BSc.(Information Technology)
(Semester II)
2019-20

Object Oriented Programming
(USIT 201 Core)
ATKT Paper Solution

By

Dr. Amita Jain

Question 1 Attempt any three of the following.

Q1a. What are the advantages and disadvantages of procedure oriented language?

Ans. Advantages of Procedure Oriented Languages:

Less Storage: The most important advantage with procedural/imperative programming is the reduced amount of storage during runtime. Once you start overloading methods and using interfaces, your code will need extra storage to track virtual method tables and other kinds of mapping. This needs to be stored somewhere within your project.

Simplicity: Another advantage is the use of the conditionals, loops and functions by which we can avoid the complexity of OOP. No need to think about classes and inheritance and polymorphism and so on.

Disadvantages of Procedure Oriented Languages:

Data are not secured: Data can be accessed by any function, if it is declared globally. Hence it will be available in all the functions of that program. Any function can change the value of that variable.

Emphasis is on doing things (algorithms): POP doesn't provide any mechanism for data abstraction to hide the data which are not required. It doesn't provide any mechanism to re-use the shared information that is available to all.

Difficult to visualize a very large program as a single concept: Programmers realize the limitations of POP when POP programs were used at a very large scale. Lengthy POP programs were used at a very large scale. Lengthy POP programs lose a 'Global View' and become very difficult to visualize as a single concept.

Q1b. List and explain applications of object oriented programming?

Ans: **Application of OOPS:**

• **User-interface design**: Another popular application of OOP has been in the area of designing graphical user interfaces such as Windows. C++ is mainly used for developing user-interfaces.

• **Simulations and Modeling:** Simulation is the technique of representing the real world entities with the help of a computer program. Simula-67 and Smalltalk are two object-oriented languages are designed for making simulations.

• **Scripting:** In recent years, OOP has also been used for developing HTML, XHTML and XML documents for the Internet. Python, Ruby and Java are the scripting languages based on object-oriented principles which are used for scripting.

• **Object Databases:** These days OOP concepts have also been introduced in database systems to develop a new DBMS named object databases. These databases store the data directly in the form of objects. However, these databases are not as popular as the traditional RDBMS.

• **Developing computer games:** OOP is also used for developing computer games such as Diablo, Startcraft and Warcraft III. These games offer virtual reality environments in which a number of objects interact with each other in complex ways to give the desired result.

Some other areas of applications include Artificial Intelligence (AI) and expert systems, office automation systems, decision support systems (DSS), Neural networks and parallel programming, and Computer-Aided Design (CAD) systems.

Q1c. Write a c++ program to add two numbers and numbers to be inputted by the user.

Ans.

```
 include <iostream>

using namespace std;

int main()

{

  int a, b, c;

 cout << "Enter two integers to add\n";

  cin >> a >> b;

  c = a + b;

  cout <<"Sum of the numbers: " << c << endl;

  return 0;

}
```

Q1d. Describe the concept of inheritance and data abstraction in OOP language.

Ans. **Inheritance:** Inheritance is a mechanism for sharing similarities among classes while preserving their differences. In the OO terminology, the individual pieces of data are known as *attributes* while the operations that objects carry out are known as *methods*. In OOP, the concept of inheritance provides the idea of *reusability*. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined feature of both the classes. The real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class i.e almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduced any undesirable side-effects into the rest of classes.

Data Abstraction:- Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and functions to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created. Since the classes use the concept of data abstraction, they are known as abstract data types(ADT)

Q1e. Explain benefits of object oriented programming.

Ans. Some benefits of object oriented programming are:

- Since everything is treated as objects, so we can model a real-world concept using OOP.
- The programmes written with OOP are really easy to understand.
- OOP approach offers the reusability of classes. We can reuse the classes that are already created without writing them again and again.
- Since the parallel development of classes is possible in OOP concept, It results in the quick development of the complete programmes.
- Programmes written in OOP technique are marginally easier to test, manage as well as maintain.
- It is a secured development technique since data is hidden and can't be accessed by external functions.

Q1f. Write a c++ program to find out area of a circle.

Ans.
```
#include<iostream>
using namespace std;

int main()
{
        float r,area;
        cout<< "\nEnter radius of circle : ";

        cin>>r;
        area = 3.14*r*r;

        cout<<"Area of circle : "<<area;



        return 0;
}
```

Q2a. Write a short note on class decleration in c++.

Ans. A class is a user-defined data type that binds data and functions that operate on the data together in a single unit. It allows the data to be hidden from external use. Like other user-defined data types, it also needs to be defined before using its objects in the program. A class definition specifies a new data type that can be treated as built-in data type.

The general form of a class declaration is:
```
 class className
{
        private :
                variable declarations;
                function declarations;
        public:
                variable declarations;
                function declaration;
```

```
      protected:
              variable declarations;
              function declaration;
  };
```

Where
class, private, public and protected are C++ keywords
className is the name of the class
variable declarations are the variables (*data members*) of the class
function declarations are *member functions* of the class that defines the behaviour of the class

The variables and functions declared within the curly braces are collectively known as members of the class.

Q2b. Explain the importance of friend classes with example.

A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example a LinkedList class may be allowed to access private members of Node.

```
// friend class
// friend functions
#include <iostream.h>
#include<conio.h>

class FriendCLS1
{
   int a, b;
  public:
   void acceptdata()
   {
        cout << "\nEnter two numbers: ";
        cin>>a>>b;
   }
   void putdata()
   {
        cout << "\na= "<<a<<"\nb= "<<b;
   }
   friend class FriendCLS2;
};
class FriendCLS2
{
  public:
   void sum (FriendCLS1 c1)
   {
        cout << "\nsum= "<<c1.a+c1.b;
   }
};

void main ()
{
  clrscr();
  FriendCLS1 c1;
  c1.acceptdata();
  cout<<"\nObject c1 data";
  c1.putdata();
  cout<<"\nSum of data members of object1";
  FriendCLS2 c2;
  c2.sum(c1);
  getch();
}
```

Q2c. Explain default constructor with example.

Ans. If the programmer does not specify the constructor in the program then compiler provides the default constructor. In C++ we can overload the default compiler generated constructor. In both cases (user created default constructor or default constructor generated by compiler), the default constructor is always parameter less.

Syntax

```
class_name() {
-----
-----
}
```

Example of Default Constructor

Let us take the example of class MarksFY which contains the marks of two subjects DM and WP.

```
#include<iostream>
using namespace std;

class MarksFY
{
public:
   int DM;
   int WP;

   //Default Constructor
   MarksFY() {
      DM=0;
      WP=0;
   }

   display() {
      cout << "DM MARKS :  " << DM <<endl;
      cout << "WP MARKS :" << WP << endl;
   }
};

int main()
{
  //invoke Default Constructor
  MarksFY m;
  m.display();
  return 0;
}
```

Output :

```
DM :  0
```

```
WP : 0
```

Q2d. Write a short note on destructor in C++.

Ans. Destructors are special member functions of the class required to free the memory of the object whenever it goes out of scope. Destructors are parameter less functions. Name of the Destructor should be exactly same as that of name of the class. But preceded by '~' (tilde). Destructors does not have any return type. Not even void.

The Destructor of class is automatically called when object goes out of scope.

For Example

```
#include<iostream>
using namespace std;

class MarksFY
{
public:
  int DM;
  int WP;

  //constructor
  MarksFY() {
    cout << "Inside Constructor"<<endl;
    cout << "C++ Object created"<<endl;
  }

  //Destructor
  ~MarksFY() {
    cout << "Inside Destructor"<<endl;
    cout << "C++ Object destructed"<<endl;
  }
};

int main( )
{
  MarksFY m1;
  MarksFY m2;
  return 0;
}
```

Output

```
Inside Constructor
C++ Object created
Inside Constructor
C++ Object created

Inside Destructor
C++ Object destructed
Inside Destructor
C++ Object destructed
```

***Q2e. Design an employee class for reading and displaying the employee information. The getInfo( ) and displayInfo( )* methods will be used respectively, where getInfo( ) will be private method.**

Ans. 1.

```
#include<iostream.h>
#include<conio.h>
 class employee
{
  char name[20];
  int age;
  float basic_sal;
  void getInfo()
  {
   cout<<endl<<"Enter name: ";
   cin>>name;
   cout<<endl<<"Enter age: ";
   cin>>age;
   cout<<endl<<"Enter basic salary: ";
  cin>>basic_sal;
  }
 public:
void displayInfo()
  {
    getInfo();
    cout<<endl<<"\tEMPLOYEE INFORMATION\n";
cout<<"\t--------------------";
    cout<<endl<<"  Name: "<<name;
  cout<<endl<<"  Age: "<<age;
cout<<endl<<"  Basic Salary: "<<basic_sal;
cout<<endl<<"  Gross Salary: "<<basic_sal + (0.6*basic_sal) + (0.4*basic_sal);
   }
};
void main()
{
 clrscr();
 employee e;
 e.displayInfo();
getch();
}
```

Output:

```
Enter name: AMITA
 Enter age: 33
Enter basic salary: 8000
EMPLOYEE INFORMATION --------------------
 Name:AMITA
Age: 33
Basic Salary: 8000
 Gross Salary: 16000
```

Q2f. Write a short note on scope resolution operator.

Ans. In C++, scope resolution operator is ::. It is used for following purposes.
1) To access a global variable when there is a local variable with same name:

```
/* C++ program to show that we can access a global variable using scope resolution operator
:: when there is a local variable with same name  */
#include<iostream>
using namespace std;

int x;  // Global x

int main()
{
  int x = 10; // Local x
  cout << "Value of global x is " << ::x;
  cout << "\nValue of local x is " << x;
  return 0;
}
```

utput:

```
Value of global x is 0

Value of local x is 10
```

2) To define a function outside a class.

```
// program to show that :: is used  to define a function outside a class
#include<iostream>
using namespace std;

class A
{
public:

   // Only declaration
   void fun();
};

// Definition outside class using ::
void A::fun()
{
   cout << "fun() called";
}

int main()
{
   A a;
   a.fun();
   return 0;
}
```

Output:

```
fun() called
```

3) To access a class's static variables.

```
//:: can be used to access static members when there is a local variable with same name
#include<iostream>
using namespace std;

class Test
{
    static int x;
public:
    static int y;

    // Local parameter 'a' hides class member  'a', but we can access it using ::
    void func(int x)
    {
       // We can access class's static variable
       // even if there is a local variable
       cout << "Value of static x is " << Test::x;

       cout << "\nValue of local x is " << x;
    }
};

// In C++, static members must be explicitly defined like this
int Test::x = 1;
int Test::y = 2;

int main()
{
    Test obj;
    int x = 3 ;
    obj.func(x);

    cout << "\nTest::y = " << Test::y;

    return 0;
}
```

Output:

```
Value of static x is 1
Value of local x is 3
Test::y = 2;
```

4) In case of multiple Inheritance:

If same variable name exists in two ancestor classes, we can use scope resolution operator to distinguish.

```
// Use of scope resolution operator in multiple inheritance.
#include<iostream>
using namespace std;

class A
{
protected:
    int x;
public:
    A() { x = 10; }
};

class B
{
protected:
    int x;
public:
    B() { x = 20; }
};

class C: public A, public B
{
public:
  void fun()
  {
    cout << "A's x is " << A::x;
    cout << "\nB's x is " << B::x;
  }
};

int main()
{
   C c;
   c.fun();
   return 0;
}
```

Output:

```
A's x is 10
B's x is 20
```

5) For namespace

If a class having the same name exists inside two namespace we can use the namespace name with the scope resolution operator to refer that class without any conflicts

```
// Use of scope resolution operator for namespace.

#include<iostream>

int main(){
   std::cout << "Hello" << std::endl;

}
```

Here, cout and endl belong to the std namespace.

6) Refer to a class inside another class:

If a class exists inside another class we can use the nesting class to refer the nested class using the scope resolution operator

```
// Use of scope resolution class inside another class.
#include<iostream>
using namespace std;

class outside
{
public:
    int x;
    class inside
    {
    public:
        int x;
        static int y;
        int foo();

    };
};
int outside::inside::y = 5;
int main(){
   outside A;
   outside::inside B;

}
```

Q3a. Define operator overloading. Explain the set of rules required for overloading operators in C++.

Ans. We can redefine or overload most of the built-in operators available in C++. Thus, a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Rules for Operator Overloading:-

- The operators which are predefined by the C++ can only be overloaded.
- Operator can be unary or binary.
- Operator functions must be a member functions or friend functions.
- A friend function will have only one argument for unary operators and two for binary operators
- A member function has no argument for unary operators and one for binary operators.
- The original meaning of the operator should not be changed.

Q3b. Write a C++ program to overload the operator unary(-) for negating the elements of a matrix.

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

#include<conio.h>

class MatrixNew{

  int a[3][3];

public:

  void getValues(){

   cout<<"\nEnter values :\n";

   for(int i=0;i<3;i++)

    for(int j=0;j<3;j++)

      cin>>a[i][j];

  }

  void display(){

   for(int i=0;i<3;i++)
   {
    for(int j=0;j<3;j++)
      cout<<a[i][j]<<"  ";
    cout<<endl;
   }
  }
  void operator -(){
   for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      a[i][j]=-a[i][j];
  }
};
void main(){
clrscr();
 MatrixNew m1;
 cout<<"\nFor Oriinal Matrix";
 m1.getValues();
 cout<<"\nOriginal Matrix :\n";
 m1.display();


 cout<<"\n Negated Matrix   :\n";
 -m1;
 m1.display();
 getch();
}
```

Q3c. Write a short note on virtual function with example.

Ans. A virtual function is a member function which is declared in base class and is re-defined (Overridden) by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at Run-time.

```
// CPP program to illustrate  concept of Virtual Functions
#include<iostream>
using namespace std;

class base
{
public:
    virtual void print ()
    { cout<< "print base class" <<endl; }

    void show ()
    { cout<< "show base class" <<endl; }
};

class derived:public base
{
public:
    void print ()
    { cout<< "print derived class" <<endl; }

    void show ()
    { cout<< "show derived class" <<endl; }
};

int main()
{
    base *bptr;
    derived d;
    bptr = &d;

    //virtual function, binded at runtime
    bptr->print();

    // Non-virtual function, binded at compile time
    bptr->show();
}
```

Output:

```
print derived class
show base class
```

Q3d. What is static Function? Explain with an example.

Ans. A static member function is a special member function, which is used to access only static data members, any other normal data member cannot be accessed through static member function. Just like static data member, static member function is also a class function; it is not associated with any class object.

```
include <iostream>

using namespace std;

class Demo

{

        private:

                //static data members

                static int X;

                static int Y;

        public:

        //static member function

        static void  Print()

        {

                cout <<"Value of X: " << X << endl;

                cout <<"Value of Y: " << Y << endl;

        }

};

//static data members initializations

int Demo :: X =10;

int Demo :: Y =20;

int main()

{

        Demo OB;

        //accessing class name with object name

        cout<<"Printing through object name:"<<endl;

        OB.Print();

        //accessing class name with class name

        cout<<"Printing through class name:"<<endl;

        Demo::Print();

        return 0;

}
```

Q3e. Write a C++ program to design a class Geometry containg the methods area( ) and volume ( ) and also overload the area ( ) function.

Ans.
```
#include<iostream.h>
#include<conio.h>
class geometry
 {
 int l,b;
 public: int area(int x)
 {
 l=b=x;
 return(l*b);
 }
 int area(int x, int y)
{
 l=x;
 b=y;
return(l*b);
 }
 int volume(int x)
 {
 l=x;
return(l*l*l);
 }
 };
 void main()
{
clrscr();
 geometry g;
 cout<<"\nArea of square= " <<g.area(10);
cout<<"\nArea of rectangle= "<<g.area(10,15);
cout<<"\nVolume of  cube= "<<g.volume(6);
getch();
}
```
Output:

```
Area of square= 100
Area of rectangle= 150

Volume of cube= 216
```

Q3f. Write short note on type conversion.

Ans. A type cast is basically a conversion from one type to another. There are two types of type conversion:

Implicit Type Conversion Also known as ‘automatic type conversion’.

Done by the compiler on its own, without any external trigger from the user.

Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.

All the data types of the variables are upgraded to the data type of the variable with largest data type.

bool -> char -> short int -> int ->

unsigned int -> long -> unsigned ->

long long -> float -> double -> long double

It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Example of Type Implicit Conversion:

```
#include <iostream>

using namespace std;

  int main()

{

    int x = 10; // integer x

    char y = 'a'; // character c

      // y implicitly converted to int. ASCII

    // value of 'a' is 97

    x = x + y;


    // x is implicitly converted to float

    float z = x + 1.0;


    cout << "x = " << x << endl
```

```
        << "y = " << y << endl
        << "z = " << z << endl;
    return 0;
}
```

```
Output:
x = 107
y = a
z = 108
```

Explicit Type Conversion: This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

In C++, it can be done by two ways:

Converting by assignment: This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

Syntax:

(type) expression

where type indicates the data type to which the final result is converted.

Example:

```
// C++ program to demonstrate
// explicit type casting

#include <iostream>
using namespace std;
  int main()
{
    double x = 1.2;
      // Explicit conversion from double to int
    int sum = (int)x + 1;
      cout << "Sum = " << sum;
      return 0;
}
```

Output:

Sum = 2

Conversion using Cast operator: A Cast operator is an unary operator which forces one data type to be converted into another data type.

Example:

```
#include <iostream>

using namespace std;

int main()

{

    float f = 3.5;

      // using cast operator

    int b = static_cast<int>(f);

      cout << b;

}
```

Output:

```
3
```

Q4a. Explain the concept of single inheritance with an example.

Ans. Single level inheritance has only one base class and only one derived class.

Syntax:

```
class Base

{

};

class Der: public Base

{

 };
```

Example :

```
#include<iostream>

#include<conio.h>

using namespace std;

class student

{

   //private:

            int roll_no;
```

```
public:
        void getno()
        {
                cout << "\nEnter roll no: ";
           cin>>roll_no;
        }
        void putno()
        {
            cout<<"\n Roll Number : "<<roll_no;
        }
};
class sem2 :public student //first level derivation
{
        int sub1,sub2;
public:
  void getmarks()
     {
                  cout<<"enter 2 sub marks: ";
             cin>>sub1>>sub2;
     }
  void putmarks()
   {
                int total;
                cout<<"\n Marks in 1 : "<<sub1;
                cout<<"\n Marks in 2 : "<<sub2;
                total=sub1+sub2;
                cout<<"\n Total = : "<<total;
```

```
    }

};

void main()

{

    Sem2 student1;//student1 created

    student1.getno();

    student1.getmarks();

    student1.putno();

    student1.putmarks();

    getch();

}
```

Q4b.Define method overriding. Expalin the concept with an example

Ans. It is the redefinition of base class function in its derived class with same signature i.e return type and parameters. It can only be done in derived class.

Example:

```
Class a
{
public:
      virtual void display(){ cout << "hello"; }
}

Class b:public a
{
public:
       void display(){ cout << "bye";};
}
// CPP program to illustrate Function Overriding
#include<iostream>
using namespace std;

class BaseClass
{
public:
    virtual void Display()
    {
        cout << "\nThis is Display() method"
                " of BaseClass";
    }
    void Show()
    {
        cout << "\nThis is Show() method "
                "of BaseClass";
    }
};

class DerivedClass : public BaseClass
{
public:
    // Overriding method - new working of
    // base class's display method
    void Display()
    {
        cout << "\nThis is Display() method"
                " of DerivedClass";
    }
};

// Driver code
int main()
{
    DerivedClass dr;
    BaseClass &bs = dr;
    bs.Display();
    dr.Show();
}
```

Output:

```
This is Display() method of DerivedClass
This is Show() method of BaseClass
```

Q4c.What is exception handling? Explain its mechanism.

Ans. An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

throw − A program throws an exception when a problem shows up. This is done using a throw keyword.

catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

try − A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

The following is an example, which throws a division by zero exception and we catch it in catch block.

```
#include <iostream>
using namespace std;

double division(int a, int b) {
   if( b == 0 ) {
      throw "Division by zero condition!";
   }
   return (a/b);
}

int main () {
   int x = 50;
   int y = 0;
   double z = 0;

   try {
      z = division(x, y);
      cout << z << endl;
   } catch (const char* msg) {
     cerr << msg << endl;
   }

   return 0;
}
```

Q4d. What a c++ program to demonstrate the use of multilevel inheritance.

Ans.

```
#include<iostream.h>

#include<conio.h>

class student

{

    //private:

    protected:

            int roll;

    public:

            void getno()

            {

                cin>>roll;

            }

            void putno()

            {

                cout<<"\n Roll Number : "<<roll;

            }
```

```
};

class test :public student //first level derivation
{
    protected:
            int sub1,sub2;
    public:
       void getmarks()
       {
             cin>>sub1>>sub2;
       }
       void putmarks()
       {
            cout<<"\n Marks in 1 : "<<sub1;
            cout<<"\n Marks in 2 : "<<sub2;
       }
       void show()
       {
              cout<<"\n Base roll : "<<roll;
       }
};

class result:public test     //second level derivation
{
    int total; //private by default
    public:
            void display();
};

void result::display()
{
    total=sub1+sub2;

    putno();

    show();

    putmarks();

    cout<<"\n Total = : "<<total;

}


void main()

{

    result student1;//student1 created

    student1.getno();

    student1.getmarks();

    student1.display();

    getch();

}
```

Q4e.Can private members of a base class are inheritable? Justify your answer.

Ans. The private members of a class can be inherited but cannot be accessed directly by its derived classes. They can be accessed using public or protected methods of the base class

With private inheritance, public and protected member of the base class become private members of the derived class. That means the methods of the base class do not become the public interface of the derived object. However, they can be used inside the member functions of the derived class.

With private inheritance, public and protected member of the base class become private members of the derived class. That means the methods of the base class do not become the public interface of the derived object. However, they can be used inside the member functions of the derived class.

```
class Person {};

class Student:private Person {};          // private

void eat(const Person& p){}               // anyone can eat

void study(const Student& s){}                      // only students study


int main()

{

            Person p;          // p is a Person
```

```
        Student s;      // s is a Student
        eat(p);         // fine, p is a Person
        eat(s);         // error! s isn't a Person
        return 0;
}
```

Q4f.Explain the use of throw and catch keywords with proper syntax.

Ans. An exception is a problem that arises during the execution of a program. .Exception handling is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. It provides a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw
try: Program statements that you want to monitor for exceptions should be kept in a try block It's followed by one or more catch blocks
catch: The exception is caught, using catch, and processed with an exception handler
throw: A program throws an exception, if we want to throw an exception based on certain condition. Generally we use it for input validation. This is done using a throw keyword

Syntax of try-catch:

```
try {
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
} catch( ExceptionName eN ) {
   // catch block
}
```

Q5a.Explain with example how function templates are used.

Ans. Templates are the foundation of generic programming, which involves writing code independent of any particular data type.

Templates has two types:

1. Function templates
2. Class templates

The general form of a template function:

```
template <class ret-type>

ret-type function-name(parameter list)

{

   // body of function

}
```

Example:-

```
template <class T>

T sum(T a,T b)

{

return a+b;

}

void main()

{

clrscr();

int a,b,c;

cout<<"enter two integers :";

cin>>a>>b;

float fc,fd;

cout<<"enter two float value :";

cin>>fc>>fd;

cout<<"\n sum of two integers"<<sum(a,b);

cout<<"\n sum of two floats"<<sum(fc,fd);

getch();

}
```

Q5b.Exaplain the use and purpose of the following functions

i.seekg() ii.seekp() iii.tellg() iv.tellp()

Ans. (i) seekg() → It is used to move the get pointer to a desired location with respect to a reference point.

Syntax: file_pointer.seekg (number of bytes ,Reference point);

(ii). seekp() → is used to move the put pointer to a desired location with respect to a reference point.

Syntax: file_pointer.seekp(number of bytes ,Reference point);

(iii) tellg() → is used to know where the get pointer is in a file.

Syntax: file_pointer.tellg();

(iv) tellp() → is used to know where the put pointer is in a file.

Syntax: file_pointer.tellp();

Q5c.Define a class named vector. Illustrate the use of vector class template for performing the scalar product of int type vectors as well as float type vecrors.

Ans. include <iostream.h>

```
#include <conio.h>

template <class T>
class vector
{
    T *arr;
    int size;

    public:

    vector() {
       arr=NULL;
       size=0;
    }

    vector(int m);
    vector(T *a,int n);
    void modify(T value,int index);
    void multiply(int scalarvalue);
    void display();
};

template <class T>
vector<T> :: vector(int m)
{
   size=m;
   arr = new T[size];
   for(int i=0;i<size;i++)
       arr[i]=0;
}

template <class T>
vector<T> :: vector(T *a,int n)
{
   size=n;
   arr = new T[size];
   for(int i=0;i<size;i++)
       arr[i]=a[i];
}

template <class T>
void vector<T> :: modify(T value,int index)
{
    arr[index]=value;
}

template <class T>
void vector<T> :: multiply(int scalarvalue)
{
    for(int i=0;i<size;i++)
        arr[i] = arr[i] * scalarvalue;
}

template <class T>
void vector<T> :: display()
{
    cout<<"(";
    for(int i=0;i<size;i++)
    {
        cout<<arr[i];
        if(i!=size-1)
            cout<<", ";
    }
    cout<<")";
}
```

Q5d.What are two different ways to open a file in c++.

Ans. Once a stream has been created, next step is to associate a file with it. And thereafter the file is available (opened) for processing. Opening of files can be achieved in the following two ways :

1. Using the constructor function of the stream class.
2. Using the function open().

The first method is preferred when a single file is used with a stream. However, for managing multiple files with the same stream, the second method is preferred.

a. Opening File Using Constructors

We know that a constructor of class initializes an object of its class when it (the object) is being created. Same way, the constructors of stream classes (ifstream, ofstream, or fstream) are used to initialize file stream objects with the filenames passed to them

To open a file named myfile as an input file (i.e., data will be need from it and no other operation like writing or modifying would take place on the file), we shall create a file stream object of input type i.e., ifstream type. Here is an example:

```
ifstream fin("myfile", ios::in) ;
```

Similarly, when you want a program to write a file i.e., to open an output file (on which no operation can take place except writing only). This will be accomplish by

creating ofstream object to manage the output stream
associating that object with a particular file

Here is an example,

```
ofstream fout("secret" ios::out) ;
```

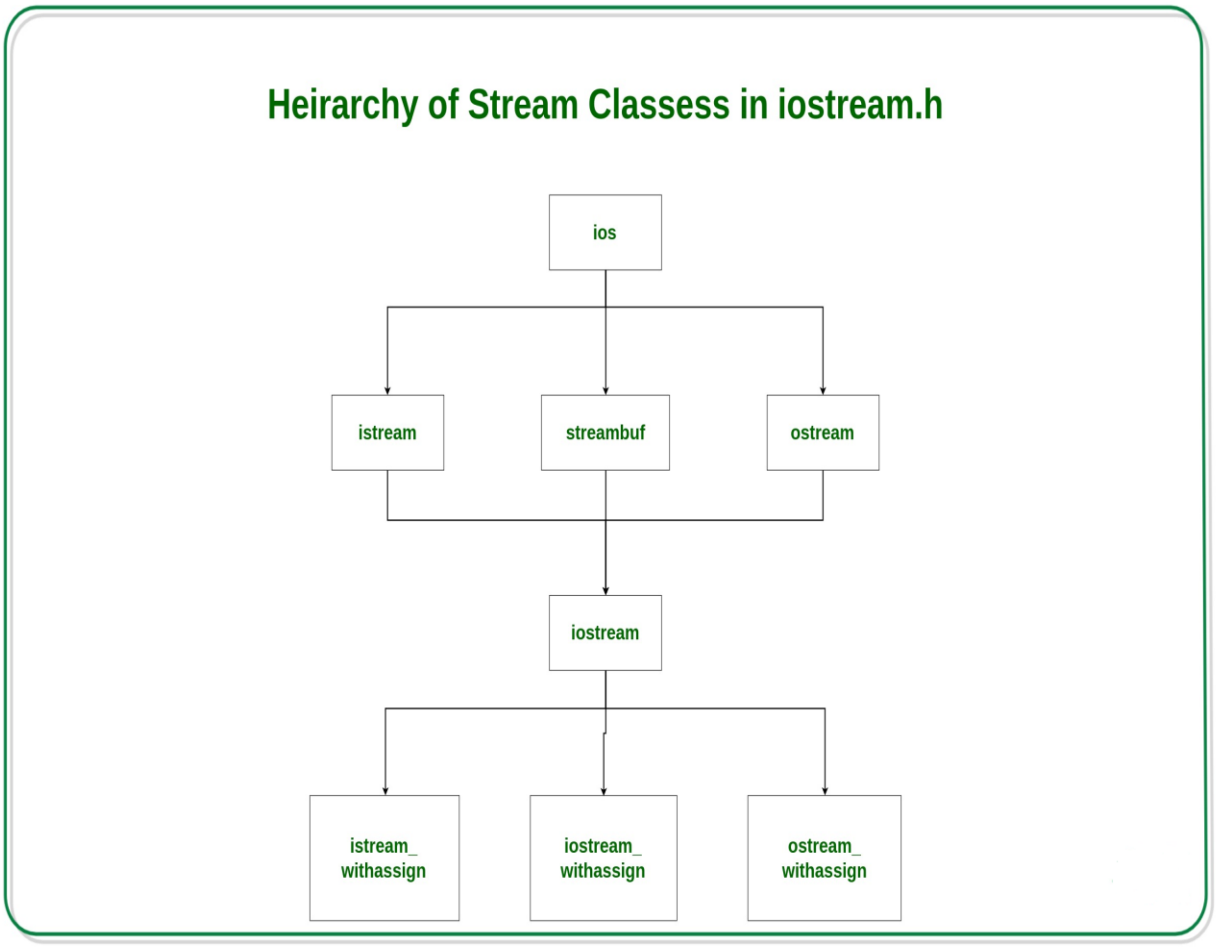
Q5e.Write a c++ program to read data from a user and write to the file.

Ans. #include <iostream>

```
#include <fstream>

using namespace std;

//class student to read and write student details
class employee
{
    private:
        char name[30];
        int age;
    public:
        void getData(void)
        { cout<<"Enter name:"; cin.getline(name,30);
          cout<<"Enter age:"; cin>>age;
        }

        void showData(void)
        {
        cout<<"Name:"<<name<<",Age:"<<age<<endl;
        }
};

int main()
{
   employee  e;

    ofstream file;

    //open file in write mode
    file.open("aaa.txt",ios::out);
    if(!file)
    {
      cout<<"Error in creating file.."<<endl;
      return 0;
    }
    cout<<"\nFile created successfully."<<endl;

    //write into file
    e.getData();    //read from user
    file.write((char*)&e,sizeof(e));    //write into file

    file.close();   //close the file
    cout<<"\nFile saved and closed succesfully."<<endl;

    //re open file in input mode and read data
    //open file1
    ifstream file1;
    //again open file in read mode
    file1.open("aaa.txt",ios::in);
    if(!file1){
        cout<<"Error in opening file..";
        return 0;
    }
    //read data from file
    file1.read((char*)&e,sizeof(e));

    //display data on monitor
    e.showData();
    //close the file
    file1.close();

    return 0;
}
```

Q5f.Explain the hierarchy of file stream class

Ans.


Heirarchy of Stream Classess in iostream.h
ios
istream
streambuf
ostream
iostream
istream_ withassign
iostream_ withassign
ostream_ withassign