Unit III

Unit IV

Unit V

Chapter - 5 Exception Handling and Templates (5 - 1) to (5 - 24)

Unit VI

Unit III

3 Polymorphism

3.1 : Introduction to Polymorphism

Q.1 What is Polymorphism ?

Ans. : • Polymorphism means **many forms.** It is one of the important features of OOP.

- Polymorphism is basically an ability to create a variable, a function, or an object that has **more than one form.**
- The **primary goal** of polymorphism is an ability of the object of different types to respond to methods and data values by using the same name.

Q.2 Differentiate between inheritance and Polymorphism.

Ans. :

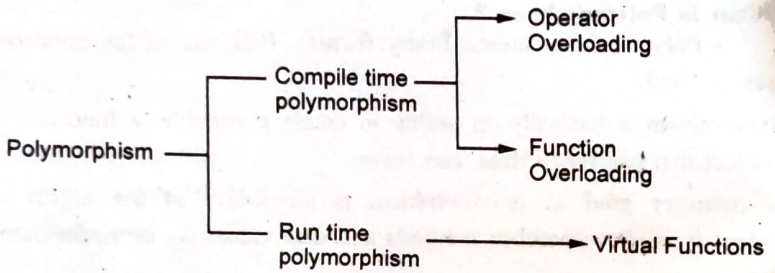
Sr. No.	Inheritance	Polymorphism
1.	Inheritance is a property in which some of the properties and methods of base class can be derived by the derived class.	Polymorphism is ability for an object to used different forms. The name of the function remains the same but it can perform different tasks.
2.	Various types of inheritance can be single inheritance, multiple inheritance, multilevel inheritance and hybrid inheritance.	Various types of polymorphism are compile time polymorphism and run time polymorphism. In compile time polymorphism there are two types of overloading possible. - functional overloading and operator overloading. In run time polymorphism there is a use of virtual function.

3.2 : Types of Polymorphism

Q.3 What are the types of Polymorphism ?

Ans. : • There are two types of poymorhism - Compile time polymorphism and runtime polymorphism.

- The compile time polymorphism is classified into Operator overloading and Function Overloading.
- The runtime polymorphism is implemented using virtual functions



3.3 : Concept of Operator Overloading

Q.4 What is operator overloading ? Name the operators that cannot be overloaded in C++ ? ☞ **[SPPU : Dec 2014, Marks 3]**

OR What is operator overloading ? ☞ **[SPPU : May 2014, Marks 2]**

Ans. : Operator overloading can be defined as an ability to define a new meaning for an existing (built-in) "operator".

How to overload operator ?

Define a function with keyword *operator*. Then write the operator(such as +, [ ] or any other valid operator) as a function name. That means we can program that specific operator.

- You can not redefine ::, sizeof, ?:, or . (dot).

Q.5 With suitable examples, demonstrate the benefits of operator overloading. ☞ **[SPPU : Dec 2015, Marks 4]**

Ans. : Due to operator overloading, an operator will not be limited to work only with primitive data Type but it can perform operation on user

defined object. For example - we can create vector class and perform addition of two vectors using the overloaded + operator.

C++ Program

```
class vector {
   public:
      int p,q;
      vector() {p=0;q=0;}// constructor without parameters
      vector (int,int);//constructor with parameters
      vector operator + (vector);//definition of operator +
};

vector::vector (int a, int b) {
   p = a;
   q = b;
}

vector vector::operator+ (vector obj)
{
   vector temp;
   temp.p = p + obj.p;
   temp.q = q + obj.q;
   return (temp);
}

int main ()
{
   vector a (10,20);
   vector b (1,2);
   vector c;
   c = a + b;
   cout<<"\n The Addition of Two vectors is...";
   cout << c.p <<" and "<<c.q;
   retrun 0;
}
```

Output

```
11 and 22
```

Q.6 Write a C++ program for overloading a increment operator ++.

Ans. :

```
#include <iostream>
using namespace std;
class coord
{
private:
    int x, y; // co-ordinate values
public:
    coord( ) { x = 0; y = 0; }//constructor for obj
    coord(int i, int j) { x = i; y = j; }//constructor with param
    void get_xy(int &i, int &j) { i = x; j = y; }
    coord operator++( );//unary operator overloading
};
// Overload ++ operator for coord class
coord coord::operator++( )
{
    x++;
    return *this; //returning the current instance
}

int main( )
{
    int x, y;
    cout<<"\n Enter the co-ordinates x and y ";
    cin>>x>>y;
    coord obj(x,y);
    ++obj;
    obj.get_xy(x, y);
    cout<<"The increment operator increments the co-ordinates
    as..."<<endl;
    cout << "X: " << x << ", Y: " << y;
    getch();
```

Calls coord operator++()


```
    return 0;
}
```

Output

```
Enter the co-ordinates x and y 10 20
The increment operator increments the co-ordinates as...
X: 11, Y: 21
```

Q.7 Write a program to overload.

1. Operator + for concatenation of two strings.

2. Operator >> for reversing a given string.

3. Operator << for displaying a given string.

☞ [SPPU : May 2014, Marks 6]

Ans. :

```
class StringClass
{
public :
    char str[30];
    StringClass()
    { strcpy(str, " "); }
    StringClass(char s[])
    { strcpy(str, s); }
    StringClass operator+(StringClass s1)
    {
        strcat(str, s1.str);
        strcat(str, "\0");
        return str;
    }
    friend ostream& operator<<(ostream &outt, StringClass &s1)
    {
        outt <<"\nString: "<< s1.str;
        return outt;
    }
    friend istream& operator>>(istream &inn, StringClass &s1)
```

```
    {

        inn >>s1.str;
        strrev(s1.str);
        cout << "\nReversed string: " << s1.str;
        return inn;
    }
};
int main()
{
    StringClass s1("Hello"), s2("friends");
    StringClass s3,s4;
    cout << s1;//invokes << operator for display of string
    cout << s2;//invokes << operator for display of string
    s3 = s1 + s2;//invokes + operator for string concatenation
    cout <<"\nConcatenated String: "<< s3.str;
    cout << "\nEnter string to reverse: ";
    cin >> s4;//invokes >> operator for string reversal
}
```

Output

```
String: Hello
String : friends
Concatenated String : Hellofriends
Enter string to reverse : abcd
Reversed string : dcba
```

Q.8 What is operator overloading ? Overload the numerical operators + and / for complex numbers addition and division respectively.

Ans. : Operator overloading is defined as an ability to define a new meaning for an existing operator.

/*Program for overloading numerical operator + and / for complex numbers addition and division*/

```
#include<iostream>
using namespace std;
class complex {
   public:
      float real,img;
      complex(){real=0;img=0;}
```



```
      complex(float,float);
      complex operator+ (complex);
      complex operator/ (complex);
};
complex::complex(float r,float i)
{
 real=r;
 img=i;
}
complex complex::operator+ (complex obj)
{
 complex temp;
 temp.real=real+obj.real;
 temp.img=img+obj.img;
  return (temp);
}
complex complex::operator/ (complex obj)
{
 complex temp;
 float new_temp;
 new_temp=(obj.real*obj.real)+(obj.img*obj.img);
 temp.real=((real*obj.real)+(img*obj.img))/new_temp;
 temp.img=new_temp;
 return(temp);
}
int main()
{
 complex a(2,6);
 complex b(4,1);
 complex c;
 c=a+b;
 cout<<"\n The addition of two complex numbers is...";
 cout<<c.real<<" and "<<c.img<<"i";
 c=a/b;
```

```
cout<<"\n The Division of two complex numbers is...";
cout<<c.real<<" and "<<c.img<<"i";
retrun 0;
}
```

Q.9 What are restrictions on use of operators ?

Ans. :

- It's **not possible** to **change** an operator's **precedence.**
- It's **not** possible **to create new operators,** For example ^ which is used in some languages for exponentiation.
- You can not redefine ::, sizeof, ?:, or . (dot).
- =, [], and –> must be member functions if they are overloaded.
- ++ and – – need special treatment because they are prefix or postfix operators.
- Assignment (=) should always be overloaded if an object dynamically allocates memory.
- It can not change the number of required operands (unary, binary, ternary).
- Overloaded operator must be either,
  - Non **static member function** of class or
  - At least **one parameter should be class** or enumeration.

3.4 : Type Casting

Q.10 What do you mean by type conversion? Explain the same with example. ☞ **[SPPU : May 2015, Marks 4]**

Ans. : Type conversion is a technique in which data of one type gets converted to another data type.

Conversion from class type to basic data type


```
class test
{
   public:
   operator int();
};
test::operator int()
{
   int sum;
   sum=5;
   return sum;
}
int main()
{
   test obj;
   int x;
   x=10;
   cout<<"\nValue of x: "<<x;
   x=int(obj); //class assigns value to a variable
   cout<<"\nValue of x assigned by the class: "<<x;
retrun 0;
}
```

Syntax of conversion function

```
operator data_type ( )
{
        .....
        .....
        .....
        return value;
}
```

Output

```
Value of x: 10
Value of x assigned by the class : 5
```

3.5 : Pitfalls of Operator Overloading and Conversion

Q.11 Enlist the pitfalls of operator overloading.

Ans. : There are following pitfalls of operator overloading and conversion -

1. The built-in operation can become **expensive.**
2. There is **increased complexity** in the program

3. The operator overloading represents **odd syntax**, hence it is difficult to trace the call to the function. For example - it is easy to trace the call for the function **addition** but it is difficult to trace for +. Due to this, the development and debugging activity gets slow down.
4. There are some operators that can not be overloaded. For example sizeof, ::, ?:
5. One **cannot change syntactical characteristics** of operator evenif it is overloaded.

3.6 : Explicit and Mutable Keywords

Q.12 Write a short note on explicit and mutable keywords.

☞ [SPPU : June-22, Marks 6]

Ans. : (1) Explicit Keyword : The explicit keyword makes a conversion constructor to non-conversion constructor. As a result, the code is less error prone.

```
class Test
{
   int Val;
public:
   explicit Test(int x) :Val(x)
   { cout << "\n Val= " << Val; };
};
int main()
{
   Test obj = 100;  //Error: conversion from int to class is not possible
   return 0;
}
```

(2) Mutable Keyword : Mutable keyword is used with the data members which we want to change even-if the object of that class is constant. For example -

```
class Test
{
   int a;
```



```
   mutable int b;
public:
void change()const
   {
      a = a + 10;//Error: 'a' cannot be modified
      //because it is being accessed through a const object
      b = b + 20;// No error  as b is mutable
   }
};
```

3.7 : Function Overloading

Q.13 What is function overloading ?

Ans. : Definition : Function overloading is a concept in which one can use **many functions** having **same function name** but can pass different number of parameters or different types of parameters.

Q.14 Write a C++ program to demonstrate the concept of function overloading.

Ans. :

```
#include <iostream>
using namespace std;
class test {
        private:
         int a,b,c,d;
        public:
         int sum(int,int);
         int sum(int,int,int);
         int sum(int,int,int,int);
       };
int test::sum(int a,int b)
{
 return (a+b);
}
int test::sum(int a,int b,int c)
{
```

```
return (a+b+c);
}
int test::sum(int a,int b,int c,int d)
{
 return (a+b+c+d);
}
int main()
{
    test obj;
    int a, b, c, d,choice;
    int result=0;
    cout<<"\n\t\t Main Menu";
    cout<<"\n\t 1.Addition of two numbers";
    cout<<"\n\t 2.Addition of three numbers";
    cout<<"\n\t 3.Addition of four numbers"<endl;
    cout<<"\n Enter Your choice : ";
    cin >> choice;
    switch(choice)
    {
     case 1: cout <<"\nEnter 2 numbers: ";
        cin >> a >> b;
        result = obj.sum(a,b);
        break;
     case 2: cout <<"\nEnter 3 numbers: ";
        cin >> a >> b >> c;
        result = obj.sum(a,b,c);
        break;
     case 3: cout <<"\nEnter 4 numbers: ";
        cin >> a >> b >> c >> d;
        result = obj.sum(a,b,c,d);
        break;
     default: cout <<"\nWrong Choice";
        break;
    }
```

Overloaded Functions


```
    cout <<"\n\nresult: " << result << endl;
    return 0;
}
```

Output

```
        Main Menu
1.Addition of two numbers
2.Addition of three numbers
3.Addition of four numbers

Enter Your choice : 2

Enter 3 numbers: 10 20 30
result: 60
```

Q.15 What is function overloading ? Explain with a suitable example.

☞ [SPPU : Dec.-19, Marks 4]

Ans. : Refer Q.13 and Q.14.

Q.16 Differentiate between function overloading and function overriding.

Ans. :

Sr. No.	Function overloading	Function overriding
1.	Function overloading occurs in same class.	The function overriding occurs in child class when the child class function overrides the parent class function.
2.	In function overloading the signature must be different for all overloaded functions.	In function overriding, the signature must be the same.
3.	Function overloading occurs during compile time polymorphism.	Function overriding occurs during run time polymorphism.
4.	In function overloading, we can have any number of overloaded functions.	In function overriding we can have only one overriding function in child class.

3.8 : Runtime Polymorphism

Q.17 Explain the concept of Runtime polymorphism in detail.

Ans. : • Runtime polymorphism is also known as **dynamic polymorphism** or **late binding**. In runtime polymorphism, the **function call is resolved at run time.**

- The runtime polymorphism can be achieved by function overriding or method overriding.
- Redefining a function in a derived class is called function overriding.

C++ Program

```
#include <iostream>
using namespace std;
class A
{
   private:
      int a,b;
   public:
      void get_msg()
      {
         a=10;
         b=20;
      }
      void print_msg()const
      {
         int c;
         c=a+b;//performing addition
         cout<<"\n C(10+20)= "<<c;
         cout<<"\n I'm print_msg() in class A";
      }
};
class B : public A
{
   private:
      int a,b;
```



```
   public:
      void set_msg()
      {
         a=100;
         b=10;
      }
      void print_msg()
      {
         int c;
          c=a-b;//performing subtratcion in this function
         cout<<"\n\n C(100-10) = "<<c;
          cout<<"\n I'm print_msg() in class B ";
      }
};
void main()
 {
   A obj_base;
   B obj_derived;
   obj_base.get_msg();
   obj_base.print_msg();//same function name
   obj_derived.set_msg();
   obj_derived.print_msg();//but different tasks
 }
```

Output

```
C(10+20)= 30
I'm print_msg() in class A

C(100-10) = 90
I'm print_msg() in class B
```

Q.18 Differentiate compile time and run time polymorphism. **MU - (Dec-19 : June-22, Marks 8)**

Ans. :

Sr. No.	Compile time polymorphism	Run time polymorphism
1.	The call to the functions having the same name is resolved at compile time.	The call to the functions having same name is resolved at run time.
2.	In this type of polymorphism the function overloading mechanism is used.	In this type of polymorphism, the function overriding mechanism is used.
3.	During compile time polymorphism, the function overloading and operator overloading techniques are used.	During run time polymorphism, the virtual functions and pointers are used.
4.	It provides fast execution.	It provides slow execution.
5.	It is less flexible as all the decisions are to be taken at compile time itself.	It is more flexible as the execution is delayed for execution time.

3.9 : Pointers to Base Class

Q.19 Is it possible to have pointer to base class? If yes, justify

Ans. : One of the key feature of inheritance mechanism is that the pointer to derived class is type compatible with pointer to its base class. That means we can create a pointer variable of a base class and assign the addresses of derived class's objects to it.

For example -

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



```
class Polygon {
 protected:
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b; }
};

class Rectangle: public Polygon {
 public:
  int area()
    { return width*height; }
};

class Triangle: public Polygon {
 public:
  int area()
    { return width*height/2; }
};

int main () {
 Rectangle rect;
 Triangle trgl;
 Polygon * ppoly1 = &rect;
 Polygon * ppoly2 = &trgl;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 cout << rect.area() << '\n';
 cout << trgl.area() << '\n';
 return 0;
}
```

3.10 : Virtual Function and its Significance in C++

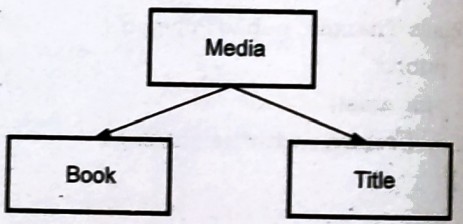
Q.20 Explain the concept of virtual function in detail.

Ans. : "A virtual function is a member function that is declared within a base class and redefined by a derived class."

The **virtual** keyword is preceded to the function name. Thus using virtual functions we can have one interface, multiple functions performing different task. This feature is called **polymorphism.**

For example - Refer program in Q.21.

Q.21 Consider an example of a book shop which sells book and video tapes. These two classes are inherited from the base class called media. The media class has command data members such as tile and publication. The book class has data members for storing number of pages in book and tape class has the playing time in a tape. Each class will have member function such as read() and show() in the base class, these members have to be defined as virtual functions, write a program which models the class hierarchy for the book shop and process the objects of these classes using pointers to the base class.



Ans. :

```
#include <iostream>
  using namespace std;
  class Media
  {
      public:
         char Title[10],publication[20];
        virtual void Read()
        {
           cout<<"\n Enter the title: ";
           cin>>Title;
           cout<<"\n Enter the publication: ";
           cin>>publication;
        }
        virtual void show()
        {
```



```
           cout<<"Title: "<<Title<<endl;
           cout<<"Publication: "<<publication<<endl;
        }
};
class Book : public Media
{
    public:
     int pages;
     virtual void Read(int num)
      {

       pages = num;
      }
     virtual void show()
     {
        cout << "Number of pages of book: ";
        cout << pages << "\n";
     }
};

class Tape : public Media
{

    public:
       int playtime;
       virtual void Read(int num)
       {

        playtime = num;
       }
       virtual void show()
       {
          cout << "Playing time in tape: ";
          cout << playtime << "\n";
       }
};
int main()
{
     Media *p;
     Media obj;
     int num;
```

```
    Book b;
    Tape t;
    obj.Read();
    obj.show();
    cout<<"\nEnter number of pages: ";
    cin>>num;
    b.Read(num);
    p=&b;
    p->show();
    cout<<"\nEnter Time in minutes: ";
    cin>>num;
    t.Read(num);
    p=&t;
    p->show();
    return 0;
}
```

Output

```
Enter the title : AAA

Enter the publication : BBB
Title : AAA
Publication : BBB

Enter number of pages : 100
Number of pages of book : 100

Enter Time in minutes : 3
Playing time in tape : 3
```

3.11 : Pure Virtual Function and Virtual Table

Q.22 What is virtual function ? Why do we need virtual function? When do we make a virtual function "pure" ? What are the implications of making a function a pure virtual function ?

☞ **[SPPU : Dec.-14, Marks 6]**

Ans. : Virtual Function and its need - Refer Q.20.

A pure virtual function is a virtual function which is to be implemented by derived class. The class that contains the pure virtual function is called the abstract class.

The pure virtual functions are declared using pure specifier i.e. = 0

As an implication of pure virtual function the base class becomes abstract, so that it can't be instantiated, but a child class can override the **pure virtual** methods to form a concrete class. This is a good way to define an interface in C++.

Example

```
#include<iostream>
using namespace std;
class B
{
   public:
      virtual void display()=0; //pure virtual function
};                  //makes class B as abstract class
class D:public B
{
public:
      void display()//overriden function
      {
         cout<<"\n Derived Class Function";
      }
};
int main()
{
   B *ptr; //Base class pointer
   D obj;  //object for derived class
   ptr=&obj;
   ptr->display();
   return 0;
}
```

3.12 : Virtual Destructor

Q.23 Write short note on - virtual destructor.

☞ [SPPU : May-16, Marks 3]

OR Explain virtual destructor with example.

☞ [SPPU : May-14, Marks 4]

Ans. : Virtual destructor is basically a base class destructor preceded by the keyword virtual. The base class is declared as virtual in order to deallocate the memory of derived class explicitly. For example

```
class Base
{
 public:
 Base()
{    cout<<"\n Calling Base class Constructor";  }
  virtual ~Base() //virtual destructor
  {    cout<<"\n Calling Base class Destructor";  }
};
class Derived:public Base
{
 public:
  Derived()
  {    cout<<"\n Calling Derived class Constructor";  }
  ~Derived()
  {    cout<<"\n Calling Derived class Destructor";  }
};

int main()
{
 Base *obj=new Derived(); //object creation
 delete obj;  //derived class memory gets deallocated.
 return 0;
}
```


3.13 : Abstract Base Class

Q.24 What is abstract base class ?

Ans. : • Abstract class is a class which is mostly used as a base class. It contains at **least one pure virtual function**. Abstract classes can be used to specify an interface that must be implemented by all subclasses.

• The virtual function is function having nobody but specified **by = 0**. This tells the compiler that nobody exists for this function relative to the base class. When a *virtual* function is made **pure**, it forces any derived class to override it. If a derived class does not, an error occurs. Thus, making a *virtual* function **pure** is a way to guarantee that a **derived class will provide its own redefinition**.

END...

Unit IV

4 Files and Streams

4.1 : Streams and Files

Q.1 What is stream ? Explain various predefined streams used in C++ with their purpose. ☞ [SPPU : June-22, Marks 4]

Ans. : Stream is basically a **channel** on which data flow from sender to receiver. Data can be sent out from the program on an output stream or received into the program on an input stream.

Stream	Meaning	Physical device
cin	Standard input	Connected to Keyboard
cout	Standard output	Connected to Screen
cerr	Standard error	Connected to Screen
clog	Buffer of error	Connected to Screen

The header file named *iostream.h* supports these I/O operations.

4.2 : Stream Classes

Q.2 Explain stream class hierarchy in detail.

Ans. : The stream class hierarchy is divided into three *areas* :

1. A buffer system given by **basic_streambuf** class. This class supports the basic low level input output operations. For advanced I/O programming the basic_streambuf class is used directly.
2. The specification system implemented by **basic_ios** class. This is high level I/O class that provides formatting, error checking. **basic_ios** is used as a base for several derived classes, including **basic_istream, basic_ostream** and **basic_iostream**. These classes are used to create streams capable of input, output and input/output, respectively.

3. A translation system implemented by certain classes like istream, ostream, iostream. (Refer Fig. Q.2.1).

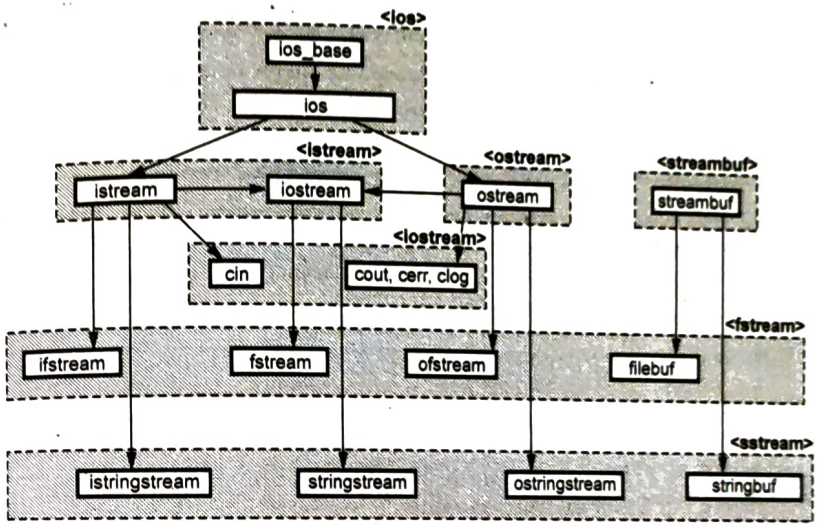


Fig. Q.2.1 Stream class hierarchy

4.3 : Stream Errors

Q.3 Explain the commonly used error functions defined by ios class.

Ans. : The **ios** class defines some member functions using which the state of the stream can be checked.

Member Function	Purpose
bool ios::good()	This functions returns true if there is no error
bool ios::bad()	This function returns true if no read/write operation is performed or invalid read/write operation is performed.
bool ios::eof()	This function returns true if the input operation is reached end of file

bool ios::fail()	This functions returns true if the input operation failed to read the expected characters, or that an output operation failed to generate the desired characters.
void ios::clear()	This function clears all the flag if no argument is supplied. It can also be used to clear a particular state flag if supplied as argument.

Q.4 Explain how does stream error checks the state of the stream and reports error messages on wrong input ?

Ans. : The **ios** class defines some member functions using which the state of the stream can be checked. For example -

```
void main()
{
    int val;
    char ans = 'y';
    cout << "\n Enter an integer value: ";
    cin >> val;
    if (cin.good() == 0)//if  error
    {
        cout << "\n Incorrect input";
        cout << "\nThe good() function returns: " << cin.good();
        cin.clear();
    }
    else
    {
        cout << "\n Correct input";
        cout << "\nThe good() function returns: " << cin.good();
    }
}
```

The above program will give error message if non integer value is entered by the user.

Q.5 What is stream ? Write a program to illustrate the stream error concept. ☞ [SPPU : Dec.-19, Marks 6, June-22, Marks 8]

Ans: Refer Q.1 and Q.4

4.4 : Disk File I/O with Streams

Q.6 Explain the ofstream, ifstream, and fstream classes.

Ans. : C++ provides following classes to perform input and output of characters to and from the files.

ofstream	This stream class is used to write on files.
ifstream	This stream class is used to read from files.
fstream	This stream class is used for both read and write from/to files.

To perform file I/O, we need to include **<fstream.h>** in the program. It defines several classes, including **ifstream, ofstream** and **fstream**. These classes are derived from **ios**, so **ifstream, ofstream** and **fstream** have access to all operations defined by **ios**. While using file I/O we need to do following tasks -

1. To create an input stream, declare an object of type **ifstream**.
2. To create an output stream, declare an object of type **ofstream**.
3. To create an input/output stream, declare an object of type **fstream**.

Q.7 Explain various flags used in modes of file open operation. ☞ [SPPU : June-22, Marks 4]

Ans. : The mode is optional parameter and is used with the flags as given below -

Ios::in	Open for input operation.
Ios::out	Open for output operation.
Ios::binary	Open for binary operations.

Ios::ate	If this flag is set then initial position is set at the end of the file otherwise initial position is at the beginning of the file.
Ios::app	The output operations are appended to the file. This is an appending mode. That means contents are inserted at the end of the file.
Ios::trunc	The contents of pre existing file get destroyed and it is replaced by new one.

Q.8 Write a C++ program to read the contents of a text file.
Ans. :

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
using namespace std;
int main()
{
      ifstream in;
       char Data[80];
       in.open("Sample.dat");
      if (!in)
       {
       // Print an error and exit
       cerr << "Sample.dat could not be opened for reading!" << endl;
       exit(1);
       }
       cout<<"The Contents of the file are..."<<endl;
       while(in)
       {
          in.getline(Data,80);
          cout<<"\n"<<Data;
       }
       in.close();
      return 0;
}
```


Note : The Sample.dat file is already created with the data as obtained in above output

Q.9 Write a C++ program that reads a file and counts the number of sentences, words and characters present in it.
Ans. :

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
int main()
{
    char Data[80];
    int wc=0;
    int cc=0;
    int lc=0;
    ifstream in_obj;
    in_obj.open("Odd.dat");
    in_obj.read((char *)&Data,sizeof(Data));
    while(in_obj)
    {
       in_obj.getline(Data,80);
       int n=strlen(Data);
       cc+=n;
       lc++;
       for(int i=0;i<n;i++)
       {
           if(Data[i]==' ')
            wc++;
       }
    }
    in_obj.close();
    cout<<"\n Number of Sentences: "<<lc;
    cout<<"\n Number of Characters: "<<cc;
    cout<<"\n Number of Words: "<<wc;
```

```
    cout<<endl;
    retrun 0;
}
```

Q.10 Write a C++ program to write a class object to the file. Read the contents of the file and display them on the console.
Ans. :

```
#include <iostream>
#include <fstream>
#include<string.h>
using namespace std;
class Employee
{
public:
    char  Name[40];
    int ID;
    double Salary;
};
void main()
{
    Employee emp1;
    cout<<"Writing Data to the stream..."<<endl;
    strcpy(emp1.Name, "Parth");
    emp1.ID =10;
    emp1.Salary = 17000;
    ofstream fp("Sample.dat", ios::binary);
    fp.write((char *)&emp1,sizeof(emp1));
    fp.close();
    cout<<"Reading Data from the stream..."<<endl;
    // writing object to file
    Employee emp2;
    ifstream fpr("Sample.dat", ios::binary);
    fpr.read((char *)&emp2,sizeof(emp2));
    cout<<"Name: "<<emp2.Name<<endl;
    cout<<"ID: "<<emp2.ID<<endl;
```



```
    cout<<"Salary: "<<emp2.Salary<<endl;
    fpr.close();

}
```

4.5 : File Pointers

Q.11 What do you understand by the term - File pointer ?
Ans. : The file pointer is used for locating the position in the file. With each file object there are two pointers associated with it. The get pointer and put pointer. These pointers basically return the current get position and current put position.

Q.12 Explain seekp, seekg and tell functions with suitable examples.
Ans. :

- **seekg** means get pointer of specific location for reading of record.
- **seekp** means get pointer of specific location for writing of record.

The syntax of **seek** is

seekg (offset, reference - position);

seekp (offset, reference - position);

where, **offset** is any constant specifying the location.

reference - position is for specifying beginning, end or current position. It can be specified as,

ios : : beg	for beginning location
ios : : end	for end of file
ios : : cur	for current location

- **tell**

This function tells us the current position.

For example

seqfile. tellg () - gives current position of get pointer (for reading the record).

seqfile. tellp () - gives current position of put pointer (for writing the record).

Q.13 Explain the concept of file pointers. ☞ [SPPU : Dec.-19, Marks 6]

Ans. : Refer Q.11 and Q.12.

4.6 : Error Handling in File I/O

Q.14 Write a C++ program to perform error handling with file I/O operations.

Ans. :

```
#include <iostream>
#include <fstream>
using namespace std;
#include <process.h>
const int MAX = 10;
int array1[MAX] = { 10,20,30,40,50 };
int array2[MAX];
int main()
{
   ofstream os; //create output stream
   os.open("d:\\test.dat", ios::trunc | ios::binary);//Opening file
   if (!os)
   {
      cerr << "Could not open output file\n";   //Error Handling
      exit(1);
   }
   cout << "Writing the contents to the file...\n\n";
   os.write((char*)&array1,sizeof(array1));//writing 'array1' to file
   if (!os)
   {
      cerr << "Could not write to file\n";  //Error handling
      exit(1);
   }
   os.close();    //close the file
   ifstream is; //create input stream for reading the contents from
                //file
   is.open("d:\\test.dat", ios::binary);
   if (!is)
   {
      cerr << "Could not open input file\n"; //Error Handling
      exit(1);
   }
```



```
   cout << "Reading the contents from the file ...\n";
   is.read((char*)&array2,sizeof(array2));  //reading the contents in
                                            //'array2'
                                            //another array
   if (!is)
   {
      cerr << "Could not read from file\n"; //Error Handling
      exit(1);
   }
   for (int j = 0; j < MAX; j++) //check data
      cout << " " << array2[j];
   return 0;
}
```

Q.15 Explain error handling in file I/O with suitable program. ☞ [SPPU : Dec.-19, Marks 6]

Ans. : Refer Q.14.

4.7 : File I/O with Member Functions

Q.16 Explain how a member function of a class can take part in file read and write operation with the help of necessary C++ code.

Ans. :

```
{
protected:
   int rollNo;
   char name[40];
   float marks;

public:
   void input_Data(void)
   {
      cout << "\n Enter Roll Number: ";
      cin >> rollNo;
      cout << "\n Enter name: ";
      cin >> name;
      cout << "\n Enter marks: ";
      cin >> marks;
   }
   void display(void)
   {
```

```
        cout << "\n Roll No : " << rollNo;
        cout << "\n Name : " << name;
        cout << "\n Marks :" << marks;
        cout << "\n----------------------------";
    }
    void ReadFile(int);
    void WriteFile();
    static int TotalRec();
};
void Student::ReadFile(int index)        // Member function for read file
{
    ifstream in_obj;
    in_obj.open("test.dat", ios::binary);
    in_obj.seekg(index*sizeof(Student));
    in_obj.read((char*)this, sizeof(*this)); //read one record at a time
}
void Student::WriteFile()       // Member function for write file
{
    ofstream out_obj;

   out_obj.open("test.dat", ios::app | ios::binary);
   out_obj.write((char*)this, sizeof(*this)); //writes current record to file
}
int Student::TotalRec()
{
    ifstream in_obj;
    in_obj.open("test.dat", ios::binary);
    in_obj.seekg(0, ios::end);
    //by tellg getting count for total number of records
    int size= (int)in_obj.tellg() / sizeof(Student);
    return size;
}
```

4.8 : Overloading Extraction and Insertion Operator

Q.17 How do you declare an overloaded stream insertion and extraction operator ? ☞ **[SPPU : Dec.-14, Marks 3]**

Ans. : In C++ the output operation is denoted using << which is called as **insertion operator**. Similarly the input operation is denoted using >>

operator which is called as **extraction operator**. The C++ code for overloaded insertion and extraction operators are as follows -

```
//inserter function
ostream &operator<<(ostream &st,Point obj)
{
 st <<obj.x << ", " <obj.y <<"\n"; //outputting the values
return st;  //returning stream
}
//extractor function
istream &operator>>(istream &st, Point &obj)
{
cout << "Enter x: ";
st >> obj.x;     //inputting the values
cout << "Enter y: ";
st >> obj.y;
 return st;
 }
```

4.9 : Memory as a Stream Object

Q.18 Write a short note on - memory as a stream object.

Ans. : • In C++, it is possible to treat particular section of memory as an object and data can be inserted into this section of memory with the help of memory object.

- The stream classes **ostrstream** is commonly used to create memory object
- The syntax for creating such memory object is

  ostrstream *ObjectName(databuffer,size_of_buffer)*

```
int main()
{
    int RollNo = 10;
    char name[] = "Rashmi";
    char Data[SIZE]; //Data Buffer for memory object
```

```
    ostrstream memObj(Data, SIZE); //create stream object by passing
    buffer and size
    memObj << "Roll Number = " << RollNo
           <<setw(10)
           << "Name = " << name
           << '\0'; //end the buffer with '\0'
    cout << Data; //display the data buffer
    return 0;
}
```

4.10 : Command-line Arguments

Q.19 What is command line arguments in C++? Write a program for the same. ☞ [SPPU : Dec.-19, Marks 6]

Ans. : The command line arguments are the arguments that are passed to the main function. These are represented as follows -

```
int main(int argc, char *argv[])
```

Here the **arc** represents the total number of arguments and **argv** is a an array of characters that store the command line arguments.

The **argv[0]** is the name of the program, or an empty string if the name is not available. After that, every element number less than **argc** is a command line argument. You can use each **argv** element just like a string.

Q.20 Write a C++ program to read the contents of the file in which the file name is specified by command line argument.

Ans. :

```
#include <fstream>
#include <iostream>
using namespace std;
#include <process.h>
int main(int argc, char* argv[] )
{
   if( argc != 2 )
   {
```



```
      cerr << "\nFile Name is Missing!!!";
      exit(-1);
   }
   char ch;
   ifstream in_file;
   in_file.open( argv[1] );
   cout<<"\n The contents of "<<argv[1]<<" are...\n";
   while( in_file.get(ch) != 0 )
      cout << ch;
   return 0;
}
```

4.11 : Printer Output

Q.21 Explain how C++ code is used to send the output to the printer ?

Ans. : In C++ it is possible to send data to the printer and print and get the formatted output.

For example

```
#include <fstream> //for file streams
using namespace std;
int main()
{
   ofstream printer; //make a file
   printer.open("PRN"); //open it for printer
   printer << "Hello friends!!!" << endl; //send data to printer
   printer << '\x0C'; //formfeed to eject page
   printer.close();
   return 0;
}
```

END...

Unit V

5 Exception Handling and Templates

5.1 : Error Handling Techniques

Q.1 What are three commonly used error handling techniques ?

Ans. : Three commonly used error handling techniques -

1. Terminate the program on error situation
2. Write the error code and assign it to global variable. On getting the error, return the corresponding code.
3. Some status code can be predefined and on success or failure the corresponding code can be returned.
4. Use of **goto** for jumping to desired statement.

Typically the functions like **abort()** or **exit()** are used for getting out of the error-prone situations.

5.2 : Simple Exception Handling

Q.2 What is an exception ? How is an exception handled in C++ ? What are the advantages of using exception handling mechanism in program ? ☞ **[SPPU : Dec.-14, Marks 6, June-22, Marks 4]**

Ans. : Definition : When any unavoidable circumstances (or runtime errors) occur in our program then exceptions are raised by handling control to special functions called **handlers.** This provides build-in error handling mechanism which is known as **exception handling.**

C++ exception handling mechanism makes use of three keywords - **try, catch** and **throw**. The **try** represents the block of statements in which there are chances of occurring some exceptional conditions.

When exception detected it is thrown using the **throw** statement.

There exists a block of statements in which the exception thrown is handled appropriately. This block is called **catch block.**

For example

```
#include <iostream>
using namespace std;
int main ()
  {
   try
   {

        throw 100;
   }
   catch (int x)
   {
    cout << "An exception at " << x << endl;
   }
   return 0;
   }
```

Output
An expection at 100

In above program the *try* block contains the portion of the code for exception handling.

The main advantage of exception handling mechanism is that any error causing situation can be gracefully handled by raising appropriate exception without carshing the code.

Q.3 Explain exception handling mechanism in C++.
☞ **[SPPU : May-15, Marks 4]**

Ans. : Refer Q.2.

Q.4 What is exception ? How is an exception handled in C++ ?
☞ **[SPPU : Dec.-19, Marks 5]**

Ans. : Refer Q.2.

Q.5 What is the difference between error and exception ?
Ans. :

Sr. No.	Error	Exception
1.	Errors are some abnormal or wrong situations that occur in the program.	Exceptions are some abnormal or wrong situations that occur in the program.
2.	Error cannot be handled.	Exception can be handled using exception handler.
3.	Error is uncoverable.	Exception is coverable.
4.	Program crashes or stops working when an error occurs.	Program reports user friendly message about the abnormal situation and exits gracefully.
5.	Error cannot be covered but it is fixed by the programmer.	The exception can be handled using try, catch and throw statements.
6.	Error is compile time error.	Exception is run time error.

5.3 : Divide by Zero

Q.6 Write a program to handle divide by zero situation while performing division operation using exception handling.
Ans. :

```
int main()
{
   double i, j;
   void divide(double, double);
   cout << "Enter numerator : ";
   cin >> i;
   cout << "Enter denominator : ";
   cin >> j;
```



```
   divide(i, j);
}
void divide(double a, double b)
{
   try
   {
      if (b == 0)
         throw b; // divide-by-zero
      cout << "Result: " << a / b << endl; // for non zero value
   }
   catch (double b)
   {
      cout << "Can't divide by zero.\n";
   }
}
```

Output

```
Enter numerator : 10
Enter denominator : 0
Can't divide by zero.
```

5.4 : Multiple Catching

Q.7 How to handle multiple exceptions occurred in a program ?
☞ [SPPU : May-14, Marks 5]

Ans. : When an exception occurs then the control is transferred to the catch block and at that time *try* block is terminated. There can be multiple exceptions in **multiple catch** statements with one *try* block. Following is a structure of the program when multiple catch blocks are allowed.

```
void function
{
   ...
   try
```

```
{
  ...
}
catch(datatype1 arg)
{
....
}
catch(datatype2 arg)
{
  ...
}
.....
.....
.....
catch(datatypeN arg)
{
  ...
}
}
```

The program illustrating multiple catch statements is as shown below -

```
#include<iostream>
using namespace std;
void function(int num)
{
   try
   {
   if(num)
      throw num;
    else throw "Value is zero";
   }
   //multiple catch for single try block
   catch(int i)
   {
```



```
    cout << "Exception for number is handled: " << i << "\n";
   }
   catch(const char *str)
   {
   cout << "Exception for string is handled: ";
   cout < str << "\n";
   }
   }
   int main()
   {
   cout << "Start"<<endl;
   function(0); // As value of num is 'not true' else part will
   function(1); // be executed
   function(2);
   function(3);
   cout << "End"<<endl;
   retrun 0;
}
```

Output

```
Start
Exception for string is handled: Value is zero
Exception for number is handled: 1
Exception for number is handled: 2
Exception for number is handled: 3
End
```

5.5 : Re-throwing an Exception

Q.8 When do we rethrow an exception ? Explain the way of rethrowing the exception.

Ans. : When the exception is thrown inside the try block it is propagated to the catch block that immediately follows it. But sometimes handler may decide to rethrow it to propagate to next catch statement. In such a situation the exception can be rethrown by simply writing throw without any argument.

Following program shows the rethrowing of the exception.

```
#include<iostream>
using namespace std;
void function()
{
    try
    {
        throw "myworld";
    }
    catch(char*)
    {
        cout<<"\nInside the catch statement of function()";
        throw; //rethrowing the exception
    }

}

int main()
{
    try
    {
        function();
    }
    catch(char*)
    {
        cout<<"\n\nInside the catch statement of main()";
    }
    return 0;
}
```

The exception which is rethrown is handled here by this catch block.

5.6 : Exception Specifications

Q.9 What is exception specification ? Explain it with suitable example.

Ans. : Exception specification is used to provide the information about the kind of exceptions that can be thrown. For example –

```
void function() throw (int,double)
{
        throw(12.34);

}
void main()
{
    try
    {   function();  }
    catch(int i)
    { cout<"Handling the integer value "<i<endl;  }
    catch(double i)
    { cout<"Handling the double value "<i<endl;  }
    catch(...)
    {  cout<"Exception handling for something else"<endl; }
}
```

In above program,the statement **void f() throw(int,double)** the function f() throws an exception of types **integer** or **double**. If it throws an exception with a different type(here other than **int** or **double** type), either directly or indirectly, it cannot be caught by a regular **int** or **double** type handler.

5.7 : User Defined Exceptions

Q.10 What is user defined exception ? How to implement it ?

☞ **[SPPU : June-22, Marks 7]**

Ans. : • User defined exception is a kind of exception defined by the user.

• As most of the exceptions that we need to handle in C++ are of class type.

- The object of class type is passed to exception handler (i.e. to catch routine) and with the help of object an error is processed.

Following is a simple program that illustrates the use of user defined exception.

```
#include<iostream>
#include<cstring>
using namespace std;
// Catching class type exceptions.
class My_Exception
{
 public:
      char str[50];
      int num;
      My_Exception() { *str = 0; num = 0; } // constructor
      My_Exception(char *s, int i)
      {
    strcpy(str, s);
    num = i;
       }
};
void main()
{
  int a;
  //testing if the number is positive or not
  try
  {
      cout << "Enter a positive number: ";
      cin >> a;
      if(a<0)
      throw My_Exception("It's a negative number", a);
      else
      cout <<"It's a positive number"<< endl;
  }
```



```
  catch (My_Exception obj)
  { // catch an error
    cout << obj.str << ": ";
    cout << obj.num << endl;
  }
}
```

Output 1

```
Enter a positive number: 10
It's a positive number
```

Output 2

```
Enter a positive number: -5
It's a negative number: -5
```

Q.11 Write a C++ program to compute the square root of a number. The input value must be tested for validity. If it is negative, the user defined function mysqrt() should raise exception.

Ans. :

```
/*************************************************************************
   Program to handle square root function using exception handling
*************************************************************************
************/
#include<iostream>
#include<math.h>
void MySqrt(double val)
{
   try
   {
   if (val < 0.0)
            throw "Negative";
      else
          cout << "The sqrt of "<<val<<" is "<<sqrt(val)<<endl;
   }
   catch(char *str)
   {
```

```
        cout<<"Can not handle "<<str<<" number"<<endl;
    }
}
int main()
{
        cout<<"\n Enter some number: ";
        double num;
        cin>>num;
        MySqrt(num);
        return 0;
    }
```

Output

```
Enter some number: -5
Can not handle Negative number
```

5.8 : Processing Unexpected Exceptions

Q.12 Justify the statement – "An unexpected exception is processed in C++"

Ans. : For processing the unexcepted exception, the **set_unexpected** function is used . This function is also called as **unexpected handler function**. The unexpected handler by default calls the **terminate** function.

```
void myunexpected ()
 {
  cerr<"unexpected called\n";
  throw 0;
}
void function() throw (int,double)
{
    throw("Hello");
}
    void main()
    {
        set_unexpected (myunexpected);//setting unexpected
                                      //exception handler
```

Exception specification lists int and double values.

The string is not specified in the list hence unexpected exception occurs.


```
    //try block calling function()
    //catch block
        }
```

5.9 : Constructor, Destructor and Exception Handling

Q.13 How does exception handing activity is carried out when constructor and destructors are present in C++ program ?

Ans. : When the exception is raised inside a try block, for all the objects created inside the blocks the destructor is called first and then the catch block executes to handle the exception raised in try block.

For example

```
class Test
{
public:
    Test() { cout < "\n Constructor is called"; }
    ~Test() { cout < "\n Destructor is called"; }
};
int main()
{
    try {
        Test obj1,obj2; //calling constructor for twice
        throw 100; //calling destructors twice
                   //and then catch block executes
    }
    catch (int e)
    {   cout < "\nException is caught!!!" < e;  }
 return 0;
}
```

5.10 : Exception and Inheritance

Q.14 Explain the mechanism of exception handling during the inheritance.

Ans. : It is possible to throw the object of derived class as exception, in that case the exception handler will check - who is the base class from which the derived class is derived. For example

```
class B
{
};
class D :public B
{
};
void main()
{
    D d;
    try
    { throw d; }
    catch (B obj)
    { cout < "\n Base class Exception handling catch block"; }
                                    //exception gets handled
                                    // by Base class
    catch (D obj)
    { cout < "\n Derived class Exception handling catch block"; }
}
```

5.11 : Introduction to Templates

Q.15 What is generic programming ?

Ans. : The **generic programming** is a technique that allows to write the code for **any data type** elements. The template is used as a tool for generic programming.

Templates allow the reusability of the code. There are two categories of templates -

1. Function template 2. Class template

5.12 : The Power of Templates

Q.16 Explain the usefulness of templates.

Ans. :

- Templates are useful for **code reusability.**
- The generic functions and generic classes are useful tool for creating the generalized code that can handle any datatype elements.
- The Standard Template Library(STL) is built upon the concept of templates, in which the commonly used library classes or functions are written as templates.
- It helps in generating high performance object code.

5.13 : Function Template

Q.17 What is function template ?

Ans. : To perform identical operations for each type of data compactly and conveniently, the function templates are used. One can write a single function template definition. Based on the argument types provided in calls to the function, the compiler automatically instantiates separate object code functions to handle each type of call appropriately.

Function templates are implemented like regular functions, except they are prefixed with the keyword **template.**

```
template <class T>
   T min(T a, T b)
   {
      if (a<b)
      return a;
      else
      return b;
   }
   int main()
   {

      cout << "min(10, 20) = " << min(10, 20) << endl ;
      cout << "min('p', 't') = " << min('p', 't') << endl ;
```

```
    cout << "min(10.3, 67.2) = " <<min(10.3,67.2)<<endl;
    return 0;
}
```

5.14 : Overloading Function Templates

Q.18 What is overloading template ? Explain it with example. ☞ [SPPU : May-15, Marks 4]

Ans. : • It is possible to overload the function template by some non-template function or by template function. This non template function has the same name as the template function but it is not related to the template function.

• Following is an example of overloading the function template

```
template <class T>
void display(T x)
{
    cout << x;
}
void display(int a)
{
    cout << "\n Inside the non template function";
}
int main()
{
    cout << "\n Displaying string: ";
    display("Hello");
    cout << "\n Displaying floating number: ";
    display(10.55);
    display(10); //calls overloaded non template function
    return 0;
}
```


Q.19 Distinguish between overloaded functions and function templates. Write a function template for finding the minimum value contained in an array. ☞ [SPPU : Dec.-14, Marks 6]

Ans. : Function overloading allows the definition of more than one function with the same name and provides a means of choosing between the functions based on parameter matching. Template functions tell the compiler how to create new functions, based on the instantiation datatypes. Functions generated from the template (instantiations) behave like normal functions.

The function body will differ in function overloading whereas the function body will not differ in function template.

```
template <class T>
  T min(T a[10])
  {
     T min;
     min=a[0];
     for(int i=0;i<5;i++)
     {
        if(a[i]<min)
        {
           min=a[i];
        }
     }
     return min;
     }
```

Q.20 Write a function template for finding the minimum value contained in an array. ☞ [SPPU : Dec.-19, Marks 3, June-22, Marks 6]

Ans. : Refer Q.19.

5.15 : Class Template

Q.21 What is class template ?

Ans. : Using class template we can write a class whose members use template parameters as types.

For Example :

```
template <class T>
class Compare {  //writing the class as usual
   T a, b;//note we have used data type as T
 public:
   Compare (T first, T second)
   {
    a=first;
    b=second;
   }
   T max ();//finds the maximum element among two
};
//template class member function definition
//here the member function of template class is max
template <class T>
T Compare <T>    ::max ()
{
 T val;
 if(a>b)
   val=a;
 else
   val=b;
 return val;
}

int main ()
{
 Compare <int> obj1 (100, 60);//comparing two integers
 Compare <char> obj2('p','t');//comparing two characters
 cout <<"\n maximum(100,60) = "<obj1.max();
 cout<<"\n maximum('p','t') = "<obj2.max();
 retrun 0;
}
```


5.16 : Template Arguments

Q.22 Explain template arguments with the help of illustrative example.

Ans. : Templates can have multiple arguments. These arguments are normally denoted by the data type **T.** But along with **T** we can also define other data type arguments. For example we can define the template function as

Compare (T first, int second); //First argument is of type T and second is int

Following program illustrates this concept -

```
#include <iostream>
using namespace std;
template <class T>
class Compare
{  //writing the class as usual
    T a, b;//note we have used data type as T
 public:
Compare (T first, int second);
       T max ();//finds the maximum element among two
};
//template class member function definition
//here the member function is constructor
template <class T>
Compare<T>::Compare(T first, int second)
{
    a=first;
    b=second;
 }
    //here the member function of template class is max
 template <class T>
 T Compare <T>::max ()
```

```
{
  T val;
  if(a>b)
      val=a;
  else
      val=b;
  return val;
}
int main ()
{
   Compare <int> obj1 (100, 160);//comparing two integers
   cout <<"\n maximum(100,160) = "<<obj1.max();
   return 0;
}
```

5.17 : Class Template and Nontype Parameters

Q.23 Explain nontype parameter.

Ans. : Definition : A non-type template argument provided within a template argument list is an expression whose value can be

- integral,
- pointer,
- or pointer to member type,
- enumeration,
- reference,
- and must be constant at compile time.

- Floating point values are not allowed as template parameters.
- The Non-type template parameters provide the **ability to pass a constant expression at compile time.** For example -

```
#include<iostream>
using namespace std;
template<int i> class C //integral expression
{
public:
   int k;
```



```
   C()
   {
      k = i;
   }
   void display() { cout << "\n  " << k; }

};
int main()
{
   C<100> x;//initialization of class with non-type template argument
   C<200> y;
   x.display();
   y.display();
   return 0;
}
```

Q.24 What is C++ template ? Describe type and non type tempate. ☞ [SPPU : Dec.-15, Marks 8]

Ans. : Refer Q.21 and Q.23.

5.18 : Template and Friends

Q.25 How to use friend function along with templates ?

Ans. : A friend function can be used within a template class.

Following C++ program illustrates how to use friend function inside the template class.

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

template <class T>
class Test
{
   template <class T> friend void f(Test<T>&); //declaration of
                                                //friend function
                                               //in template class
```

```
private:
    T value;
public:
    T get() { return value; }
    void set(T v) { value = v; }

};

template<class T>
void f(Test<T>& obj)
{
    cout << "Inside Friend Function!!!";
    cout << "\n\t You have entered..."<<obj.get();
}

int main()
{
    Test<int> object;
    int num;
    cout << "\n Enter the value to set ";
    cin >> num;
    object.set(num);
    f(object);//calling friend function
    return 0;
}
```

5.19 : Generic Functions

Q.26 Write a generic function for searching an element from the list of integers and double.

Ans. :

```
template <class T>
void display(T *list,T key, int count)
{
```



```
    int flag=0;
    for (int i = 0; i < count; i++)
    {
        if (key == list[i])
            flag = 1;

    }
    if (flag == 1)
        cout << "\n The element is present in the list";
    else
        cout << "\n The element is not present in the list";

}
int main()
{
    int a[5] = { 33,12,78,100,55 };
    cout << "\n Searching element 100 from the list";
    display(a, 100, 5);
    double b[7] = { 33.33,12.12,78.78,100.10,55.55,66.66,91.67};
    cout << "\n Searching element 78.78 from the list";
    display(b,78.78,7);
    return 0;
}
```

Q.27 Write short note on - Generic classes.

Ans. : • When the class logic can be generalized then it becomes a generic class. For example the class that creates a liked list for integer numbers, same class can create a linked list for double or characters.

- The generic classes are created using the template class.

Example

```
template <class T>
class SumClass {
    T a, b;
public:
    SumClass(T first, T second)
    {
        a = first; b = second;
    }
    T addition();
};

template <class T>
T SumClass<T>::addition()
{
    T retval;
    retval = a + b;
    return retval;
}
```

Q.28 What is template ? Write a program to handle addition of two numbers using template. ☞ [SPPU : Dec.-19, Marks 6]

Ans. : Refer Q.21 and Q.27.

5.20 : The Typename and Export Keywords

Q.29 Explain the typename and export keywords.

Ans. :

I) typename : The keyword **typename** can be used in place of **class** in template parameter list. It can be used in a template declaration and definition.

`template<class type1, class type2>`

can be replaced by

`template<typename type1,typename type2>`

II) export : The **export** keyword can be used to export the template definitions to other files. The keyword **export** is preceded the **template** keyword in the template definition.

For example -

```
export template <typename T>
void display(T *list, T key, int count);
```

can be placed in some header file say **test.h.** This test.h can then be included in the main application program which uses the templates.

END...

Unit VI

6 Standard Template Library

6.1 : Introduction to STL

Q.1 What is STL ?

Ans. : The Standard Template Library (STL) is collection of well structured generic C++ classes (templates) and functions. Basically STL consists three basic components -

1. Container 2. Algorithms 3. Iterators

6.2 : STL Components

Q.2 Explain the terms – Containers, algorithms and iterators.

Ans. : Container : The container is a collection of objects of different types. These objects store the data.

Iterator : The iterators are basically objects but sometimes they can be pointers and hence iterators

Algorithm : The algorithms are used to process the contents of the containers. The functionalities provided in container are not sufficient to perform complex operations hence the algorithms are used to support more complex operations for the containers specify the positions in container. The iterators are used to traverse the contents of container.

Q.3 What is STL ? List different types of STL containers.

☞ [SPPU : Dec.-19, Marks 6]

Ans. : Refer Q.1 and Q.2.

6.3 : Sequence Container

Q.4 What is sequence container ?

Ans. : • It consists of all the classes who represent the sequence or linear list.

• For example, **vector** class defines dynamic array. **Deque** is for creating doubly ended queue i.e. we can perform insertion and deletion of elements from both the ends of queue and list provides a linear list. The **list** class defines the collection of elements in which only sequential access to the elements is allowed.

Q.5 Write and explain some commonly used functionalities used in vector. ☞ [SPPU : June-22, Marks 4]

Ans. : Some commonly used vector functions are -

Function	Description
begin()	Returns the first element of the vector.
end()	Returns the last element of the vector.
size()	Returns the size of the vector.
erase()	Erases the given elements.
push_back()	Insert the element in the vector at the end.
pop_back()	Deletes the last element of the vector.
resize()	Modifies the original size of the vector.

Q.6 Write a C++ code for performing various operations on deque.

Ans. : The doubly ended queue(Deque) is a type of data structure in which the element can be inserted from both the front as well as rear end. Similarly the element can be deleted from the front as well as rear end.

```
#include <deque>
using namespace std;
int main()
```

```
{
    int item, choice;
    char ans = 'y';
    deque<int> dq;
    deque<int>::iterator i;
    do
    {
        cout << "\n Program for Implementing DEQUE using Sequence
Container";
        cout << "\n Main Menu";
        cout << "\n1. Insert from Rear";
        cout << "\n2. Insert from Front";
        cout << "\n3. Delete from Front";
        cout << "\n4. Delete from Rear";
        cout << "\n5. Get the Size of DEQUE";
        cout << "\n6. Display";
        cout << "\n Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:cout << "\n Enter the element to be inserted: ";
            cin >> item;
            dq.push_back(item);
            break;
        case 2:cout << "\n Enter the element to be inserted: ";
            cin >> item;
            dq.push_front(item);
            break;
        case 3:item = dq.front();
            dq.pop_front();
            cout << "\n The deleted element is: " << item;
            break;
        case 4:item = dq.back();
            dq.pop_back();
```



```
            cout << "\n The deleted element is: " << item;
            break;
        case 5:cout << "\n The size of DEQUEU is: " << dq.size();
            break;
        case 6:cout << "Elements of DEQUE are:  ";
            for (i = dq.begin(); i != dq.end(); i++)//i is for iterator
                cout << *i << "  ";
        }
        cout << "\n Do you want to continue?(y/n): ";
        cin >> ans;
    } while (ans == 'y' || ans == 'Y');
}
```

6.4 : Associative Container

Q.7 What is associative container ?

Ans. : The associative container allows efficient retrieval of values based on keys. The key can be usually an integer or string value using which the data can be arranged in ascending or descending order.

There are various types of associative containers - Set, multiset, map and multimap.

Q.8 Explain the terms sets, multi-sets, map and multi-maps with its declaration syntax.

Ans. :

i) **Set :** Set is a container that contains unique elements in some specific order. We can perform various operations on set such as insertion or deletion of element, searching particular element from the set, displaying of size of the set and so on.

Syntax : set<object_type> set_name

ii) **Multi-sets :** Multi-sets are similar to sets but the difference between set and multi-set is the set does not allow duplicating elements but multiset allows duplicating elements.

Syntax : multiset <object_type> multiset_name

iii) **Map :** The map is an associative container in which the elements are stored in the form of key value and mapped value. For example -
{(1, a), (2, b), (3, c)}

Syntax : map<key,value> map_name

iv) **Multimaps :** It is associative container which is used for fast retrieval of values using the keys. Thus similar to maps multimaps also contain values in the form of key-value pair.

The difference between map and multimaps is that multimap allows the duplicate values. That means multiple values can be associated with a single key.

multimap<key,value> multimap_name

Q.9 Write a C++ code to perform various set operations.

Ans. :

```
Switch (choice)
{
case 1:cout << "\n Enter the element to be inserted: ";
    cin >> item;
    s.insert(item);
    break;
case 2:cout << "\n Enter the element to be deleted: ";
    cin >> item;
    s.erase(item);
    break;
case 3:count = s.size();
    cout << "\n The size of set is: " << count;
    break;
case 4:i = s.find(item);
if (i != s.end())
       cout << "Element " << *i << " is present in the set" << endl;
    else
       cout << "Element is not present in the set" << endl;
    break;
case 5:cout << "Elements of SET are:  ";
    for (i = s.begin(); i != s.end(); i++)//i is for iterator
```



```
       cout << *i << "  ";
}
```

Q.10 Explain various operations that can be performed on map.

Ans. : • Various operations that can be performed on map are -

Function name	Purpose
insert(pair)	The element is inserted into the map.
erase(iterator i)	The element pointed by the iterator is deleted from the map.
void swap(map)	Swaps the contents of the map.
void clear()	Clears the contents.
size_type size()	Returns the size of the container.

6.5 : Container Adapter

Q.11 Explain the term - Container adapter.

Ans. : There are some classes that are derived from the sequence containers. These are sometimes known as **derived container or container adapters.**

Examples of them are - stack, queue, and priority queue.

Q.12 Write a C++ implementation of stack operations using STL.

☞ **[SPPU : June-22, Marks 6]**

Ans. :

```
#include<iostream>
#include<stack> //header file stack must be included
using namespace std;
int main()
{
   stack<int> s; //object is created for stack class
//Note that this class handles the int values
    int item;
    char ans;
```

```
int choice;
do
{
   cout<<"\n Main Menu";
   cout<<"\n 1.Push ";
   cout<<"\n 2.Pop ";
   cout<<"\n 3.Display ";
   cout<<"\n Enter your choice ";
   cin>>choice;
   switch(choice)
   {
      case 1: cout<<"\n Enter the element to be pushed ";
      cin>>item;
      s.push(item);//pushing the element onto the stack
      cout<<"\n Item is pushed!!!";
      break;
      case 2: if(!s.empty())//checking stack is empty or not
      {
         s.pop();//popping the element from the stack
         cout<<"\n Popped an item!!!";
      }
      else
         cout<<"\n stack empty!!!";
      break;
      case 3:if(!s.empty())
            {
             cout<<"Top element of the stack is "<<s.top();
            }
         else
            cout<<"\n The stack is empty!!!";
         break;
   }
   cout<<"\n Do you want to continue? ";
   cin>>ans;
```

Displaying the stack top element


```
   }while(ans=='y');
   return 0;
   }
```

Q.13 Write a C++ implementation of priority queue operations using STL.

Ans. :

```
#include<iostream>
#include<queue> //header file queue must be included
using namespace std;
int main()
{
   priority_queue<int> pq;
   int item;
   char ans;
   int choice;
   do
   {
      cout << "\n Main Menu";
      cout << "\n 1.Insert element ";
      cout << "\n 2.Delete element ";
      cout << "\n 3.Size of queue";
      cout << "\n 4.Display of queue";
      cout << "\n Enter your choice ";
      cin >> choice;
      switch (choice)
      {

      case 1: cout << "\n Enter the element to be inserted: ";
         cin >> item;
         pq.push(item);//inserting the element onto the queue
         cout << "\n Item is inserted!!!";
         break;
      case 2: item = pq.top();
         pq.pop();//Deleting element from queue
```

```
            cout << "\n Deleted item is"<<item;
            break;
        case 3:cout << "\n Size of queue = " << pq.size();
            break;
        case 4:while(!pq.empty())
                {
                    cout << pq.top() << " ";
                    pq.pop();
                }
        }
        cout << "\n Do you want to continue? ";
        cin >> ans;
    } while (ans == 'y');
    return 0;
}
```

Q.14 Write a C++ program to implement Queue using STL.

Ans. :

```
#include<iostream>
#include<queue> //header file queue must be included
using namespace std;
int main()
{
    queue<int> q; //object is created for stack class
                  //Note that this class handles the int values
    int item;
    char ans;
    int choice;
    do
    {
        cout << "\n Main Menu";
        cout << "\n 1.Insert element ";
        cout << "\n 2.Delete element ";
        cout << "\n 3.Display Rear element";
        cout << "\n 4.Display Front element";
```



```
        cout << "\n 5.Size of queue";
        cout << "\n Enter your choice ";
        cin >> choice;
        switch (choice)
        {

        case 1: cout << "\n Enter the element to be inserted: ";
            cin >> item;
            q.push(item);//inserting the element onto the queue
            cout << "\n Item is inserted!!!";
            break;
        case 2: item = q.front();
            q.pop();//Deleting element from queue
            cout << "\n Deleted item is"<<item;
            break;
        case 3:cout << "The element at rear end of queue is " <<
q.back();
            break;
        case 4:cout << "The element at front end of queue is " <<
q.front();
            break;
        case 5:cout << "\n Size of queue = " << q.size();
            break;
        }
        cout << "\n Do you want to continue? ";
        cin >> ans;
    } while (ans == 'y');
    return 0;
}
```

Q.15 What is container ? List container classes in C++. Explain any one of them using program. ☞ [SPPU : Dec.-19, Marks 7]

Ans. : Refer Q.11 and Q.14.

6.6 : Algorithms

Q.16 What is algorithms in STL ? Explain various types of algorithms in STL.

Ans. : The algorithms are used to process the contents of the containers. The functionalities provided in container are not sufficient to perform complex operations hence the algorithms are used to support more complex operations for the containers.

Q.17 What are various categories of algorithm in STL ?

Ans. : Various categories of algorithm are -

1. **Sorting algorithms** - These algorithms contain the functionalities related to sorting of the list.
2. **Mutating sequence algorithms** - These algorithms modify the contents of the container. For example copy() operation will modify the contents of the container.
3. **Nonmutating sequence algorithms** - These algorithms do not modify the contents of the container as they work. For instance count() will simply count the occurrences in the container.
4. **Numerical algorithms** - These algorithms are useful for performing some computations. For instance sum of all the elements can be obtained by the function accumulate().

Q.18 Enlist mutating and non mutating algorithms.

Ans. :

Mutating algorithm

Sr. No.	Function	Description
1.	copy()	Copies the sequence of elements.
2.	copy_backward()	Copies the sequence of elements from end of the list, i.e. copies the list in backward direction.

3.	reverse()	This function is used to reverse the given sequence of elements.
4.	unique()	It finds the adjacent duplicate elements and removes them.

Nonmutating algorithm

Sr. No.	Function	Description
1.	find()	It will find the position of desired element.
2.	count()	This function counts the number of elements in the given sequence.
3.	equal()	It checks whether the two sequences are equal or not. If two sequences are matching then it returns true.
4.	search()	This operation is used for searching the desired element from the given sequence.

Q.19 Write a C++ program to perform binary search using STL.

Ans. :

```
#include <iostream>
#include <algorithm>
using namespace std;
int a[] = { 10,20,30,40,50,60,70,80 };
int key;
char ans='y';
int main()
{
    cout << "\nArray a[8]= ";
    for (int i = 0; i < 8; i++)
        cout << " " << a[i];
```

```
    do
    {
        cout << "\n Enter the element to be searched: ";
        cin >> key;
        if (binary_search(a, a + 8, key))
            cout << "\n The " << key << " is present";
        else
            cout << "\n The " << key << " is not present";
        cout << "\n Do you want to continue?(y/n) ";
        cin >> ans;
    } while (ans == 'y');
    return 0;
}
```

Q.20 Explain how sorting algorithm can be used to sort the list of elements using STL.

Ans. : Following program can be used to use the sort function provided by the STL

```
#include <iostream>
#include <algorithm>//keyword algorithm included
#define SIZE 10
using namespace std;
void main()
{
    int n,item,array[SIZE], i;
    cout<<"How Many Elements You Want to Enter";
    cin>>n;
    int *Limit= array + n; //setting the range for sorting
    cout<<"Enter The Numbers";
    for (i = 0; i < n; ++i)
    {
     cin>>item;
     array[i] = item;
    }
    sort(array, Limit);      //calling the function from algorithm
```



```
    cout<<"\n The sorted list is ..."<<endl;  //displaying the sorted
                                               //list of elements
    for (i = 0; i < n; ++i)
     cout << array[i] << '\t';
}
```

Q.21 Explain the minmax() function used in STL algorithm.

Ans. : • The **minmax()** is a unique function to retrieve min and max values from two values or from an entire array.

• With the keyword **auto** we don't need to tell what is the type of each element.

```
#include <iostream>
#include <algorithm>
using namespace std;
int main()
{
    auto a = { 10,20,30,40,50 };
    auto result = minmax(a);
    cout << "\n Using minmax(10,20,30,40,50)\n";
    cout << "\n The minimum element =" << result.first;
    cout << "\n The maximum element =" << result.second;
    return 0;
}
```

Q.22 Write a C++ program in STL to find out maximum of two elements.

Ans. :

```
#include <iostream>
#include <algorithm>
using namespace std;
int main()
{
    cout << "\n max(20,10)= " << max(20, 10);
    cout << "\n max('t','z')= " << max('t', 'z');
```

```
    cout << "\n max(88.99,100.44)= " << max(88.99, 100.44);
    return 0;
}
```

6.7 : Set Operations

Q.23 What are various set operations. Give the implementation of these operations using STL.

Ans. : Various set operations are union, intersection, difference and symmetric difference. For example

Consider set

$$A = \{10,20,30,40\}$$

$$B = \{10,30,50,60\}$$

$$A \cup B = \{10,20,30,40,50,60\} \quad \text{//Union}$$

$$A \cap B = \{10,30\} \quad \text{//intersection}$$

$$A\text{-}B = \{20,40\} \quad \text{//difference}$$

$$B\text{-}A = \{50,60\} \quad \text{//difference}$$

```
#include <algorithm>
#include <vector>
using namespace std;
void main()
{
    int A[] = { 10,20,30,40 };
    int B[] = { 50,60,30,10 };
    vector<int> v(10);
    vector<int>::iterator it;
    sort(A, A + 4);
    sort(B, B + 4);
    cout << "\n Set A = ";
    for (int j = 0; j << 4; j++)
        cout << "  " << A[j];
    cout << "\n Set B = ";
    for (int j = 0; j << 4; j++)
```



```
        cout << "  " << B[j];
    it = set_union(A, A + 4, B, B + 4, v.begin()); //Union operation
    for (it = v.begin(); it != v.end(); ++it)
        cout << ' ' << *it;
    it = set_intersection(A, A + 4, B, B + 4, v.begin()); //intersection
                                                          //operation
    for (it = v.begin(); it != v.end(); ++it)
        cout << ' ' << *it;
    it = set_difference(A, A + 4, B, B + 4, v.begin());//difference
                                                       //operation
    for (it = v.begin(); it != v.end(); ++it)
        cout << ' ' << *it;
}
```

6.8 : Heap Sort

Q.24 What is Heap ? Explain types of heaps.

Ans. : Definition : Heap is a **complete binary tree** or a almost complete binary tree in which every **parent node** be either **greater** or **lesser than** its child nodes.

A **Max heap** is a tree in which value of each node is greater than or equal to the value of its children nodes.

For example :

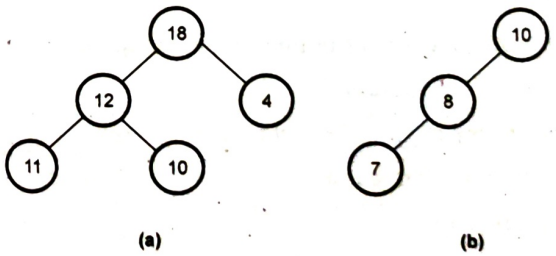


Fig. Q.24.1 Max heap

A **Min heap** is a tree in which value of each node is less than or equal to value of its children nodes.

For example :

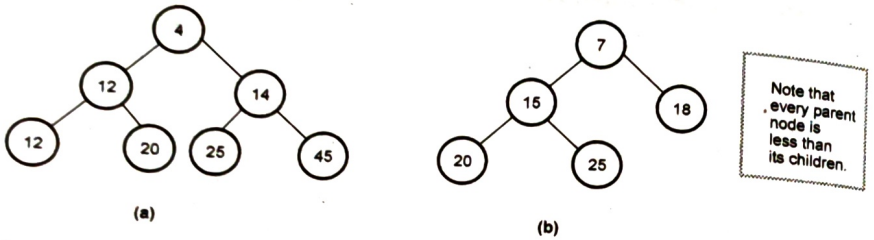


Fig. Q.24.2 Min heap

We can construct heap using top down approach with repeated insertion of element.

Q.25 Give C++ implementation of heap sort using STL.

☞ **[SPPU : Dec.-19, Marks 7]**

Ans. :

```
int main()
{
   int a[] = { 14,10,67,29,57,12,32, };
   vector<int> v(a, a + 7);
   cout<<"\n\t\t Heapifying....";
   make_heap(v.begin(), v.end());//makes the heap from range of
                                 //elements
   cout << "\n\n The root of max heap   : "<< v.front() << '\n';
   cout << " List after heap operation is... \n";
   for (unsigned i = 0; i<v.size(); i++)
      cout << ' ' << v[i];
   cout<<"\n\t\t Sorting....";
```



```
   sort_heap(v.begin(), v.end());//sorts heap
   cout << "\n sorted List is... \n";
   for (unsigned i = 0; i<<v.size(); i++)
      cout << ' ' << v[i];
   return 0;
}
```

6.9 : Iterators - Input, Output, Forward, Bidirectional and Random Access

Q.26 What is iterator ? Explain its types. ☞ **[SPPU : June-22, Marks 4]**

Ans. : The iterators are used to traverse the contents of container. There are five types of iterators.

Iterator	Description
Random access	Elements can be stored or retrieved randomly.
Forward	The elements can be stored or retrieved but only forward moving is allowed.
Input	Elements retrieving is allowed with forward moving.
Output	Elements storing is allowed with forward moving.
Bidirectional	Store and retrieve the elements and forward/backward moving is allowed.

Q.27 Write a program using STL to demonstrate the use of iterators.

Ans. : Following program demonstrates how to use iterator for iterating the list of integers.

```
#include <iostream>

#include <list>
```

```
using namespace std;

int main()
{

    list<int> lst;
    for (int i = 1; i <= 5;i++)
        lst.push_back(i);
    // iterate over all elements and print, separated by space
    cout << "\n Iterating through list...\n";
    list<int>::const_iterator it;
    for (it = lst.begin(); it != lst.end(); ++it)
        cout << *it << ' ';
    cout << endl;
    return 0;
}
```

Output

```
Iterating through list...
1 2 3 4 5
```


6.10 : Object Oriented Programming - A Road Map to Future

Q.28 Why object oriented programming is considered to be roadmap to future.

Ans. : • C++ plays an important role for **object oriented modeling and design(OOMD)**. The Unified Modeling Language (UML) take full advantages of all the features offered by C++.

- There are certain features such as polymorphism, overloading of operators, templates, STL, Runtime Type Identification(RTTI), dynamic type casting, Strings and I/O streams with the support for international character set and localization - and so on that has a great support for generation of runtime efficient applications.
- C++ is a natural choice for robotic programming because **robotics** require high performance code. For handling low level motor controlling routines, the C++ language is useful.

END...

JUNE-2022 (END SEM) [5869] - 273 | Solved Paper

Course 2019

Time : $2\frac{1}{2}$ Hours] **[Maximum Marks : 70**

Instructions to the candidates :

1) ***Attempt questions Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.***

2) ***Draw neat and clean diagram.***

3) ***Assume suitable data, if necessary.***

Q.1 a) *Differentiate between compile time polymorphism and runtime polymophism along with examples.*

(Refer Q.18 of Chapter - 3) **[5]**

b) *Explain the use of mutual and explicit keywords in C++ language along with example.*

(Refer Q.12 of Chapter - 3) **[6]**

c) *Write a program to demostrate for overloading decrement[--] operator in C++. (Assume a class number with data member as int x)* **[6]**

Ans. :

```
#include<iostream>
using namespace std;
class Number {
private:
        int x;
public:
        // Parameterised constructor
        Number(int x = 0)
        {
                this->x = x;
        }
        // Overloading the -- operator
        Number operator--()
        {
                Number temp;
                temp.x = --x;
                return temp;
        }
        // Function to display the value of x
        void display()
        {
                cout << "x = " << x << endl;
        }
};
int main()
{
        Number a(10);
        cout << "Before decrement: ";
        a.display();
        Number b = --a;
        cout << "After decrement: ";
        b.display();
        return 0;
}
```

Output

```
Before decrement : x = 10
After decrement : x = 9
```

OR

Q.2 a) *Explain the concept of method / function overriding along with examples. What is operator overloading in polymorphism ? Write the program to overload - operator for substracting two complex numbers which are object of below complex class by defining operator function in below.* **(Refer Q.22 of Chapter - 3)** **[5]**

b) *Class.*

Class complex{

Private:int real, imag; };

(Refer similar Q.8 of Chapter - 3) **[6]**

c) *What is virtual function and its use ? How we can access virtual functions through the use of pointer to the base class.* **(Refer Q.19 and Q.20 of Chapter - 4)** **[6]**

Q.3 a) *What are cin and cout ? Explain iostream.* **(Refer Q.1 of Chapter - 4)** **[4]**

b) *Write a program that returns the size in bytes of a program entered on the command line.* **[6]**

Ans. :

```
#include<iostream>
#include<fstream>
using namespace std;
int main(int argc, char *argv[])
{
    if(argc!=2)
    {
        cout<<"Error!!! File name is missing";
        return;
```



```
    }
    ifstream inf;
    inf.open(argv[1]);
    inf.seekg(0, ios::end);

     // now print the file pointer
    // position - the size is in bytes
    cout << inf.tellg() << endl;
    return 0;
}
```

c) *What is a stream ? Write a program to illustrate the stream errors.* **(Refer Q.5 of Chapter - 4)** **[8]**

OR

Q.4 a) *What is a file mode ? Describe the various file mode options available.* **(Refer Q.7 of Chapter - 4)** **[4]**

b) *Write a program to create files using constructor function.* **(Refer Q.10 of Chapter - 4)** **[6]**

c) *Write a program using C++ file input and output class with open(), get(), close () for opening, reading from and writing to a file.* **(Refer Q.8 and Q.9 of Chapter - 4)** **[8]**

Q.5 a) *What is need of exception handling ? Explain types of exception.* **(Refer Q.2 of Chapter - 5)** **[4]**

b) *What is template ? Write a program to handle addition of two numbers.* **[6]**

Ans. :

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

```
template<class t1,class t2>
void addition(t1 a,t2 b)
{
   cout<<"\nSum of two numbers = "<<a+b<<endl;
}

int main()
{
   int p,q;
   float x,y;
   cout<<"\nEnter two integer data: ";
   cin>>p>>q;
   cout<<"\nEnter two float data: ";
   cin>>x>>y;
   sum(p,q);
   sum(x,y);
}
```

c) *List and explain four common examples of exceptions.* **(Refer Q.6 and Q.7 of Chapter - 5)** [7]

OR

Q.6 a) *What is difference between class template and function template in C++ ?* **(Refer Q.17 and Q.21 of Chapter - 5)** [4]

b) *Write a function template for finding the minimum value contained in an array.* **(Refer Q.19 of Chapter - 5)** [6]

c) *What are user-defined exceptions ? Explain with suitable example.* **(Refer Q.10 of Chapter - 5)** [7]

Q.7 a) *List and explain different types of iterators in STL.* **(Refer Q.26 of Chapter - 6)** [4]

b) *What is stack ? How is it implemented in STL.* **(Refer Q.12 of Chapter - 6)** [6]

c) *Use minimum 8 functions of Deque STL. Write a program to explain the same.* **(Refer Q.6 of Chapter - 6)** [8]

OR

Q.8 a) *Elaborate advantages and disadvantages of LIST sequential container.* [4]

Ans. : Advantages :

1. There is no wastage of memory.
2. Insertion and deletion operations are easier.

Disadvantages :

1. As LIST is sequential container, we have to traverse sequentially hence searching is in-efficient.

b) *What is meant by associative container ? State and explain the types.* **(Refer Q.7 and Q.8 of Chapter - 6)** [6]

c) *Use minimum 8 functions of vector STL. Write a program to explain the same.* **(Refer Q.5 of Chapter - 6)** [8]

END...