float> obj2;
 cout<<endl<<"Enter two float input.\n";
 obj2.getdata();
 obj2.Add();
 obj2.showdata();
 getch();
}
```

Q.108 Check if the template works for char* where Add() must concatenate the two strings. If not find a solution. **(4)**

Ans:

It will not work for char * where Add() must concatenate the two strings.
This can be done by overloading the function add() as

```
void Add(char *str1, char *str2)
{
   char *str3;
    strcpy(str3,str1);
    strcat(str3,str2);
    cout<<str3;
}
```

Q.109 Write short notes on
(i) Constructor
(ii) Stream
(iii) Access control.
(iv) Friend. **(3.5 x4)**

Ans:

i) **Constructor:**

A constructor is a member function which has the same name as at that of the class. It is used to initialize the data members of the class. A constructor is automatically invoked by the compiler when an object is created. A constructor which does not take any arguments is called default constructor.

The restrictions applied to a constructor are:

- A constructor cannot have any return type as it does not return any value.
- A constructor should be declared in the public part of the class.
- A derived class can call the constructor of the base class but a constructor is never inherited.
- A constructor function cannot be made virtual.
- That object which has a constructor cannot be used as a member of the union.

C++ provides us with very helpful feature i.e., copy constructor. It is the mechanism in which we can declare and initialize an object from another both belonging to the same class. For example, the following statement will create an object obj2 and also initialize it by taking values from obj1.

integer obj2(obj1);

We can also use the following instead the above for the same purpose.

integer obj2=obj1;

This process of initializing objects using copy constructor is known as “copy initialization”.

Another statement obj2=obj1 will not assign the values using the copy constructor. Although the statement is valid this can be done using operator overloading.

A copy constructor accepts a reference to an object of the same class as an argument. We cannot pass an argument by value to a copy constructor.

The following is an example program:

```
#include<iostream.h>
class item
{
                    int code;
  public:
                    item(){}
                    item(int n){code=n;}
```

```
                item(item & a)
                {
                        code=a.code;
                }

                void show(void)
                {cout<<code;}
};
int main()
{
                item obj(12);
                item obj2(obj1);
                item obj3=obj1;
                item obj4;
                obj4=obj1;//copy constructor not called
                cout<<"\n code of obj1 : ";
                obj1.show();
                cout<<"\n code of obj2 : ";
                obj2.show();
                cout<<"\n code of obj3 : ";
                obj3.show();
                cout<<"\n code of obj4 : ";
                obj4.show();
        }
```

ii) Stream: Stream is defined as an interface supplied by the I/O system to the programmer and that which is independent of the actual device being used. Stream is actually a sequence of bytes. These are basically of two types: one is when it is a source through which input can be taken and the other is when it acts as a destination to which output can be sent. The source stream is called the input stream and the destination stream is called the output stream. There are several streams likewise cin is an input stream which extracts input from the keyboard and cout is the output stream which inserts the ouput to the screen. The three streams used are:-

- istream(input stream).this stream inherits the properties of ios and declares input functions such get(), getline() and read().the extraction operator used for this >>
- ostream(output stream). This also inherits properties from ios and declared output functions such as put() and write(). It uses the insertion operator <<.
- Iostream(input/output stream). This inherits properties from ios istream and ostream and through multiple inheritance contains all the input output functions.

iii). **Access control:** This is done using three keywords public, private and protected. These keywords are visibility labels. When the data members of a class are declared as private, then they will be visible only within the class. If we declare the class members as public then it can be accessed from the outside world also. Private specifier is used in data hiding which is the key feature of object oriented programming. The use of the keyword private is optional. When no specifier is used the data member is private by default. If all the data members of a class are declared as private then such a class is completely hidden from the outside world and does not serve any purpose.

```
class class-name
{
   private:
    int x,y;//No entry to private area
   private:
    int a,b;//entry allowed to public area
}
```

These access specifiers determine how elements of the base class are inherited by the derived class. When the access specifier for the inherited base class is public, all public members of the base class become public members of the derived class. If the access specifier is private, all public members of the base class become private members for the derived class. Access specifier is optional. In case of a derived 'class' it is private by default. In case of a derived 'structure' it is public by default.

The protected specifier is equivalent to the private specifier with the exception that protected members of a base class are accessible to members of any class derived from the base. Outside the base or derived class, protected members are not accessible. The protected specifier can appear anywhere in the class.

When a protected member of a base class is inherited as public by the derived class, it becomes protected member of the derived class. If the base is inherited a private, the protected member of the base becomes private member of the derived class.

iv) **Friend:**

Friend is keyword in C++ which is used to declare friend function and friend classes. It can access the private data members of the class it has to take the help of the object of that class. A friend function is not the member of the class. When we create an object memory is not located for the friend function. It is defined like a normal function. While calling a friend function we not use object as friend function is not the part of the object. Thus is called like any normal non member function. It also helps in operator overloading.

The following is an example program:

```
#include<iostream.h>
using namespace std;
class one;

class two
{
    int a;
  public:
    void setvalue(int n){a=n;}
    friend void max(two,one);
};
class one
{
    int b;
  public:
    void setvalue(int n){b=n;}
    friend void max(two,one);
};
void max(two s,one t)
{
    if(s.a>=t.b)
                cout<<s.a;
    else
                cout<<t.b;
}
int main()
{
    one obj1;
    obj1.setvalue(5);
    two obj2;
    obj2.setvalue(10);
    max(obj2,obj1);
    return 0;
}
```

When we declare all the member functions of a class as friend functions of another then such a class is known as a friend class.

```
class Z
{
friend class X;//all member functions af X are friends of Z
}
```

Q.110 What is the difference between passing a parameter by reference and constant reference? **(2)**

Ans: By passing a parameter by reference, the formal argument in the function becomes alias to the actual argument in the calling function. Thus the function actually works on the original data. For example:

```
void swap(int &a, int &b)//a and b are reference variables
{
 int t=a;
 a=b;
 b=t;
}
```

The calling function would be:

```
swap(m,n);
```

the values of m and n integers will be swapped using aliases.

Constant Reference: A function may also return a constant reference. Example:

```
int & max(int &x,int &y)
{
 If((x+y)!=0)
          return x;
 return y;
}
```

The above function shall return a reference to x or y.

Q.111 Define the following terms and give an example to show its realization in C++

(i) Encapsulation (ii) Class variables and class functions
(iii) Repeated inheritance (iv) Overloading **(3.5 x 4)**

Ans:

(i) **Encapsulation:** The term encapsulation is often used interchangeably with information hiding. Not all agree on the distinctions between the two though; one may think of information hiding as being the principle and encapsulation being the technique. A software module hides information by encapsulating the information into a module or other construct which presents an interface.

A common use of information hiding is to hide the physical storage layout for data so that if it is changed, the change is restricted to a small subset of the total program. For example, if a three-dimensional point (x,y,z) is represented in a program with three floating point scalar variables and later, the representation is changed to a single array variable of size three, a module designed with information hiding in mind would protect the remainder of the program from such a change.

(ii) **Class variables and class functions:** Static variables are associated with the class rather than any class objects, so they are known as class variables. The type and scope of each class variable must be defined outside the class definition as they are stored separately rather than as a part of an object.

Static functions can have access to only other static members (functions and variables) declared in the same class. These functions are known as class functions. A class function is called by using the class name instead of its objects like

Class-name:: function-name;

(iii)**Repeated Inheritance**: C++ requires that the programmer state which parent class the feature to use should come from. C++ does not support explicit repeated inheritance since there would be no way to qualify which superclass to use. Repeated inheritance means not being able to explicitly inherit multiple times from a single class. Multiple Inheritance usually corresponds to this-a relationship. It is a common mistake to use multiple inheritance for has-a relationship. For example, a Filter circuit may consist of Resistors, Capacitors and Inductors. It'll be a mistake to declare Filter as a subclass of resistor, capacitor and inductor (multiple inheritance). Instead, it is better to declare component objects of resistors, capacitors, and inductors within the new Filter class, and reference the component methods indirectly through these component objects.

(iv)**Overloading:** The process of making a function or an operator behave in different manners is known as overloading. Function overloading means the use of same function name to perform a variety of different tasks. The correct function to be invoked is selected by seeing the number and type of the arguments but not the function type.

Operator overloading refers to adding a special meaning to an operator. The operator doesn't lose its original meaning but the new meaning is added for that particular class. For example, we can add two strings as:

String1+String2=string3;

"good"+ "girl"= "goodgirl";

An example program of function overloading:

```
//Function volume is overloaded three times.
#include<iostream.h>
using namespace std;

int vol(int);
double vol(double,int);
long vol(int,int,int);
```

```
int main()
{
  cout<<volume(2)<<"\n";
  cout<<volume(2.5,8)<<"\n";
  cout<<volume(2,3,5)<<"\n";
  return 0;
}

int volume(int s)
{
  return(s*s*s);
}

double volume(double r,int h)
{
  return(3.14519*r*r*h);
}

long volume(int l,int b,int h)
{
  return(l*b*h);
}
```

The output would be:

```
8
157.26
30
```

Q.112 Define a class Point which is a three dimensional point having x, y and z coordinates. Define a constructor which assigns 0 if any of the coordinates is not passed. **(6)**

Ans:

```
#include <iostream.h>
 #include<conio.h>
 class Point {
    int X;
    int Y;
    int Z;
 public:
```

```
  Point() {X=Y=Z=0;}
  Point(int x, int y, int z)
  { X=x; Y=y;Z=z;}
  int getX(){ return X;}
  int getY(){return Y;}
  int getZ(){return Z;}
};
```

Q.113 Define a class Sphere which has a radius and a center. Use the class Point defined in (Q 112) to define the center. Define the constructor. Define a member function Volume (which is given as πr^3) which computes the volume. Use the class in main() to create an instance of sphere. Compute the volume and print it.

(10)

Ans:

```
class Sphere
{
  int radius;
  Point center;
  public:
   Sphere(){}
   Sphere(int rad,Point &p)
   {  radius=rad;
     center=p;
   }
   void showdata()
   {  cout<<"Radius="<<radius;

cout<<"\nCenter="<<"("<<center.getX()<<","<<center.getY()<<","<<center.get
Z()<<")";
    }
  float volume()
  {  return (3.14*radius*radius*radius);
  }
};
void main()
{  clrscr();
  Point YourPoint();
  Point p(3,4,5);
  Sphere s1(1,p);
```

```
    s1.showdata();
    float f=s1.volume();
    cout<<"\nVolume="<<f;
    getch();
}
```

Q.114 What is the access specification that should precede a ***friend*** function? **(2)**

Ans: The access specifier that should precede a friend function can be either public or private. The meaning and working of the friend function is not altered by using any of the two access specifiers.

Q.115 Define a class Vector which can hold 20 elements of type *int*. Write a member function read() which reads the values from the input stream and assigns them to the Vector object. **(6)**

Ans:

```
#include<conio.h>
#include<iostream.h>
const size = 20;
class vector
{
 int v[size];
 public:
 vector();
 void read();
 vector(int *x);
 friend vector operator +(vector a, vector b);
 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[i];
}

void vector :: read()
{
 int x[20];
 cout<<"\nEnter 20 values";
```

```
 for(int i=0;i<size;i++)
     cin>>x[i];
 for(i=0; i<size; i++)
     v[i] = x[i];
}
```

Q.116 Overload the operator + for the class Vector defined in Q115 above which adds two vectors. Define the addition operator as a ***friend*** function. The result must be returned as an object of the class Vector.

(8)

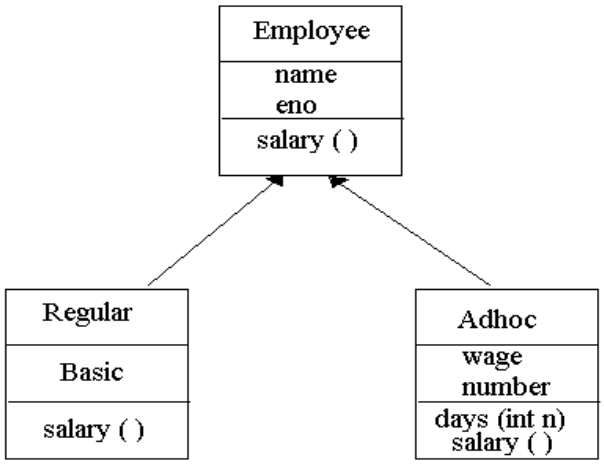
Ans:

```
vector operator +(vector a, vector b)
{
 vector c;
 for(int i=0;i<size;i++)
    c.v[i] = a.v[i] + b.v[i];
 return c;
}
istream & operator >> (istream&x, vector&b)
{
 for(int i=0;i<size;i++)
    x >> b.v[i];
    return(x);
}
ostream & operator << (ostream&dout, vector&b)
{  dout<<"("<<b.v[0];
 for(int i=1;i<size;i++)
    dout <<  " , " << b.v[i];
    dout <<" ) ";
    return(dout);
}

void main()
{
 clrscr();
 vector m;
 m.read();
 vector n;
 cout<< "enter elements of vector n"<<'\n';
 cin>>n;
 cout<<'\n';
 cout<<"m = "<<m<<'\n';
 cout<<"\nn="<<n<<'\n';
```

```
 vector p;
 p = m+n;
cout<<'\n';
 cout<<"p = "<<p<<'\n';
getch();
}
```

Q.117 An organization has two types of employees: regular and adhoc. Regular employees get a salary which is basic + DA + HRA where DA is 10% of basic and HRA is 30% of basic. Adhoc employees are daily wagers who get a salary which is equal to Number * Wage.

(i) Define the classes shown in the following class hierarchy diagram:



(ii) Define the constructors. When a regular employee is created, basic must be a parameter. When adhoc employee is created wage must be a parameter.

(iii) Define the destructors.

(iv) Define the member functions for each class. The member function days ( ) updates number of the Adhoc employee.

(v) Write a test program to test the classes. **(4+4+2+4+2)**

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

class Employee
{
 char name[25];
 int eno;
 public:
```

```
 Employee()
 {
        strcpy(name, " ");
        eno=0;
 }
 void getdata()
 {
        cout<<endl<<"Name of Employee:-> ";
        cin>>name;
        cout<<endl<<"Employee Number:-> ";
        cin>>eno;
 }

 void salary()
 {
        cout<<endl<<eno<<'\t'<<name;
 }

 ~Employee()
 {

 }
};
class Regular : public Employee
{
 float Basic;
 public:
        Regular(float bs)
        {
                Basic=bs;
        }

        void getdata()
        {
                Employee::getdata();
        }
        void salary()
        {
                //call Base showdata();
                Employee::salary();
                cout<<endl<<"Basic = "<<Basic;
                cout<<endl<<"DA    = "<<Basic*0.1;
                cout<<endl<<"HRA   = "<<Basic*0.3;
                cout<<endl<<"Total Payble = "<<(Basic + (Basic*0.1) + (Basic*0.3));

        }
```

```
        ~Regular()
        {
        }
};
class Adhoc : public Employee
{
 float wage;
 int number;
 public:
        Adhoc(float wg)
        {
                wage=wg;
                number=0;
        }

        void days(int n)
        {
                number+=n;
        }

        void salary()
        {
                //call Base class Showdata
                Employee::salary();
                cout<<endl<<"Total Payble = "<<wage*number;
        }
        void getdata()
        {
                Employee::getdata();
        }
        ~Adhoc()
        {
        }
};
void main()
{
 class Regular robj(5000);
 class Adhoc aobj(65);
 robj.getdata();
 aobj.getdata();
 aobj.days(25);
 cout<<endl<<"Details of Regular Employee1\n";
 robj.salary();
 cout<<endl<<"Details of Adhoc Employee1\n";
 aobj.salary();
 getch();
}
```

Q.118 Write a function template for the function Power ( ) which has two parameters base and exp and returns $base^{exp}$. The type of base is the parameter to the template and exp is *int*. If exp is negative then it must be converted to its positive equivalent. For example, 2^3 and 2^{-3} must both return 8. **(6)**

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<math.h>
template <class T>
class math
{
  int exp;
  T base,s;
  public:
  math(T x, int y)
  {
            exp=abs(y);
            base=x;
            s=1;
  }
  T Power()
  {
     s=pow(base,exp);
     return s;
  }
};
```

Q.119 Instantiate the template function defined above for *int* type. **(2)**

Ans:

```
void main()
{       clrscr();
        math <int>obj1(10,-2);
        cout<<endl<<"Value= "<<obj1.Power();
        getch();
}
```

Q.120 What is exception handling? **(2)**

Ans: Exception Handling:

Using exception handling we can more easily manage and respond to run time errors. It helps us in coping up with the unusual predictable problems that arise while executing a program.

Q.121 What are the keywords on which exception handling is built? Explain each one of them. Give an example. **(6)**

Ans: C++ exception handling is built upon 3 keywords: try, catch and throw.

- The 'try' block contains program statements that we want to monitor for exceptions.
- The 'throw' block throws an exception to the 'catch' block if it occurs within the try block.
- The 'catch' blocks proceeds on the exception thrown by the 'throw' block.

When an exception is thrown, it is caught by its corresponding catch statement which processes the exception. There can be more that one catch statement associated with a try. The catch statement that is used is determined by the type of the exception. An example illustrates the working of the three blocks.

```
#include<iostream.h>
using namespace std;
int main()
{
 cout<<"\n Start "<<endl;
 try
 {
        cout<<"\n Inside try block "<<endl;
        throw 10;
        cout<<"\n this will not execute ";
 }
 catch(int i)
 {
        cout<<"\n catch number "<<endl;
        cout<<i<<endl;
 }
 cout<<"End";
 return 0;
}
```

Q.122 Explain the following:

(i) ios class (ii) setf( ) function
(iii) rdstate() function (iv) writing of objects to files **(8)**

Ans:

i. **ios class:** ios is the base class meant for istream(input sream), ostream(output stream) and iostream(input/output stream).ios is declared as the virtual base class so that only one copy of its members are inherited by the iostream.

ii. **setf() function**: This function is used to specify format flags that can control the form of output display (such as left-justification) and (right justification). The syntax is:

setf(arg1,arg2);

where arg1 is a formatted flag and arg2 is a bit field.

iii. rdstate() function:
ios::rdstate() const;
This function returns the current internal error state flags of the stream. The internal error state flags are automatically set by calls to input/output functions to signal certain types of errors that happened during their execution. It does not contain any parameter.

iv. Writing objects to a file: C++ allows us to read from and write to the disk files objects directly. The binary function write() is used to write object to files. The function write copies a class object from memory byte to byte with no conversion. It writes only the data members and the member functions.

Q.123 Write a program which asks for a file name from the keyboard, opens a file with that name for output, reads a line from the keyboard character by character and writes the line onto the file. **(8)**

Ans:

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<fstream.h>
#include<stdlib.h>
void main()
{
 cout<<endl<<"Enter the file name for output:-> ";
 char name[25];
 cin>>name;
 ofstream out(name);

 if (out==NULL)
 {
        cout<<endl<<"\nUnable to open file.";
        getch();
        exit(1);
 }

 char ch;
 cout<<endl<<"Press 'T' to terminate input.\n";

 while(1)
 {
        cin.get(ch);
        if (ch=='T')break;
        out.put(ch);
 }
```

```
out.close();
ifstream in(name);
if (in==NULL)
{
        cout<<endl<<"\nUnexpected Error.";
        getch();
        exit(1);
}
cout<<endl<<"U have entered following\n";
while(in)
{
        in>>ch;
        cout<<ch;
}
in.close();
cout<<endl<<"Thanks";
getch();
}
```

Q.124 Differentiate between

(i) int* q = p; and n = *p;
(ii) Constructor and destructor
(iii) Interface and implementation of a class
(iv) *public* and *private* member of a class
(v) Overloaded pre-increment operator and overloaded post increment operator
(vi) Abstract and concrete class
(vii) *get* ( ) and *getline* ( ) function
(viii) Composition and inheritance **(16)**

Ans:

(i) int* q=p;
the value of p will be assigned to pointer q which will be treated as an address.
int n=*p;
in this an integer variable n is initialized with the value stored at the address to which the pointer p points.

(ii) A constructor is a member function which is used to initialize the data members of a class. It has the same name as that of the class. It does not have any return type and is invoked automatically on the creation of an object.
test() //constructor for the class test
A destructor is a member function that is used to destroy the objects that's have been created by the constructor. It frees the memory space for future use. It also has no return type but its declaration is preceded by a tilde.
~test(); //destructor function

(iii) The keywords public and private are visibility labels. When the data members of a class are declared as private, then they will be visible only within the class. If we declare the class members as public then it can be accessed from the outside world also. Private specifier is used in data hiding which is the key feature of object oriented programming. The use of the keyword private is optional. When no specifier is used the data member is private by default. If all the data members of a class are declared as private then such a class is completely hidden from the outside world and does not serve any purpose.

(iv) In earlier version of C++ when an increment operator was overloaded there was no way of knowing whether the operator precedes or follows its operand. That is assuming that the two statement ob++ and ++ob had identical meanings. However the modern specification for C++ has defined a way through which compiler distinguishes between these operator. For this two versions were used. One defined as

return-type class-name :: operator #( ) and the second is defined as
rerurn-type class-name :: operator #(int notinuse). In (int notinuse) zero value is passed.

(v) In C++ an abstract class is one which defines an interface, but does not necessarily provide implementations for all its member functions. An abstract class is meant to be used as the base class from which other classes are derived. The derived class is expected to provide implementations for the member functions that are not implemented in the base class. A derived class that implements all the missing functionality is called a concrete class.

(vi) get() function extracts a character from the stream and returns its value (casted to an integer).The C++ string class defines the global function getline() to read strings from an I/O stream. The getline() function reads a line from is and stores it into s. If a character delimiter is specified, then getline() will use delimiter to decide when to stop reading data. Example: getline(cin, s);

(vii)The difference between inheritance and composition is the difference between “is-a” and “has-a”. For example, we would not design a “car” object to be inherited from “engine”, “chassis”, “transmission”, and so on (“car is-a engine” makes no sense). We would “compose” the car object from its constituent parts (“car has-a engine” makes a lot of sense). Once we start thinking in terms of “is-a” versus "has-a", it's a lot easier to do proper design.

Q.125 What is the wrong with the following program?

```
#include <iostream.h>
void main ( )
{
   class Test {
```

```
    int x;
  public: int getX ( ) {return x;}
          void setX ( int a) { x = a;}
          void showX{ cout << getX ( ) << endl;}
 };
}
```

Ans:
Parameter names are used with function body.

Q.126 What is the output of the following program, if any?

```
#include <iostream.h>
class A {
   int a;
  public: A( int aa) { a = aa;}
          void print ( ) { cout << " A = " << a; }
};
class B : public A {
     int b;
public: B ( int aa, int bb) : A (aa) { b = bb;}
      void print ( ) { cout << " B = " << b; }
};
void main ( )
{ A objA(10);
  B objB(30, 20);
  objA = obj B;
  objA.print ( );
}
```

Ans: A=30

Q.127 What is wrong with the following code?

```
int *p = new int;
int *p = new int;
cout << "p = "  << p << ", q = " << q << ", p+q = " << p+q << endl;
```

Ans: p and q both the variables are not declared or initialized.

Q.128 How many times is the copy constructor called in the following code?

```
class copy_const{
};
copy_const f(copy_const u)
{ copy_const v(u);
  copy_const w = v;
   return w;
```

```
}
main ( )
{
  copy_const x;
  copy_const y = f (f (x));
                                    }
```

(4 x 4)

Ans: Copy constructor is not called in the given function.

Q.129 Discuss the role of inheritance in object-oriented programming. What is public, private and protected derivation? **(8)**

Ans:

Inheritance: it is the property by which one class can be derived from another class such that it acquires the properties of its base class. Just like a child inherits the properties of its parents in the same way OOP allows us to reuse the members and functions of class to be derived by another class. There are many different kinds of inheritance like multiple inheritance, multilevel inheritance and hierarchical inheritance.

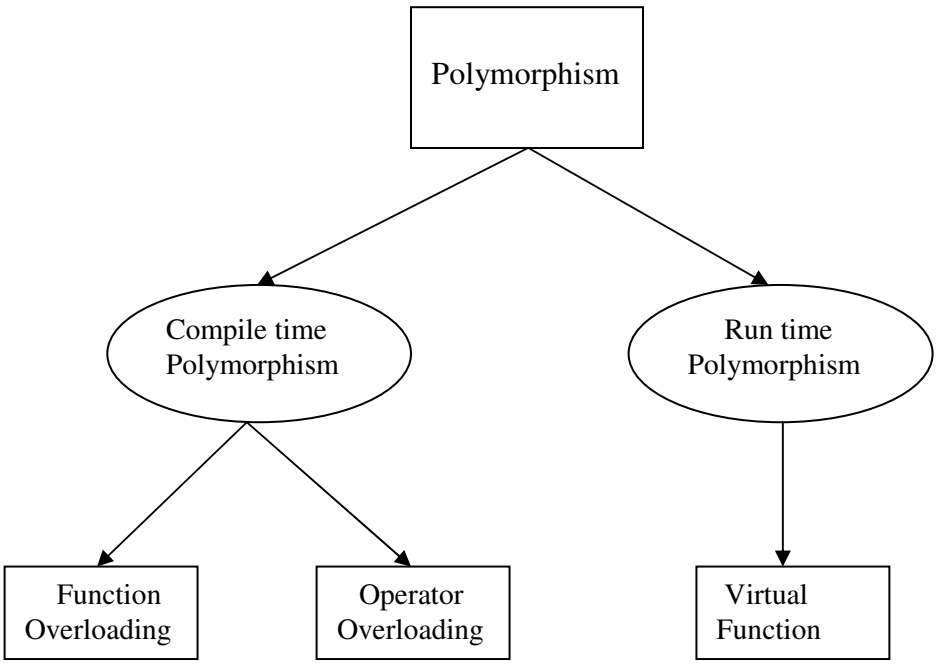
The visibility mode specified while declaring a derived class plays a very important role in the access control of the data members of the class. The visibility mode is one of the three keywords: public, private and protected. These access specifiers determine how elements of the base class are inherited by the derived class. When the access specifier for the inherited base class is public, all public members of the base class become public members of the derived class. If the access specifier is private, all public members of the base class become private members for the derived class. Access specifier is optional. In case of a derived 'class' it is private by default. In case of a derived 'structure' it is public by default.

The protected specifier is equivalent to the private specifier with the exception that protected members of a base class are accessible to members of any class derived from the base. Outside the base or derived class, protected members are not accessible. The protected specifier can appear anywhere in the class.

When a protected member of a base class is inherited as public by the derived class, it becomes protected member of the derived class. If the base is inherited a private, the protected member of the base becomes private member of the derived class.

Q.130 Explain the meaning of polymorphism. Describe how polymorphism is accomplished in C++ taking a suitable example? **(8)**

Ans:



Polymorphism is the property of representing one operation into many different forms. One operation may express different in different situations. The process of making a function or an operator behave in different manners is known as overloading the function or the operator.
Polymorphism is one of the most useful features of object oriented programming. It can be achieved both at run time and at compile time. At compile time polymorphism is achieved using function overloading and operator overloading. At the time of compilation the compiler knows about the exact matching as to which function to call or invoke. This is known as compile time polymorphism or early binding.
Polymorphism can also be achieved at run time. This is done using the concept of virtual functions. Which class function is to be invoked is decided at run time and then the corresponding object of that class is created accordingly. This is also called as late binding.
The following example program explains how polymorphism can be accomplished using function overloading.

```
//Function volume is overloaded three times.
#include<iostream.h>
int volume(int);
double volume(double,int);
long volume(long,int,int);
```

```
int main()
{
   cout<<volume(10)<<"\n";
   cout<<volume(2.5,8)<<"\n";
   cout<<volume(100L,75,15)<<"\n";
   return 0;
}

int volume(int s)
{
   return(s*s*s);
}

double volume(double r,int h)
{
   return(3.14519*r*r*h);
}

long volume(long l,int b,int h)
{
   return(l*b*h);
}
```

The output would be:

```
1000
157.26
112500
```

Q.131 Define a class **Fraction** whose objects represent rational numbers (i.e. fractions). Include integer data members **num** and **den** for storing the numerator and denominator respectively.

a. Write the following functions for fraction:

(i) parameterised constructors with one and two arguments.
(ii) copy constructors.
(iii) a function, eval-fract() for evaluating the value of the rational number.
(iv) a function, invert() for inverting the given rational number.

Ans:

a.

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

class fract
{
 int num;
```

```
 int den;
 public:
fract(int a);
 fract(int a,int b);
 fract(fract &);
 float eval_fract();
 void invert();
 void display();
};

fract::fract(int a)
{

  num=den=a;
}

fract :: fract(int a,int b)
{

  num=a;
  den=b;
}

fract :: fract(fract &f)
{
  cout<<"copy constructor called\n";
    num=f.num;
  den=f.den;
}

float fract :: eval_fract()
{
  float res;
  res=(float)num/den;
  return res;
}

void fract ::invert()
{
  int t=num;
  num=den;
  den=t;
}
```

```
void fract ::display()
{

 cout<<num<<"/"<<den;
}

void main()
{
 clrscr();
 fract f1(2);
 f1.display();
 cout<<"\n";
 fract f3(2,3);
 fract f2=f3;
 float f=f2.eval_fract();
 cout<<"The value of fraction is:"<<f<<"\n";
 f2.invert();
 f2.display();
 getch();
}
```

b. Overload >> and << operators for reading and printing the fraction object from the keyboard. **(2+8+6)**

Ans: Function declared inside the class fract for overloading << and >> operator:

```
friend istream & operator >> (istream &, fract &);
friend ostream & operator << (ostream &, fract &);
```

Functions defined outside the class:

```
istream & operator >> (istream&x, fract&b)
{
   x >> b.num>>b.den;
   return(x);
}
ostream & operator << (ostream&dout, fract&b)
  {  dout << b.num;
   dout <<" / "<<b.den;
   return(dout);
}
```

Statements in main() to invoke overloading of << and >> are:

```
cout<< "enter num and den of the fraction f"<<'\n';
 cin>>f;
 cout<<'\n';
 cout<<"fraction = "<<f<<'\n';
```

Q.132 What do you mean by static variable and static function? Give an example? **(5)**

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

- It has the following characteristics:
- On the creation of the first object of the class a static variable is always initialized by zero.
- All the objects share the single copy of a static variable.
- The scope is only within the class but its lifetime is through out the program.

A static function is just like a static variable. a static function has the following properties:

- A function can access only the static variable where both belong to the same class.
- A static function is called using the class name directly and not the object like

class-name :: function-name;

The example given below demonstrates the use of static variable 'count' and static function 'display()':

```
#include<iostream.h>
using namespace std;
class demo
{
  int num;
  static int count;
  public:
  void set(void)
  {
        num=++count;
  }
  void show(void)
  {
        cout<<"Number : "<< num<<"\n";
  }
  static void display(void)
  {
        cout<<"count : "<<count<<"\n";
  }
};
int demo::count;
int main()
{
  demo d1,d2;
  d1.set();
  d2.set();
  demo::display();//accessing static function
  demo d3;
  d3.set();
  d1.show();
```

```
  d2.show();
  d3.show();
  return 0;
}
```

Q.133 Write a template function to find the maximum number from a template array of size N. **(8)**

Ans:

```
using namespace std;
template<class T>
void max(T a[],T &m, int n)
{
  for(int i=0;i<n;i++)
        if(a[i]>m)
              m=a[i];
}

int main()
{
  int x[5]={10,50,30,40,20};
  int m=x[0];
  max(x,m,5);
  cout<<"\n The maximum value is : "<<m;
  return 0;
}
```

Q.134 Describe the basic characteristics object-oriented programming. **(3)**

Ans: Object Oriented Programming (OOP) is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.
Following are the basic characteristics of Object Oriented Programming:

- Encapsulation: The wrapping up of data and functions into a single unit is called encapsulation. The data can only be accessed by the function with which they have been tied up and not by the outside world. It creates an interface between the object's data and the program.
- Inheritance: It is the property by which one class can be derived from another class such that it acquires the properties of its base class. Just like a child inherits the properties of its parents in the same way OOP allows us to reuse the members and functions of class to be derived by another class. There are many different kinds of inheritance like multiple inheritance, multilevel inheritance and hierarchical inheritance.

- Polymorphism: Polymorphism is the property of representing one operation into many different forms. One operation may express different in different situations. The process of making a function or an operator behave in different manners is known as overloading the function or the operator.
- Data Abstraction: Abstraction is the method of representing only the essential features and only which is necessarily required by the user. The important and crucial data is encapsulated.

Q.135 What is meant by exceptions? How an exception is handled in C++? Explain with the help of an example. **(4)**

Ans:

Exception Handling: Using exception handling we can more easily manage and respond to run time errors. C++ exception handling is built upon 3 keywords: try, catch and throw.

- The 'try' block contains program statements that we want to monitor for exceptions.
- The 'throw' block throws an exception to the 'catch' block if it occurs within the try block.
- The 'catch' blocks proceeds on the exception thrown by the 'throw' block.

When an exception is thrown, it is caught by its corresponding catch statement which processes the exception. There can be more that one catch statement associated with a try. The catch statement that is used is determined by the type of the exception. An example below illustrates the working of the three blocks.

```
#include<iostream.h>
using namespace std;
int main()
{
  cout<<"\n Start "<<endl;
  try
  {
        cout<<"\n Inside try block "<<endl;
        throw 10;
        cout<<"\n this will not execute ";
  }
  catch(int i)
  {
        cout<<"\n catch number "<<endl;
        cout<<i<<endl;
  }
  cout<<"End";
  return 0;
}
```

Q.136 What do you mean by operator overloading? How unary and binary operators are implemented using the member and friend functions? **(8)**

Ans: When an operator is overloaded it doesn't lose its original meaning but it gains an additional meaning relative to the class for which it is defined. We can overload unary and binary operators using member and friend functions.
When a member operator function overloads a binary operator the function will have only one parameter. This parameter will receive the object that is on right of the operator. The object on the left side is the object that generates the call to the operator function and is passed implicitly by "this".
Overloading a unary operator is similar to overloading a binary operator except that there is only one operand to deal with. When you overload a unary operator using a member function the function has no parameter. Since there is only one operand, it is this operand that generates the call to the operator function.
Overloading operator using a friend function: as we know that a friend function does not have a 'this' pointer. Thus in case of overloading a binary operator two arguments are passed to a friend operator function. For unary operator, single operand is passed explicitly. We cannot use a friend function to overload an assignment operator(=).

Q.137 Explain the importance of using friend function in operator overloading with the help of an example. **(4)**

Ans: We can overload operators using both member function and friend functions. There are certain cases when we prefer using a friend function more than a member function. For examples suppose we wish to pass two different types of operands to the operator function, one of the operands may be an object and the other may be an integer. In case of overloading a binary operator:
A = ob + 4; or A = ob * 2;
The above statement is valid for both a member function and a friend function. But the opposite is not valid for a member function:
A = 4 + ob; or A = 2 * ob;
This statement is valid only in case of a friend function. This is because the left hand operand should be an object as it is used to invoke a member operator function.
The following program illustrates the use of a friend function:

```
#include<iostream.h>
class coord
{
  int x,y;
  public:
  coord()
  {
            x=0;y=0;
  }
```

```
  coord(int i,int j)
  {
        x=i;y=j;
  }
  void get_xy(int &i,int &j)
  {
        i=x;j=y;
  }
  friend coord operator +(int n,coord ob);
}
coord operator +(int n,coord ob)
{
  coord temp;
  temp.x= n + ob.x;
  temp.y= n + ob.y;
  return temp;
}
int main()
{
  coord ob(5,3),obj;
  int x,y,n;
  obj = n + obj;
  obj.get_xy(x,y);
  cout<<"\n x = "<<x<<"\n y = "<<y;
  return 0;
}
```

Q.138 Define a class **Date** with three variables for **day**, **month** and **year**.

a. Write the default and parameterised constructors,
b. Overload the operators <<,>> to read and print Date object,
c. Overload > to compare two dates.
d. Write the destructor that sets values to zero for all three variables. Define object of the class in main and call the above defined functions. **(2+2+3+3+4+2)**

Ans:

```
#include<iostream.h>
#include<conio.h>
class Date
{  int day,month,year;
  public:
a.    Date()
   { day=month=year=0;
   }
   Date(int d,int m,int y)
   { day=d;
```

```
        month=m;
        year=y;
       }
       Date(Date &d1)
       { day=d1.day;
        month=d1.month;
        year=d1.year;
       }
b.      friend istream & operator >> (istream &, Date &);
     friend ostream & operator << (ostream &, Date &);
c.      friend int operator >(Date &, Date &);
d.      ~Date(){day=month=year=0;}
};

istream & operator >> (istream&x, Date&b)
{
   x >> b.day>>b.month>>b.year;
   return(x);
}
ostream & operator << (ostream&dout, Date&b)
{
dout<<b.day<<"/"<<b.month<<"/"<<b.year;

   return(dout);
}
int operator>(Date &d1, Date&d2)
{  if(d1.year>d2.year)
   return 1;
  else if(d1.year<d2.year)
   return 0;
   else
   {  if(d1.month>d2.month)
     return 1;
     else
   if(d1.month<d2.month)
    return 0;
   else
    {if(d1.day>d2.day)
     return 1;
    else
     return 0;
    }
   }
}
```

```
void main()
        {  clrscr();
           Date d1(3,4,2005);
           cout<<d1;
           Date d2;
           cout<<"\n Enter day, month and year:\n";
           cin>>d2;
           cout<<d2;

           if(d1>d2)
            cout<<"\nBigger date is:"<<d1;
           else
             cout<<"\nBigger date is:"<<d2;
             getch();

        }
```

Q.139 Define the class **Student** which has name (char name[20]) and age(int). Define the default constructor, member functions **get_data()** for taking the name and age of the Student, **print()** for displaying the data of Student. **(5)**

Ans:

```
include<iostream.h>
#include<conio.h>
#include<string.h>
#include<fstream.h>

class Student
{
 char name[20];
 int age;

 public:

 Student()
 {
        strcpy(name,"");
        age=0;
 }
 void getdata()
 {
        cout<<endl<<"Enter Name:-> ";
        cin>>name;
        cout<<endl<<"Enter Age:-> ";
        cin>>age;
 }
```

```
 void print()
 {
          cout<<endl<<name<<'\t'<<age;
 }
 friend void sort(Student *s1,int N);
};
```

Q.140 For the above defined class (of Q139) create an array of students of size N and write the friend function sort(Student arr[N]) which sorts the array of Students according to their age. **(6)**

Ans:
```
void sort(Student *s1, int N)
        {
          Student temp;
          for(int i=0;i<N-1;i++)
           for(int j=N-1;j>i;j--)
           {
              if (s1[j].age < s1[j-1].age)
               {  temp=s1[j];
                     s1[j]=s1[j-1];
                     s1[j-1]=temp;

                    }
           }
            cout<<"\n Sorted Data is:\n";
            for(i=0;i<N;i++)
             s1[i].print();
        }
```

Q.141 Create a file namely "STUDENT.DAT" for storing the above objects in sorted order. **(5)**

Ans:
```
void main()
{         Student obj[100];
 int N,i,j;
 clrscr();
 cout<<endl<<"Enter maximum number of records u want to enter:-> ";
 cin>>N;
 for (i=0;i<N;i++)
          obj[i].getdata();
 cout<<endl<<"U have entered following records:";
 for (i=0;i<N;i++)
          obj[i].print();
 sort(obj,N);
 fstream inoutfile;
 inoutfile.open("STUDENT1.DAT",ios::ate|ios::in|ios::out|ios::binary);
```

```
inoutfile.seekg(0,ios::beg);
for(i=0;i<N;i++)
inoutfile.write((char *) &obj,sizeof obj);
getch();

}
```

Q.142 Consider a publishing company that markets both book and audio cassette version to its works. Create a class **Publication** that stores the title (a string) and price(type float) of a publication. Derive the following two classes from the above Publication class: **Book** which adds a page count (int) and **Tape** which adds a playing time in minutes(float). Each class should have get_data() function to get its data from the user at the keyboard.
Write the main() function to test the Book and Tape classes by creating instances of them asking the user to fill in data with get_data() and then displaying it using put_data().
(16)

Ans:

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

class Publication
{
 protected:
  char title[20];
  float price;
 public:
  virtual void get_data(){}
 virtual void put_data(){}
};

class Book:public Publication
{
  int page_count;
  public:
  void get_data();
  void put_data();
};

class Tape:public Publication
{
  float play_time;
  public:
  void get_data();
  void put_data();
};
```

```
void Book::get_data()
{
  cout<<"\nTitle? ";
  cin.getline(title,20);
  cout<<"\nPrice? ";
  cin>>price;
  cout<<"\nPage_count";
  cin>>page_count;
}
void Book::put_data()
{
  cout<<"\nTitle:  "<<title;
  cout<<"\nPrice:  "<<price;
  cout<<"\nPage_count: "<<page_count;
}

void Tape::get_data()
{  cin.get();
  cout<<"\nTitle? ";
  cin.getline(title,20);
  cout<<"\nPrice? ";
  cin>>price;
  cout<<"\nPlay_time?";
  cin>>play_time;
}
void Tape::put_data()
{
  cout<<"\nTitle:  "<<title;
  cout<<"\nPrice:  "<<price;
  cout<<"\nPlay_time:"<<play_time;
}

void main()
{
 clrscr();
 Book book1;
 cout<<"Please enter book details:\n";
 book1.get_data();
 Tape tape1;
 cout<<"Please enter Tape details:\n";
 tape1.get_data();
 Publication* list[2];
 list[0]=&book1;
 list[1]=&tape1;
 cout<<"\n Book details:\n";
 list[0]->put_data();
```

```
cout<<"\n\n Tape Details:\n";
list[1]->put_data();
getch();
}
```

Q.143 Write short notes on the following:

(i) Nested class.
(ii) Scope resolution operator.
(iii) **this** pointer. **(4*3=12)**

Ans:

(i) Nested class:

Inheritance, one of the major features of OOP, is to derive certain properties of the base class in the derived class. A derived class can make use of all the public data members of its base classes.

When one class is defined inside another class, it is known as containership or nesting. It is also one of the inheriting properties. We can thus imply that one object can be a collection of many different objects. Both the concepts i.e. inheritance and containership have a vast difference. Creating an object that contains many objects is different from creating an independent object. First the member objects are created in the order in which their constructors occur in the body. Then the constructor of the main body is executed. An initialization list of the nested class is provided in the constructor. In the following example the arglist is the list of arguments that is to be supplied when a gamma object is defined. These parameters are used to define the members of gamma. The arglist1 and arglist2 are meant for the constructors a and b respectively.

```
class alpha{....};
class beta{....};

class gamma
{
  alpha a;
  beta b;
  public:
  gamma(arglist): a(arglist1), b(arglist2)
  {
        //constructor body
  }
};
```

(ii). Scope Resolution Operator:

The :: (scope resolution) operator is used to qualify hidden names so that we can use them. We 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 local 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 accesses the variable named count declared in global namespace scope.

We also use the class scope operator to qualify class names or class member names. If a class member name is hidden, we can use it by qualifying it with its class name and the class scope operator.

In the following example, the declaration of the variable X hides the class type X, but you can still use the static class member count by qualifying it with the class type X and the scope resolution operator.

```
#include <iostream>
using namespace std;
class X
{
public:
static int count;
};
int X::count = 10;              // define static data member

int main ()
{
int X = 0;                  // hides class type X
      cout << X::count << endl;   // use static member of class X
}
```

(iii) this pointer:

C++ contains a special pointer that is called 'this'. 'this' is a pointer that is automatically passed to any member function when it is called and it is a pointer to the object that generates the call.

```
#include<iostream.h>
#include<string.h>
class inventory
{
  char item[20];
  double cost;
  int on_hand;
  public:
  inventory(char *i,double c,int o)
  {
            strcpy(this→item, i);
```

```
            this->cost=c;
            this->on_hand=o;
  }

  void show()
};
void inventory::show()
{
  cout<<this->item;
  cout<<this->cost;
  cout<<this->on_hand;
}

int main()
{
  inventory ob("Wrench",495,4);
  ob.show();
  return 0;
}
```

Q.144 Define the following terms related to Object Oriented Paradigm:
Encapsulation, Inheritance, Polymorphism, Data Abstraction **(12)**

Ans: The following are the features of Object Oriented Programming:

- **Encapsulation:** The wrapping up of data and functions into a single unit is called encapsulation. The data can only be accessed by the function with which they have been tied up and not by the outside world. It creates an interface between the object's data and the program.
- **Inheritance:** It is the property by which one class can be derived from another class such that it acquires the properties of its base class. Just like a child inherits the properties of its parents in the same way OOP allows us to reuse the members and functions of class to be derived by another class. There are many different kinds of inheritance like multiple inheritance, multilevel inheritance and hierarchical inheritance.

- **Polymorphism:** polymorphism is the property of representing one operation into many different forms. One operation may express different in different situations. The process of making a function or an operator behave in different manners is known as overloading the function or the operator.
- **Data Abstraction:** Abstraction is the method of representing only the essential features and only which is necessarily required by the user. The important and crucial data is encapsulated.

Q.145 Identify the errors in the following code fragment **(4)**

```
            ...
char ch; int vowels = 0, others = 0;
cout<<"enter character";
while((ch>='a' &&ch<=z) ||  (ch>=A && ch<=Z))
{
switch(ch)
{
case a:case A:case e:case E:case I:case I:case e:case O:case u:case U:++vowels:
     break;
default:++others;
}}
cout<<"vowels are:"<<vowels;
cout<<"others are:"<<others;
….
```

Ans: In the given code, there is no cin statement to read the character input 'ch' from the user. The char constants should be enclosed in '' (single quotes).

Q.146 Declare a class to represent bank account of customers with the following data members:Name of the depositor, Account number, Type of account(S for saving and C for Current), balance amount. The class also contains member functions to do the following:
(i) To initialise the data member (ii) To deposit money (iii) To withdraw money after checking the balance (minimum balance is Rs.1000) (iv) To display the data members. **(6)**

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
class bank
{
 char name[25],type;
 long ac_no;
 float bal,amt;
 public:
 class bank *next;
```

```
bank()
{
        strcpy(name," ");
        type=' ' ;
        ac_no=0;
        bal=0;
        next=NULL;
}
void deposit()
{
        cout<<endl<<"Amount to be deposited:-> ";

        cin>>amt;
        bal+=amt;
        cout<<endl<<"Amount deposited successfully";
}
void withdraw()
{
        cout<<endl<<"How much u want to withdraw:-> ";
        cin>>amt;
        if (bal-amt>=1000)
        {
                bal-=amt;
                cout<<endl<<"Withdraw Done";
        }
        else
        {
                cout<<endl<<"Amount exceeds the Available I";
                getch();
        }
}
void showdata()
{
        cout<<endl<<name<<'\t'<<ac_no<<'\t';;
        if (type=='S')cout<<"Saving";
        else if (type=='C')cout<<"Current";
        cout<<'\t'<<bal;

}
int search(long x)
{
        if (ac_no==x)
                return 1;
        else
                return 0;
}
```

```
        void getdata();

};
 void bank::getdata()
        {
                cout<<endl<<"Enter name of Account Holder:-> ";
                cin>>name;
                cout<<endl<<"Enter Account Number:-> ";
                cin>>ac_no;
                rep:
                cout<<endl<<"Enter type of Account S=Saving, C=Current):-> ";
                cin>>type;
                if (type!='S' && type!='C')
                {
                        cout<<endl<<"Wrong Input";
                        getch();
                        goto rep;
                }
                cout<<endl<<"Enter Amount:-> ";
                cin>>bal;
                next=NULL;
        }


void main()
{
 class bank *bptr=NULL,*tmp=NULL,*counter=NULL;
 int i=0,found=0,ch=0;
 long acno=0;

 while (1)
 {
        clrscr();
        cout<<endl<<"1.Add a new customer";
        cout<<endl<<"2.View Customer Details";
        cout<<endl<<"3.Deposite Amount";
        cout<<endl<<"4.Withdraw Amount";
        cout<<endl<<"5.Exit";
        cout<<endl<<"Enter your choice:-> ";
        cin>>ch;
        switch (ch)
        {
        case 1: if (bptr==NULL)
                        {
                                bptr= new bank;
                                if (bptr==NULL)
```

```
                    {
                        cout<<endl<<"Memory Allocation Problem";
                        getch();
                        exit(1);
                    }
                    bptr->getdata();
                }
                else
                {
                    tmp=new bank;
                    if (tmp==NULL)
                    {
                        cout<<endl<<"Memory Allocation problem";
                        getch();
                        exit(1);
                    }
                    tmp->getdata();
                for (counter=bptr;counter->next!=NULL;counter=counter->next);
                    counter->next=tmp;
                }
                break;
case 2:
                cout<<endl<<"Name\tAc. No.\tType\tBalance\n";
                for (counter=bptr;counter!=NULL;counter=counter->next)
                    counter->showdata();
                getch();
                break;
case 3:         cout<<endl<<"Enter the Account Number to Deposite:-> ";
                long acno;
                cin>>acno;
                for (counter=bptr;counter!=NULL;counter=counter->next)
                {
                    found=counter->search(acno);
                    if (found==1)
                    {
                        counter->deposit();
                        getch();
                        break;
                    }
                }
                if (found==0)
                {
                    cout<<endl<<"Account not exists";
                    getch();
                }
                break;
```

```
        case 4:  cout<<endl<<"Enter the Account Number to Withdraw:-> ";

                        cin>>acno;
                        for (counter=bptr;counter!=NULL;counter=counter->next)
                        {
                                found=counter->search(acno);
                                if (found==1)
                                {
                                        counter->withdraw();
                                        break;
                                }
                        }
                        if (found==0)
                        {
                                cout<<endl<<"Account not exists";
                                getch();
                        }
                        break;
        case 5:  exit(1);
        default: cout<<endl<<"Wrong Choice";
        }
 }

 getch();
}
```

Q.147 How is the working of member function different from friend function and a non member function? **(6)**

Ans: A member function, a non –member function and a friend function, all the three are entirely different from each other.

A member function is that which is declared and defined inside the class. It has full access to the private data members of a class. We can directly access the data members without taking the help of the object. It is not the case with a friend function. Although it can access the private data members of the class it has to take the help of the object of that class. A friend function is not the member of the class. When we create an object of a class, memory is not allocated for the friend function. It is defined like a normal function. While calling a friend function we not use object as friend function is not the part of the object. Thus is called like any normal non member function.

A non member function cannot access the private members of a class. It is not declared inside the class definition.

Q.148 Explain a pure virtual function with an example. **(4)**

Ans: A pure virtual function has no definition related to the base class. Only the function prototype is included. To make a pure virtual function the following syntax is used:
virtual return_type function_name(par_list) = 0;
when a class contains atleast one pure virtual function then it is called an abstract class.
The following is an example program:

```
#include<iostream.h>
using namespace std;
class area
{
   double dim1,dim2;
  public:
   void set_area(int d1,int d2)
   {
            dim1=d1;
            dim2=d2;
   }
   void get_dim(double &d1,double &d2)
   {
            d1=dim1;
            d2=dim2;
   }
   virtual double get_area()=0;
};
class rectangle: public area
{
  public:
   double get_area()
   {
            double d1,d2;
            get_dim(d1,d2);
            return d1*d2;
   }
};
class triangle: public area
{
  public:
   double get_area()
   {
            double d1,d2;
            get_dim(d1,d2);
            return 0.5*d1*d2;
   }
};
```

```
int main()
{
   area *p
   rectangle r;
   triangle t;
   r.set_area(3.3,4.5);
   t.set_area(4.0,5.0);
   p=&r;
   cout<<"\n the area of rectangle is "<<p->get_area()<<endl;
   p=&t;
   cout<<"\n area of triangle is "<<p->get_area()<<endl;
   return 0;
}
```

Q.149 Discuss the various situations when a copy constructor is automatically invoked. **(6)**

Ans:

A copy constructor is invoked in many situations:

- When an object is used to initialize another in a declaration statement.
- When an object is passed as a parameter to a function.
- When a temporary object is created for use as a return value of a function.

A copy constructor only applies to initialization. It does not apply to assignment.

```
Class-name(const class name ob)
{
      //body of copy constructor
}
```

When an object is passed to a function a bitwise copy of that objet is made and given to the function parameter that receives the object. However there are cases in which this identical copy is not desired. For example, if the object contains a pointer to allocated memory. The copy will point to same memory as does the original object. Therefore, if the copy makes a change to the content of this memory it will be changed for the original object too. Also when the function terminates the copy will be destroyed causing its destructor to be called.

Q.150 What are the advantages and disadvantages of inline functions? **(6)**

Ans: Inline functions:

Advantage of an inline function is that they have no overhead associated with function call and return statements. This means that inline function can be executed much faster than the normal function.

The disadvantage of an inline function is that if they are too large and called regularly, program grows larger. For this reason, only short functions are declared as inline.

Q.151 Define Rules for operator Overloading. Write a program to overload the subscript operator '[ ]'. **(8)**

Ans:

The rules for operator overloading are as follows:

- New operators are not created. Only the current operators are overloaded.
- The overloaded operator must have at least one operand of user defined type.
- We cannot alter the basic meaning of the operator.
- When binary operators are overloaded through a member function then the left hand operand is implicitly passed as and thus it must be an object.
- Unary operators when overloaded through member functions will not take any explicit argument.

The subscript operator can be overloaded only by means of a member function:

```
#include<iostream.h>
const int size=5;
class arrtype
{
   int a[size];
  public:
   arrtype()
   {
            int i;
            for(i=0;i<size;i++)
                     a[i]=i;
   }
   int operator [](int i)
   {
            return a[i];
   }
};
int main()
{
   arrtype ob;
   int i;
   for(i=0;i<size;i++)
            cout<<ob[i]<<" ";
   return 0;
}
```

Q.152 How is a class converted into another class? Explain with one example. **(8)**

Ans: C++ allows us to convert one class type to another class type. For example

```
obj1 = obj2;
```

The conversion of one class to another is implemented using a constructor or a conversion function. The compiler treats both of them in the same manner. To decide which form to use depends upon where we want the type conversion function to be located in the source file or in the destination class.

```
#include<iostream.h>
using namespace std;
class invent2;
class invent1;
{
        int code;
        int items;
        float price;
  public:
        invent1(int a,int b,float c)
        {
                code=a;
                items=b;
                price=c;
        }
        void putdata()
        {
                cout<<"Code : "<<code<<"\n";
                cout<<"Items : "<<items<<"\n";
                cout<<"Price : "<<price<<"\n";
        }
        int getcode(){return code;}
        int getitems(){return items;}
        float getprice(){return price;}
        operator float(){return (items*price);}
        /*operator invent2()
        {
                invent2 temp;
                temp.code=code;
                temp.value=price*items;
                return temp;
        }*/
};
class invent2
{
        int code;
        float value;
  public:
        invent2()
        {
                code=0;
                value=0;
```

```
        }
        invent2(int x,float y)
        {
                code=x;
                value=y;
        }
        void putdata()
        {
                cout<<"Code : "<<code<<"\n";
                cout<<"Value : "<<value<<"\n\n";
        }
        invent2(invent1 p)
        {
                code=p.getcode();
                value=p.getitems()*p.getprice();
        }
};
int main()
{
        invent1 s1(115,9,178.6);
        invent2 d1;
        float total_value;
        total_value=s1;
        d1=s1;
        cout<<"\n Product details - invent1 type"<<"\n";
        s1.putdata();
        cout<<"\n Stock Value"<<"\n";
        cout<<"\n Value = "<<total_value<<"\n";
        cout<<"\n Protype details - invent2 type"<<"\n";
        d1.putdata();
        return 0;
}
```

Q.153 How does visibility mode control the access of members in the derived class? Explain with an example. **(6)**

Ans:

The visibility mode specified while declaring a derived class plays a very important role in the access control of the data members of the class. The visibility mode is one of the three keywords: public, private and protected. These access specifiers determine how elements of the base class are inherited by the derived class. When the access specifier for the inherited base class is public, all public members of the base class become public members of the derived class. If the access specifier is private, all public members of the base class become private members for the derived class. Access specifier is optional. In case of a derived 'class' it is private by default. In case of a derived 'structure' it is public by default.

The protected specifier is equivalent to the private specifier with the exception that protected members of a base class are accessible to members of any class derived from the base. Outside the base or derived class, protected members are not accessible. The protected specifier can appear anywhere in the class.

When a protected member of a base class is inherited as public by the derived class, it becomes protected member of the derived class. If the base is inherited a private, the protected member of the base becomes private member of the derived class.

Example:

```
#include<iostream.h>
using namespace std;
class one
{
        int x;
  public:
        int y;
        void get_xy();
        int get_x(void);
        void dis_x(void);
};
class two: public one
{
        int z;
  public:
        void prod(void);
        void display(void);
};
void one:: get_xy(void)
{
        x=7;
        y=9;
}
int one:: get_x()
{
        return x;
}
void one::dis_x()
{
        cout<<"x= "<<x<<"\n";
}
void two::prod()
{
        z=y*get_x;
}
void two::display()
{
        cout<<"x= "<<get_x()<<"\n";
```

```
        cout<<"y= "<<y<<"\n";
        cout<<"z= "<<z<<"\n";
}
int main()
{
        two m;
        m.get_xy();
        m.prod();
        m.dis_x();
        m.display();
        d.prod();
        d.display();
        return 0;
}
```

Q.154 What is run time polymorphism? How it is achieved? Explain with an example? **(4)**

Ans: Polymorphism is one of the most useful features of object oriented programming. Polymorphism means to have many forms of one name. it can be achieved both at run time and at compile time.

Polymorphism can also be achieved at run time. This is done using the concept of virtual functions. Which class' function is to be invoked is decided at run time. And then the corresponding object of that class is created accordingly. This is also called as **late binding**. For this concept we always need to make use of pointers to objects.

The example illustrates the concept of virtual functions:

```
#include<iostream.h>
using namespace std;
class base
{
  public:
            int i;
            base(int x)
            {
              i=x;
            }
            virtual void func()
            {
              cout<<"\n using base version of function";
              cout<<i<<endl;
            }
};
```

```
class derived: public base
{
  public:
            derived(int x):base(x){}
            void func()
            {
              cout<<"\n using derived version";
              cout<<i*i<<endl;
            }
};
class derived1: public base
{
  public:
            derived1(int x): base(x){}
            void func()
            {
              cout<<"\n using derived1 version";
              cout<<i+i<<endl;
            }
};
int main()
{
            base *p;
            base ob(10);
            derived d_ob(10);
            derived1 d_ob1(10);
            p=&ob;
            p->func();
            p=&d_ob;
```

```
        p->func();
        p=&d_ob1;
        p->func();
        return 0;
}
```

Q.155 Define a class loan with Principle, Period and Rate as data members. Incr (float) is a member function, which increases the rate of interest by the parameter value. If rate of interest is same for all loans, use appropriate declarations and implement the class. **(6)**

Ans:

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

class loan
{
 float principle,rate;
 int period;

 public:
 loan(float f,int p)
 { principle=f;
   rate=0.15;
   period=p;
 }
 void Incr(int N)
 {
     rate=rate+N;
 }

 float calculatevalue(loan &l)
 {
  int year=1;
  float sum=l.principle;
  if(l.principle>5000)
  l.Incr(1);
  while(year<=period)
  {  sum=sum*(1+rate);
     year=year+1;
          }
  return sum;
}
};
```

```
void main()
{ float amount;
  loan l1(6000,5);

  amount=l1.calculatevalue(l1);
  cout<<"\nAmount="<<amount;

  getch();

}
```

Q.156 Write a program having a base class Student with data member rollno and member functions getnum() to input rollno and putnum() to display rollno.
A class Test is derived from class Student with data member marks and member functions getmarks() to input marks and putmarks() to display marks. Class Sports is also derived from class Student with data member score and member functions getscore() to input score and putscore() to display score. The class Result is inherited from two base classes, class Test and class Sports with data member total and a member function display() to display rollno, marks, score and the total(marks + score). **(12)**

Ans:

```
#include<iostream.h>
#include<conio.h>
using namespace std;
class student
{
  protected:
        int roll_num;
  public:
        void get_num(int a)
        {
           roll_num=a;
        }
        void put_num(void)
        {
           cout<<"Roll number: "<<roll_num<<"\n";
        }
};
class test: public student
{
  protected:
        float marks;
  public:
```

```
        void get_marks(float m)
        {
           marks=m;
        }
        void put_marks(void)
        {
           cout<<"Marks obtained: "<<marks<<"\n";
        }
};
class sports
{
  protected:
        float score;
  public:
        void get_score(float s)
        {
           score=s;
        }
        void put_score(void)
        {
           cout<<"Score: "<<score<<"\n";
        }
};
class result: public test,public sports
{
        float total;
  public:
        void display(void)
        {
           total=marks+score;
           put_num();
           put_marks();
           put_score();
           cout<<"Total score: "<<total<<"\n";
        }
};
void main()
{
        result stu;
        stu.get_num(12);
        stu.get_marks(87.6);
        stu.get_score(5.5);
        stu.display();
        getch();
}
```

Q.157 What will be the result of following expressions when they are executed in sequence? **(4)**

```
…
…
int a = 10;
int b = 20;
c = ++a    +   ++a  +  ++a;
b=b++      +   b++;
e=a++      +   --a     + b--;
f=b--  & ++a  +  b++;
cout<<c<<d<<e<<f;
…
…
```

Ans:
The variables c,d,e,f are not declared. If these variables are declared as int then output of statements will be
39 40 46 1

c=++a + ++a + ++a=13+13+13=39
d=b++ +b++=20+20=40
e=a++ + --a + b--=13+12+21=46
f=b-- & ++a + b++ =1

Q.158 What are generic classes? Why are they useful? Explain with an example how these are implemented in c++. **(8)**

Ans:
Generic class:
When we declare a generic class we are able to define all algorithms used by that class but the actual type data being manipulated will be specified as a parameter when object of that class is created. Generic classes are useful when a class contains generalised logic. For example the algorithm that is used to maintain a queue of integer can also be used for a queue of character. The compiler will automatically generate the correct type of object based upon the type you specify when the object is created. General form generic class:

```
template <class type> class class-name
{
 //body;
}
```

Here the type is place holder type-name that will be specified when the object of the class is created. We can define more than one generic data-type by using comma separated list.
To create an object
Class-name <type> ob;
Member functions of a generic class are themselves automatically generic. They need not be explicitly specified by using template.

Q.159 Explain the following functions(with example) for manipulating file pointers: seekg(),seekp(),tellg(),tellp(). **(8)**

Ans: A file pointer is used to navigate through a file. We can control this movement of the file pointer by ourselves. The file stream class supports the following functions to manage such situations.

- **seekg ():**moves get pointer (input) to a specified location.
- **seekp ():**moves put pointer (output) to a specified location.
- **tellg ():**gives the current position of the get pointer.
- **tellp ():** gives the current position of the put pointer.

For example:

infile.seekg (10);will move the file pointer to the byte number 10.

```
ofstream fileout;
fileout.open("morning", ios::app);
int p = fileout.tellp();
```

the above statements will move the output pointer to the end of the file morning and the value of p will represent the number of bytes in the file.

seekg and seekp can also be used with two arguments as follows.

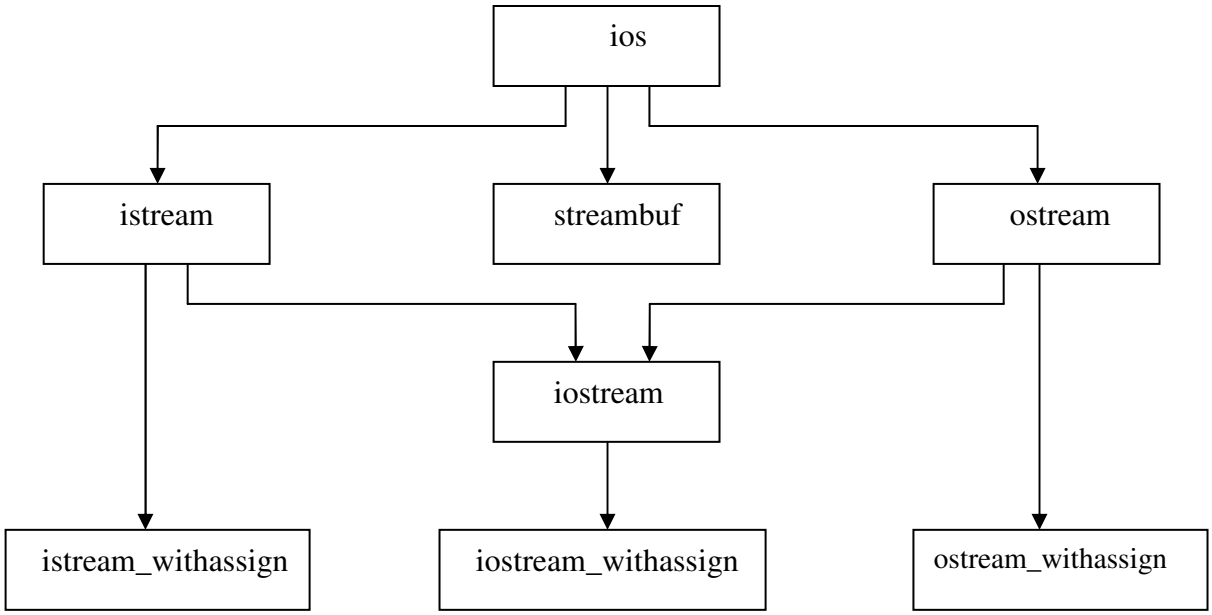
```
seekg (offset, ref position);
seekp (offset, ref position);
```

where offset referes to the number of bytes the file pointer is to be moved from the location specified by the parameter ref position.

Q.160 Explain the following:

i) Stream class hierarchy.
ii) Exception handling.
iii) Abstract class **(4*4=16)**

Ans:
(i)



The above hierarchy of classes are used to define various streams which deals with both the console and disc files. These classes are declared in the header file iostream. Ios is the base for istream, ostream and iostream base classes. Ios is declared as a virtual base class so that only one copy of its members are inherited by the iostream. The three classes istream_withassign, ostream_withassign and iostream_withassign are used to add assignment operators to these classes.

ii) Exception Handling:

using exception handling we can more easily manage and respond to run time errors. C++ exception handling is built upon 3 keywords: try, catch and throw.

- The 'try' block contains program statements that we want to monitor for exceptions.
- The 'throw' block throws an exception to the 'catch' block if it occurs within the try block.
- The 'catch' block proceeds on the exception thrown by the 'throw' block.

When an exception is thrown, it is caught by its corresponding catch statement which processes the exception. there can be more that one catch statement associated with a try. The catch statement that is used is determined by the type of the exception. An example illustrates the working of the three blocks.

```
#include<iostream.h>
using namespace std;
int main()
{
   cout<<"\n Start "<<endl;
   try
   {
            cout<<"\n Inside try block "<<endl;
            throw 10;
            cout<<"\n this will not execute ";
   }
   catch(int i)
   {
            cout<<"\n catch number "<<endl;
            cout<<i<<endl;
   }
   cout<<"End";
   return 0;
}
```

(iii) Abstract class:

When a class contains at least one pure virtual function, it is referred to as an abstract class. An abstract class is something which is to be hidden by people and on which modifications can be done in future. Since an abstract class contains at least one function for which no body exists, technically it can be considered as incomplete and therefore no object is created. Thus we can also define an abstract class as a class for which no object is created. We may conclude that an abstract class exists only to be inherited. We may still create pointer to an abstract class. In the following example, class area is an abstract class because it has a pure virtual function get_area().

```
#include<iostream.h>
using namespace std;
class area
{
   double dim1,dim2;
  public:
   void set_area(int d1,int d2)
   {
            dim1=d1;
            dim2=d2;
   }
   void get_dim(double &d1,double &d2)
   {
            d1=dim1;
            d2=dim2;
   }
   virtual double get_area()=0;
};
```

Q.161 What are the benefits of object oriented programming? Explain. **(8)**

Ans:

Object Oriented Programming (OOP) is an approach that provides a way of modulating programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

The advantages of OOP are as follows:

- Function and data both are tied together in a single unit.
- Data is not accessible by external functions as it is hidden.
- Objects may communicate with each other only through functions.
- It is easy to add new data and functions whenever required.
- It can model real world problems very well.

- Though inheritance we can eliminate redundant code and extend the use of existing classes.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple instances of an object to co-exist without any interference.
- The data centric design approach enables us to capture more details of a model in implemented form.
- It is possible to map objects in the problem domain to those in the program.
- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
- Software complexity can be managed.

Q.162 Explain Inline functions and the situations where inline expansion may not work and why? **(8)**

Ans: Inline functions: Whenever we write functions, there are certain costs associated with it such as jumping to the function, saving registers, pushing arguments into the stack and returning to the calling function. To remove this overhead we write an inline function. It is a sort of request to the compiler. If we declare and define a function within the class definition then it is automatically inline. We do not need to sue the term inline.

Advantage of inline functions is that they have no overhead associated with function call or return mechanism.

The disadvantage is that if the function made inline is too large and called regularly our program grows larger.

There are certain conditions in which an inline function may not work:

- If a function is returning some value and it contains a loop, a switch or a goto statement.
- If a function is not returning a value and it contains a return statement.
- If a function contains a static variable.
- If the inline function is made recursive.

Q.163 Define a class Coord having two members type int as x and y. Use this class to define another class Rectangle which has two members of type Coord as UpperLeftCoord and BottomRightCoord. Define the constructors and member functions to get the length and breadth of rectangle. Write a global function which creates an instance of the class Rectangle and computes the area using the member functions. **(10)**

Ans:

```
#include <iostream.h>
 #include<conio.h>
#include<math.h>
 class coord {
    int X;
    int Y;
 public:
   coord() {X=Y=0;}
   coord(int x, int y)
```

```
    { X=x; Y=y;}
    int getX(){ return X;}
    int getY(){return Y;}
    //int getZ(){return Z;}
  };

  class rectangle
  {
    coord UpperLeftCoord;
    coord BottomRightCoord;
    public:
     rectangle(){}
     rectangle(coord &p1,coord &p2)
     { UpperLeftCoord=p1;
       BottomRightCoord=p2;
     }
     void showdata()
     {
  cout<<"UpperLeftCoordinate="<<"("<<UpperLeftCoord.getX()<<","<<UpperLeftCoord.getY(
  )<<")";
  cout<<"\nBottomLeftCoordinate="<<"("<<BottomRightCoord.getX()<<","<<BottomRightCoo
  rd.getY()<<")";
     }

  int getLength()
    { return(abs(UpperLeftCoord.getX()- BottomRightCoord.getX()));
    }
    int getBreath()
    { return(abs(UpperLeftCoord.getY()- BottomRightCoord.getY()));
    }
    int Area()
    {  return (getLength()*getBreath());
 }
  };

  void main()
  {  clrscr();
    coord c1(2,4);
    coord c2(6,2);
    rectangle r1(c1,c2);
       r1.showdata();
    int f=r1.Area();
  cout<<"\nArea="<<f;
    getch();
  }
```

Q.164 What is the output of following codes:

(i)
```
void main()
{
int a[] = {10,20,30,40,50};
int i;
for (i=0;i<5;i++)
{
cout<<"\nelement is "<<*(i+a)<<" "<<*(a+i)<<" "<<i[a]<<" "<<a[i];
}
}
```

(ii)
```
void main()
{
char *ptr = "xyzw";
char ch;
ch = ++ *ptr ++;
cout <<ch;
cout << ch++;
}
```
(4+4)

Ans:

(i) output:

element is10 10 10 10
element is20 20 20 20
element is30 30 30 30
element is40 40 40 40
element is50 50 50 50

(ii) output:

yy

Q.165 Write a C++ program using classes and objects to simulate result preparation system for 20 students. **(8)**

The data available for each student includes:
Name (includes first, mid and last name each of size 20 characters),
Rollno,
Marks in 3 subjects.
The percentage marks and grade are to be calculated from the following information

Percentage marks	**grade**
<50	'F'
≥50<60	'D'
≥60<75	'C'
≥75<90	'B'
≥90<100	'A'

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<process.h>
class students
{                       struct name
                        {
                                char first[20],mid[20],last[20];
                        };
                        struct name e[20];
                        int rollno[20],marks[20][3],counter;
                        int per[20];
                        char grade[20];
                        public:
                        students();
                        void getdata();
                        void calculate();
                        void showdata();
};
students :: students()
{                       counter=0;
                        for (int i=0;i<20;i++)
                        {                       strcpy(e[i].first,"");
                                strcpy(e[i].mid,"");
                                strcpy(e[i].last,"");
                                rollno[i]=0;
                                for (int j=0;j<3;j++)
                                        marks[i][j]=0;
                                per[i]=0;
                                grade[i]=' ';
                        }}
```

```
void students :: getdata()
{
                    cout<<endl<<"Enter First Name of the Student:-> ";
                    cin>>e[counter].first;
                    cout<<endl<<"Enter Middle Name of the Student:-> ";
                    cin>>e[counter].mid;
                    cout<<endl<<"Enter Last Name of the Student:-> ";
                    cin>>e[counter].last;
                    cout<<endl<<"Enter Roll Number of the student:-> ";
                    cin>>rollno[counter];
                    cout<<endl<<"Enter the Marks in Subject 1;-> ";
                    cin>>marks[counter][0];
                    cout<<endl<<"Enter the Marks in Subject 2;-> ";
                    cin>>marks[counter][1];
                    cout<<endl<<"Enter the Marks in Subject 3;-> ";
                    cin>>marks[counter][2];
                    per[counter]=(marks[counter][0]+marks[counter][1]+marks[counter][2])/3;

                    if (per[counter]>=90 && per[counter]<=100)
                            grade[counter]='A';
                    else if (per[counter]>=75)
                            grade[counter]='B';
                    else if (per[counter]>=60)
                            grade[counter]='C';
                    else if (per[counter]>=50)
                            grade[counter]='D';
                    else
                            grade[counter]='F';
                    counter++;

}

void students :: showdata()
{
                    cout<<endl<<"First\tMiddle\tLast   Marks1   Marks2   Marks3   Percentage
                    Grade\n";
                    for (int i=0;i<counter;i++)
                    {
                            cout<<endl<<e[i].first;
                            cout<<' '<<e[i].mid;
                            cout<<' '<<e[i].last;
                            cout<<' '<<rollno[i];
                            cout<<' '<<marks[i][0];
```

```
                        cout<<' '<<marks[i][1];
                        cout<<' '<<marks[i][2];
                        cout<<' '<<per[i];
                        cout<<' '<<grade[i];
                }
}
void main()
{               students obj1;
                int i,n;
                cout<<endl<<"Enter how many records u want to enter:-> ";
                cin>>n;
                if (!(n<=20))
                {
                        cout<<endl<<"Maximum number of records can be 20\n";
                        getch();
                        exit(1);
                }
                for (i=0;i<n;i++)
                        obj1.getdata();
                cout<<endl<<"Records added successfully.";
                obj1.showdata();
                getch();
}
```

Q.166 Is it possible that a function is friend of two different classes? If yes, then how it is implemented in C++. **(8)**

Ans: Yes, it is possible that a function can be a friend of two classes. Member functions of one class can be friend with another class. This is done by defining the function using the scope resolution operator.

We can also declare all the member functions of one class as the friend functions of another class. In fact we can also declare all the member functions of a class as friend. Such a class is then known as friend class.

The following program shows how a friend function can be a friend of two classes:

```
#include<iostream.h>
using namespace std;
class one;
class two
{
                int a;
  public:
                void setvalue(int n){a=n;}
                friend void max(two,one);
};
```

```
class one
{
                int b;
  public:
                void setvalue(int n){b=n;}
                friend void max(two,one);
};
void max(two s,one t)
{
                if(s.a>=t.b)
                        cout<<s.a;
                else
                        cout<<t.b;
}
int main()
{
                one obj1;
                obj1.setvalue(5);
                two obj2;
                obj2.setvalue(10);
                max(obj2,obj1);
                return 0;
}
```

Q.167 Write a program to overload << and >> operator to I/O the object of a class. **(8)**

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<<endl<<"c1= "<<c1<<endl<<"c2= "<<c2;
        cout<<endl<<"Enter a complex number: ";
        cin>>c3;
        cout<<"c3= "<<c3;
}
```

Q.168 What is the difference between a virtual function and a pure virtual function? Give example of each. **(8)**

Ans: Virtual functions are important because they support polymorphism at run time. A virtual function is a member function that is declared within the base class but redefined inside the derived class. To create a virtual function we precede the function declaration with the keyword virtual. Thus we can assume the model of "one interface multiple method". The virtual function within the base class defines the form of interface to that function. It happens when a virtual function is called through a pointer.

The following is an example that illustrates the use of a virtual function:

```
#include<iostream.h>
using namespace std;
class base
{
  public:
  int i;
  base(int x)
  {
        i=x;
  }
  virtual void func()
  {
        cout<<"\n using base version of function";
        cout<<i<<endl;
  }
};
class derived: public base
{
  public:
  derived(int x):base(x){}
```

```
  void func()
  {
        cout<<"\n sing derived version";
        cout<<i*i<<endl;
  }
};
class derived1: public base
{
  public:
  derived1(int x): base(x){}
  void func()
  {
        cout<<"\n using derived1 version";
        cout<<i+i<<endl;
  }
};

int main()
{
  base *p;
  base ob(10);
  derived d_ob(10);
  derived1 d_ob1(10);
  p=&ob;
  p->func();
  p=&d_ob;
  p->func();
  p=&d_ob1;
  p->func();
  return 0;
}
```

A pure virtual function has no definition related to the base class. Only the function prototype is included to make a pure virtual function the following syntax is used:

```
virtual return_type function_name(par_list) = 0;
```

The following is an example program:

```
#include<iostream.h>
using namespace std;
class area
{
  double dim1,dim2;
  public:
  void set_area(int d1,int d2)
  {
        dim1=d1;
        dim2=d2;
  }
```

```
  void get_dim(double &d1,double &d2)
  {
        d1=dim1;
        d2=dim2;
  }
  virtual double get_area()=0;
};
class rectangle: public area
{
  public:
  double get_area()
  {
        double d1,d2;
        get_dim(d1,d2);
        return d1*d2;
  }
};

class triangle: public area
{
  public:
  double get_area()
  {
        double d1,d2;
        get_dim(d1,d2);
        return 0.5*d1*d2;
  }
};
int main()
{
  area *p
  rectangle r;
  triangle t;
  r.set_area(3.3,4.5);
  t.set_area(4.0,5.0);
  p=&r;
  cout<<"\n the area of rectangle is "<<p->get_area()<<endl;
  p=&t;
  cout<<"\n area of triangle is "<<p->get_area()<<endl;
  return 0;
}
```

Q.169 Write a program for Conversion from Basic to Class Type. **(8)**

Ans:

```
#include<iostream.h>
class time
{
  int hours;
  int minutes;
  public:
  time(){}
  time(int t)
  { hours=t/60;
    minutes=t%60;
  }
  void showtime()
  { cout<<hours<<"hrs "<<minutes<<"min";
  }
};
void main()
{
  time t1;
  int duration=90;
  t1=duration;
  t1.showtime();
}
```

Q.170 What is the output of following programs: **(8)**

(i)
```
#include<iostream.h>
int global = 10;
void func(int & x, int y)
{
x = x – y;
y = x * 10;
cout<<x<<“\t”<< y<< “\n”;
}
void main()
{
int global = 7;
func(::global, global);
cout<<global<<“\t”<<::global<<“\n”;
func(global,::global);
cout<<global<<“\t”<<::global<<“\n”;
}
```

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

```
{
int i, j, m;
int a[5]={8, 10, 1, 14, 16}
i = ++a[2];
m = a[i++];
cout <<i<<m;
}
```

Ans:

i) 3 30
7 3
4 40
4 3
ii) 3 2

Q.171 Write a function template for sorting a list of arrays. **(8)**

Ans:

```
#include<iostream.h>
using namespace std;
template<class T>

void sort(T a[],int n)
{
  for(int i=0;i<n-1;i++)
        for(int j=i+1;j<n;j++)
        {       if(a[i]>a[j])
                        swap(a[i],a[j]);
        }
}
template<class X>
void swap(X &a,X &b)
{
  X temp=a;
  a=b;
  b=temp;
}
int main()
{
  int v[5]={32,78,12,98,56};
  sort(v,5);
  cout<<"\n the list of sorted array is :- ";
  for(int i=0;i<5;i++)
        cout<<"\n"<<v[i];
  return 0;
}
```

Q.172 How does inheritance influence the working of constructor and destructor?
Given the following set of definitions

```
Class x
{
};
class y: public x
{
};
class z: public y
{
};
z obj;
```

What order will the constructor and destructor be invoked? **(8)**

Ans: Inheritance has a great deal of influence over the working of constructors and destructors. In case of inheritance the base class constructor is invoked first and then the derived class constructor. In case of multiple inheritance, the constructors of base classes are executed in the order in which they occur in the declaration in the derived class. Similarly in case of multilevel inheritance, the base class constructors are executed in the order of their inheritance. Destructors in case of inheritance are executed exactly in the opposite order in which constructors are executed. Likewise in case of multilevel inheritance destructors are executed in the reverse of that is, from derived to base.

```
class x
{
};
class y: public x
{
};
class z: public y
{
};
z obj;
```

In the above definitions, the contructor will be fired in the following sequence-
class x → class y → class z
similarly, destructors will be fired exactly in the reverse sequence-
class z → class y → class x

Q.173 What is meant by a constant member function? How is it declared? Give an example. **(8)**

Ans: Constant Member functions : A const member function is that which does alter the value of any data in the program. It is declared as follows:

```
void area(int, int) const;
float dis() const;
```

when a const function tries to alter any data ,the compiler generates an error message.

Q.174 Write a program to create a database of the students information such as name, roll no and the program should have the following facilities. **(10)**

i) Adds a new records to the file.
ii) Modifies the details of an record.

Displays the contents of the file.

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<fstream.h>
#include<stdlib.h>
#include<iomanip.h>

class Students
{
 int rollno;
 char name[25];
 public:
   void getdata()
        {
                cout<<endl<<"Enter Name:-> ";cin>>name;
                cout<<endl<<"Enter Roll Number:-> ";cin>>rollno;
        }
        void showdata()
        {
                cout<<setw(10)<<rollno<<setw(10)<<name<<endl;
        }
};
void main()
{
 clrscr();
 int r;
 fstream file;
 file.open("Student.TXT",ios::ate|ios::in | ios::out|ios::binary);
 file.seekg(0,ios::beg);
 Students obj;
 char ch;
 while (1)
 {
 cout<<endl<<"1.Add\n2.Modify\n3.Display\n4.Exit\nEnter your choice:-> ";
        cin>>ch;
        switch (ch)
        {
        case '1':
           obj.getdata();
```

```
                file.write((char *) & obj, sizeof(obj));
                break;
            case '2':
                int last = file.tellg();
                int n = last/sizeof(obj);
                cout<<"number of objects = "<< n << '\n';
                cout<<"total bytes in the file = "<<last<<'\n';
 //modify the details of an item
                cout<<"enter object no. to b updtd \n";
                int object;
                cin>>object;
                cin.get(ch);
                int location = (object - 1)*sizeof(obj);
                if(file.eof())
                file.clear();
                file.seekp(location);
                cout<<"enter new values of object \n";
                obj.getdata();
                cin.get(ch);
                file.write((char *)&obj,sizeof obj)<<flush;
                file.seekg(0);
                break;
            case '3':
                 file.seekg(0);
                 while(file.read((char *) &obj,sizeof obj))
                        obj.showdata();
                 break;
            case '4': exit(0);
            default:
                                cout<<endl<<"Input out of reange.";
            }
 }
}
```

Q.175 Write a program to read three numbers x, y and z and evaluate R given by

$$R = z/(x - y)$$

Use exception handling to throw an exception in case division by zero is attempted. **(8)**

Ans:

```
#include<iostream.h>
  using namespace std;
  int main()
  {
    int x,y,z,R;
    cout<<"\n Enter three values x,y and z : ";
```

```
        cin >> x >> y >> z;
        int a=(x-y);
        try
        {
                if(a!=0)
                {
                        R=z/a;
                        cout<<"Result(z/(x-y)) = "<< R << "\n";
                }
                else
                        throw(a);
        }
        catch(int i)
        {
                cout<<"\n Exception caught : (x-y)" << i << "\n";

        }
        cout << "END";
}
```