

SUBJECT CODE : 204194

Choice Based Credit System

SAVITRIBAI PHULE PUNE UNIVERSITY - 2019 SYLLABUS

S.E. (E & Tc / Elex.) Semester - IV

OBJECT ORIENTED PROGRAMMING

(For END SEM Exam - 70 Marks)

Mrs. Anuradha A. Puntambekar

M.E. (Computer)

Formerly Assistant Professor in

PE.S. Modern College of Engineering,

Pune

FEATURES

- Written by Popular Authors of Text Books of Technical Publications
- Covers Entire Syllabus Question - Answer Format
- Exact Answers and Solutions
- Chapterwise Solved SPPU Questions Dec.-2016 to Dec.-2022

SOLVED SPPU QUESTION PAPERS

• May - 2017 • Dec. - 2017 • May - 2018 • Dec. - 2018
• May - 2019 • Dec. - 2019 • June - 2022 • Dec. - 2022

DECODE[®]

A Guide For Engineering Students

TABLE OF CONTENTS

Unit III

Chapter - 3	Operator Overloading	(3 - 1) to (3 - 15)
1	Fundamentals of Operator Overloading	3 - 1
2	Restrictions on Operators Overloading.....	3 - 1
3	Operator Functions as Class Members vs. as Friend Functions	3 - 3
4	Overloading Unary Operators	3 - 4
5	Overloading Binary Operators.....	3 - 8
6	Overloading of Operators using Friend Functions	3 - 14

Unit IV

Chapter - 4	Inheritance and Polymorphism	(4 - 1) to (4 - 32)
	Introduction to Inheritance.....	4 - 1
	Base and Derived Classes	4 - 1
	Friend Classes.....	4 - 1
	Types of Inheritance	4 - 3
	Hybrid Inheritance.....	4 - 10
	Member Access Control	4 - 11
	Multiple Inheritance	4 - 12
	Ambiguity	4 - 13
	Virtual Base Class	4 - 15

4.10	Introduction to Polymorphism	4 - 16
4.11	Pointers to Objects.....	4 - 17
4.12	Virtual Functions	4 - 18
4.13	Pure Virtual Functions	4 - 21
4.14	Abstract Base Class.....	4 - 22
4.15	Polymorphic Class	4 - 25
4.16	Virtual Destructors.....	4 - 25
4.17	Early and Late Binding.....	4 - 27
4.18	Container Classes and Contained Classes	4 - 30
4.19	Singleton Class.....	4 - 31

Unit V

Chapter - 5	Templates, Namespaces and Exception Handling	(5 - 1) to (5 - 27)
--------------------	---	----------------------------

Part I : Templates

5.1	Introduction to Templates	5 - 1
5.2	Function Template and Class Template	5 - 2
5.3	Function Overloading vs. Function Templates	5 - 6

Part II : Namespaces

5.4	Introduction to Namespaces.....	5 - 8
5.5	Rules of Namespaces	5 - 9

Part III : Exception Handling

5.6	Basics of Exception Handling.....	5 - 10
5.7	Exception Handling Mechanism	5 - 11
5.8	Throwing and Catching Mechanism.....	5 - 13

5.9	Specifying Exceptions	5 - 14
5.10	Multiple Exceptions	5 - 15
5.11	Exceptions with Arguments	5 - 17
5.12	C++ Streams	5 - 19
5.13	Stream Classes	5 - 20
5.14	Unformatted I/O	5 - 21
5.15	Formatted I/O and I/O Manipulators	5 - 22

Unit VI

Chapter - 6	Working with Files	(6 - 1) to (6 - 22)
6.1	Classes for File Stream Operations	6 - 1
6.2	Opening and Closing Files	6 - 1
6.3	Detecting End_Of_File (EOF)	6 - 4
6.4	File Pointers and Manipulators	6 - 13
6.5	Updating File	6 - 14
6.6	Error Handling During File Operations	6 - 20

Solved SPPU Question Papers (S - 1) to (S - 9)

Unit III

3

Operator Overloading

3.1 : Fundamentals of Operator Overloading

Q.1 What is operator overloading ? Why it is necessary to overload an operator ?

☞ [SPPU : June-22, Dec.-22, Marks 6]

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

- Various types of operators are -
 - Mathematical operators such as + - * / ++
 - Relational operators such as < > ==
 - Logical operators such as && ||
 - Access operators [] ->
 - Assignment operator =
 - Stream I/O operators << >>
 - Type conversion operators and several others.

- All of these operators have a predefined and unchangeable meaning for the built-in types. All of these operators can be given a special interpretation for different classes or combination of classes. C++ provides the flexibility to the programmers in extending these built-in operators.

3.2 : Restrictions on Operators Overloading

Q.2 Discuss the usage of the operators that can not be overloaded in C++.

☞ [SPPU : May-18, Marks 6]

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.
- Makes no assumptions about similar operators. For example, the fact that you overloaded + does not mean that you have also defined += for your class type.

Q.3 What are the rules for over loading operators ?

[SPPU : June-22, Marks 6, Dec.-22, Marks 4]

Ans. :

1. Only existing operators can be overloaded.
2. The basic meaning of the operator can not be changed.
3. Overloaded operators must follow the syntax of original operator. For example for binary operator operand1 operator operand2 is the syntax and this can not be changed during overloading.
4. Overloaded operators must have at least one operand that is of user defined type.
5. Binary arithmetic operators(+, -, * and /) must return a value.
6. When binary operators overloaded through a member function, the left hand operand must be an object of relevant class.

7. Binary operators overloaded through a member function must take one explicit argument.
8. Binary operators overloaded through friend function takes two arguments.
9. Unary operators overloaded through a member function must take no explicit argument and no return value.
10. Unary operators overloaded through friend function takes one explicit argument.

3.3 : Operator Functions as Class Members vs. as Friend Functions

Q.4 Compare operator overloading using operator function as class member with operator overloading as friend function.

Ans : Operator overloading can be achieved in two ways -

- 1) By an operator overloading by member function
- 2) By an operator overloading non-member friend function

Sr. No.	Operator function as class member function	Operator function as friend function
1.	An overloaded operator function should be declared in the public section of a class.	An overloaded operator friend could be declared in either private or public section of a class.
2.	When an operator overloaded function is a member function, it takes one operand of user-defined data type.	When an operator overloaded function is a friend function, it takes two operands of user-defined data type.

<p>3.</p> <p>Declaration : Suppose Test is a class name, then</p> <p>Test operator +(Test);</p> <p>Definition :</p> <p>Test operator +(Test obj)</p> <pre>{ Test temp; temp.a = a + a + obj.a; return temp; }</pre>	<p>Declaration : Suppose Test is a class name, then</p> <p>friend Test operator +(Test,Test);</p> <p>Definition :</p> <p>Test operator +(Test obj1, Test obj2)</p> <pre>{ Test temp; temp.a = obj1.a + obj2.a; return temp; }</pre>
---	---

3.4 : Overloading Unary Operators

Q.5 Explain operator overloading of unary operator.

Ans. : • The unary operators require only one operand. In C++ the unary operators are

-, !, ~, & and *.

- It can be declared as **member functions** taking no arguments. That means for any operator -, - obj is interpreted as **obj.operator()**
- It can be declared as **non member functions** taking one argument that must be the variable of class type (i.e. Object) or reference. That means, for any operator - the - obj is interpreted as **operator-(obj)**.
- If both types of definitions are present then, the function declared as **member** takes the precedence.

Syntax : Function definition for operator overloading

Return Type :: Operator symbol (Arguments..list)

Class name or basic data type Keyword Operator symbols like +, - etc.

```
{
function body
}
```

Q.6 Write a C++ program for overloading a unary minus operator.

Ans. :

```

/*****
Program for overloading an unary minus operator
*****/
#include <iostream>
using namespace std;
class point
{
private:
    int x, y; // co-ordinate values
public:
    point() { x = 0; y = 0; } //constructor
    point(int i, int j) { x = i; y = j; } //constructor
    void get_xy(int &i, int &j) { i = x; j = y; }
    point operator-( ); // operator overload for unary
                        // minus
};
point point::operator-( )
{
    x = - x;
    y = - y;
    return *this; // Use of this pointer
}

```

```

}

int main( )
{
    point obj(10, 10);
    int x, y;
    clrscr();
    obj.get_xy(x, y);
    obj = - obj;
    obj.get_xy(x, y);
    cout<<"\n The use of unary operator is ..."<<endl;
    cout << " X: " << x << ", Y: " << y;
    return 0;
}

```

Negation calls operator - ()

Output

The use of unary operator is ...

X: -10, Y: -10

Program explanation :

- Note that in above code the unary minus operator function is overloaded by passing no argument to it.

obj = -obj;

- When this statement occurs the call to operator -() function is given. Thereby negated values of x and y are obtained.

Q.7 Write a C++ program for overloading an 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();
    return 0;
}

```

Calls coord operator++()

Output

Enter the co-ordinates x and y 10 20

The increment operator increments the co-ordinates as...

X: 11, Y: 21

Q.8 Write down program to overload unary operators.

[SPPU : June-22, Dec.-22, Marks 6]

Ans. : Refer Q.7.

3.5 : Overloading Binary Operators

Q.9 Explain overloading of binary operator with suitable example.

Ans. : • For the operator function which is actually the member function of a class, requires one argument which is actually the right argument.

Syntax

Classname **operator** *oper-symbol* (class name)

For example : For the operator +, we can invoke operator overloaded function as

```
obj3 = obj1 + obj2; // call
```

The definition of above statement is

```
classname operator + (obj2)
```

```
{
// body of function
}
```

- Following C++ program illustrates how to overload + operator to add two objects

C++ program

```
#include <iostream>
using namespace std;
class vector
{
public:
    int a, b;
    vector() { a = 0; b = 0; } // constructor without
        // parameters
    vector(int, int); // constructor with parameters
```

Classname
as return
type

```
vector operator + (vector);
```

One argument
Declaration of operator function
Operator symbol
Keyword

```
};
vector::vector(int x, int y)
```

```
{
    a = x;
    b = y;
}
```

It refers to
second operand

```
vector vector::operator+ (vector obj)
{
    vector temp;
    temp.a = a + obj.a;
    temp.b = b + obj.b;
    return (temp);
}
```

Definition of
operator
function

```
void main()
```

```
{
    vector v1(10, 20);
    vector v2(1, 2);
    vector v3;
    v3 = v1 + v2;
```

```
cout << "\n The Addition of Two vectors is...";
cout << v3.a << " and " << v3.b;
```

Output

The Addition of Two vectors is...11 and 22

Q.10 Which operators can not be overloaded ? Write steps to overload + operator so that it can add two complex numbers.

[SPPU : June-22, Dec.-22, Marks 6]

Ans. : Operators that can not be overloaded : Refer Q.2.

Overloading of + operator for adding two complex numbers :

```
#include <iostream>
using namespace std;
class complex
{
public:
    float real,img;
    complex(){real=0;img=0;}
    complex(float,float); // parameterized constructor
    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);
}
int main()
{
    complex a(2,6);
    complex b(4,1);
    complex c;
    cout << "\n The first Complex number is: ";
    cout << a.real << " and " << a.img << "i";
```

```
cout << "\n The second Complex number is: ";
cout << b.real << " and " << b.img << "i";
c=a+b;
cout << "\n The addition of two complex numbers is...";
cout << c.real << " and " << c.img << "i";
return 0;
```

Output

The first Complex number is: 2 and 6i

The second Complex number is: 4 and 1i

The addition of two complex numbers is...6 and 7i

Q.11 Write a C++ program to concatenate two strings using operator overloading on + operator.

Ans. :

```
#include <iostream>
#include <cstring>
using namespace std;
class string1
{
public:
    char S[15];
    string1()
    {
        strcpy(S,"0");
    }
    string1(char T[15])
    {
        strcpy(S,T);
    }
    string1 operator+(string1 k)
    {
        strcat(S,k.S);
        strcat(S,"\0");
        return S;
    }
}
```



```

};
int main()
{
    string s1("Hello"),s2("Friends");
    string s;
    s=s1+s2;
    cout<<s.S<<endl;
    return 0;
}

```

Q.12 Define a class DATE, use overloaded + operator to add two dates and display the resultant-date. Assume non-leap year dates.

Ans. :

```

#include <iostream>
using namespace std;
class DATE
{
private:
    int dd;
    int mm;
    int yy;
public:
    DATE(){}
    DATE(int d,int m,int y)
    {dd=d,mm=m,yy=y;}
    DATE operator+(DATE);
    void display();
};
DATE DATE::operator+(DATE D)
{
    DATE temp;
    int day,flag=0;
    int month;
    day=dd+D.dd;
    if(day>30) //total days exceeding month
    {

```

```

        day=day-30;
        flag=1; //flag 1 means sum of days exceeds one month
    }
    temp.dd=day;
    month=mm+D.mm;
    if(month>12)
    {
        if(flag==1)
            month=(month+1)-12;
        else
            month=month-12;
    }
    temp.mm=month;
    if(yy==D.yy)
        temp.yy=D.yy;
    else
        temp.yy=yy+D.yy;
    return temp;
}
void DATE::display(void)
{
    cout<<"\n";
    cout<<" Day: "<<dd;
    cout<<" Month: "<<mm;
    cout<<" Year: "<<yy;
}
int main()
{
    DATE d1,d2,d3;
    d1=DATE(30,10,10);
    d2=DATE(5,5,01);
    d3=d1+d2;
    d1.display();
    d2.display();
    d3.display();
}

```

```
return 0;
```

3.6 : Overloading of Operators using Friend Functions

Q.13 Explain with suitable example, how to overload an operator using friend function ?

[SPPU : Dec.-22, Marks 6]

Ans. : • The friend functions are not the members of a class, similarly they do not have this pointer. Hence all the operands of the operator must be passed explicitly to the friend operator function.

```
/******
Program for overloading the operator > for comparing two values
******/
```

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
class GreaterOp
```

```
{
    int a;
    public:
    GreaterOp(){
    GreaterOp(int x)
    {a=x;}
    friend int operator >(GreaterOp obj1, GreaterOp obj2);
};
```

```
int operator >(GreaterOp obj1, GreaterOp obj2)
```

```
{
    if(obj1.a > obj2.a)
    return 1;
    else
    return 0;
}
```

```
int main()
{
    GreaterOp val1(1);
    GreaterOp val2(10);
    if(val1 > val2)
        cout << "\n The first value is greater than the second";
    else
        cout << "\n The second value is greater than the first";
    return 0;
}
```

Output

The second value is greater than the first

END... ✍

Unit IV

4

Inheritance and Polymorphism

4.1 : Introduction to Inheritance

Q.1 Define the term - Inheritance. [SPPU : Dec.-22, Marks 6]

Ans. : Definition : Inheritance is a property in which data members and member functions of some class are used by some other class.

- Inheritance allows the reusability of the code in C++.

4.2 : Base and Derived Classes

Q.2 Explain the terms - Base and derived class.

Ans. : • The class from which the data members and member functions are used by another class is called the **base class**.

- The class which uses the properties of base class and at the same time can add its own properties is called **derived class**.

4.3 : Friend Classes

Q.3 Explain the concept of friend class with suitable example.

Ans. : • A class can be declared as a friend class. This allows the friend class to access the private data members of the another class. In the following program class B is declared as friend of class A. Therefore class B can access the variable *data*, and variable is private member of class A.

C++ Program

```
#include <iostream>
using namespace std;
class A
{
    private:
        int data;
        friend class B; // class B is friend of class A
    public:
        A() // constructor
        {
            data = 5;
        }
};
class B
{
    public:
        int sub(int x)
        {
            A obj1; // object of class A
            // the private data of class A is accessed in class B
            // data contains 5 and x contains 2
            return obj1.data - x;
        }
};
int main()
{
    B obj2;
    cout << "Result is " << obj2.sub(2);
    getch();
    return 0;
}
```

Output

Result is = 3

4.4 : Types of Inheritance

Q.4 Explain different types of inheritance.

Ans. : • Various types of inheritance are -

- 1) Single inheritance
- 2) Multilevel inheritance
- 3) Multiple inheritance
- 4) Hybrid inheritance
- 5) Hierarchical inheritance

Q.5 Explain the single inheritance with a suitable example.

[SPPU : Dec.-22, Marks 6]

Ans. : • In single inheritance there is one parent per derived class. This is the most common form of inheritance.

- The simple program for such inheritance is -

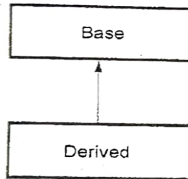


Fig. Q.5.1 Single Inheritance

C++ Program

```

#include <iostream>
using namespace std;
class Base
{
public:
    int x;
    void set_x(int n)
    {
        x = n;
    }
    void show_x()
    {
        cout << "\n\t Base class ...";
    }
}
  
```

```

        cout << "\n\t x = " << x;
    }
};
class derived : public Base
{
    int y;
public:
    void set_y(int n)
    {
        y = n;
    }
    void show_xy()
    {
        cout << "\n\n\t Derived class ...";
        cout << "\n\t x = " << x;
        cout << "\n\t y = " << y;
    }
};
int main()
{
    derived obj;
    int x, y;
    cout << "\n Enter the value of x";
    cin >> x;
    cout << "\n Enter the value of y";
    cin >> y;
    obj.set_x(x); // inherits base class
    obj.set_y(y); // access member of derived class
    obj.show_x(); // inherits base class
    obj.show_xy(); // access member of derived class
    return 0;
}
  
```

Output

Enter the value of x 10

Enter the value of y 20

Base class ...

x = 10

Derived class ...

x = 10

y = 20

Q.6 Explain multi-level inheritance.

Ans. : • When a derived class is derived from a base class which itself is a derived class then that type of inheritance is called multilevel inheritance.

- For example - If class A is a base class and class B is another class which is derived from A, similarly there is another class C being derived from class B then such a derivation leads to multilevel inheritance.
- The implementation of multilevel inheritance is as given below -

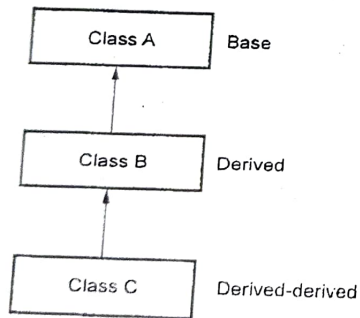


Fig. Q.6.1 Multilevel Inheritance

C++ Program

```

#include <iostream>
using namespace std;
class A
{
    protected:
        int x;
    public:
        void get_a(int);
        void put_a();
};
void A::get_a(int a)
{
    x=a;
}
void A::put_a()
{
    cout<<"\n The value of x is "<<x;
}
class B:public A
{
    protected:
        int y;
    public:
        void get_b(int);
        void put_b();
};
void B::get_b(int b)
{
    y=b;
}
void B::put_b()
{
    cout<<"\n The value of y is "<<y;
}
  
```

```

class C:public B
{
    int z;
    public:
        void display();
};
void C::display()
{
    z=y+10;
    put_a();//member of class A
    put_b();//member of class B
    cout<<"\n The value of z is "<<z;
}
int main()
{
    C obj;//object of class C
    //accessing class A member via object of class C
    obj.get_a(10);
    //accessing class B member via object of class C
    obj.get_b(20);
    ///accessing class C member via object of class C
    obj.display();
    cout<<endl;
    return 0;
}

```

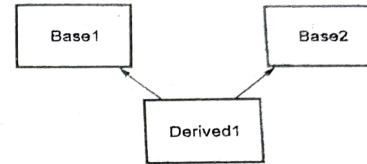
Output

The value of x is 10
 The value of y is 20
 The value of z is 30

Q.7 Explain multiple inheritance with suitable C++ program.

Ans. : • In multiple inheritance the derived class is derived from more than one base class.

- The implementation of multiple inheritance is as shown in Fig. Q.7.1.

**Fig. Q.7.1 Multiple Inheritance****C++ Programs**

```

#include <iostream>
using namespace std;
class Operation
{
    protected:
        int x, y;
    public:
        void set_values (int a, int b)
        {
            x=a;
            y=b;
        }
};

class Coutput
{
    public:
        void display (int i);
};

void Coutput::display (int i)
{
    cout << i << endl;
}

```

```

//product class inherits two base classes -
//Operation and Coutput
class product: public Operation, public Coutput
{
    public:
        int function ()
        {
            return (x * y);
        }
};

//sum class inherits two base classes -
//Operation and Coutput
class sum: public Operation, public Coutput
{
    public:
        int function ()
        {
            return (x + y);
        }
};

int main ()
{
    product obj_pr;//object of product class
    sum obj_sum;//object of sum class
    obj_pr.set_values (10,20);
    obj_sum.set_values (10,20);
    cout << "\n The product of 10 and 20 is " << endl;
    obj_pr.output (obj_pr.function());
    cout << "\n The sum of 10 and 20 is " << endl;
    obj_sum.output (obj_sum.function());
    return 0;
}

```

Output

The product of 10 and 20 is
200

The sum of 10 and 20 is
30

4.5 : Hybrid Inheritance

Q.8 Define inheritance. Explain different types of inheritance.

 [SPPU : Dec.-17, 18, Marks 3 , May-18, June-22, Marks 8]

Ans. : Inheritance : Refer Q.1.

Types : Refer Q.5, Q.6, Q.7.

Q.9 Explain inheritance and its types in C++. Explain multiple inheritance in detail with syntax.


 [SPPU : May-19, Marks 8]

Ans. : Inheritance : Refer Q.1.

Types : Refer Q.4.

Multiple inheritance : Refer Q.7.

Q.10 Discuss the ways in which inheritance promotes software use, saves time during program development and helps prevent errors.

 [SPPU : June-22, Marks 8]

Ans. : • Inheritance allows developers to create derived classes that reuse code declared already in a base class.

- When a inheritance hierarchy is built using several classes, then it avoids the duplication of common functionality. This saves considerable amount of development time spend by developers.
- When we place a common functionality in a single base class (super class) instead of duplicating the code in derived class, helps to prevent the same errors from appearing in multiple source code files.

- If several classes each contain duplicate code containing an error the software developer has to spend time correcting each source code file with the error.
- However, if inheritance technique is used, then if the error that occurs in the common functionality of the base class, the software developer need to modify only the base class's code.
- Thus use of super class for common functionalities and different functionalities in derived classes allow promote software reuse and saves time during program development and helps prevent errors.

4.6 : Member Access Control

Q.11 Explain public, private and protected keywords using program.

[SPPU : Dec.-17, Marks 6]

Ans. : In C++ there are three access specifiers -

- 1) **public** : When the public access specifier is used for some member of a class, then that member is accessible outside the class.
- 2) **private** : When the private access specifier is used for some member of a class, then that member is not accessible outside the class.
- 3) **protected** : Members can not be accessible outside the class, however, they can be accessible in inherited classes.

C++ Program using public, private and protected :

```
#include <iostream>
#include <fstream>
using namespace std;
class test

public: // Public access specifier
    int x; // Public attribute
    void fun1()
    {
        cout<<"Public function";
    }
protected:
```

```
int y;
void fun2()
{
    cout<<"Protected function";
}
private: // Private access specifier
int z; // Private attribute
void fun3()
{
    cout<<"Private function";
}
};

int main()
{
    test obj;
    obj.x = 10;
    obj.fun1();
    obj.y=20;//error can't be accessed outside the class
    obj.fun2();//error can't be called outside the class
    obj.z=30;//error can't be accessed outside the class
    obj.fun3();//error can't be called outside the class
    return 0;
}
```

Q.12 Discuss the role of access specifiers in inheritance and show their visibility when they are inherited as public, private and protected.

[SPPU : June-22, Marks 6]

Ans. : Refer Q.11.

4.7 : Multiple Inheritance

Q.13 Give suitable example to demonstrate multiple inheritance.

Ans. : Refer Q.7.

4.8 : Ambiguity

Q.14 What is multiple inheritance ? What is ambiguity in multiple inheritance ? Give suitable example to demonstrate multiple inheritance.

[SPPU : May-19, Marks 6]

Ans. : Multiple inheritance : Refer Q.7.

Ambiguity : • Ambiguity is the problem that arise in multiple inheritance. It is also called as **diamond problem**.

- Consider the following Fig. Q.14.1.

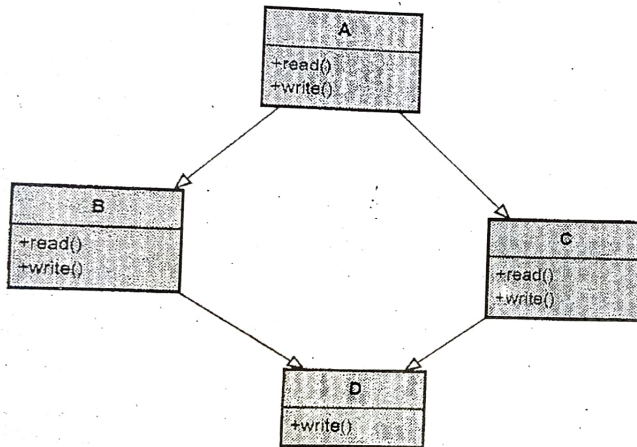


Fig. Q.14.1 Ambiguity In Inheritance

- Now the code for the above design can be written as

```

class A
{
public:
    read();
    write();
};
  
```

```

class B:public A
{
public:
    read();
    write();
};
class C:public A
{
public:
    write();
};
class D:public B,public C
{
public:
    write();
};
  
```

- We try to inherit the `write()` function of base class A in class B and C, which will be alright. But if try to use the `write()` function in class D then the compiler will generate error, because it is ambiguous. We don't know which `write()` function to choose whether of class B or class C. This ambiguity occurs because the compiler understands that the class D is derived from both class B and class C and both of these classes have the versions of `write()` function. So the A class gets duplicated inside the class D object.
- The compiler will complain when compiling the code: error: 'request for member "write" is ambiguous', because it can't figure out whether to call the method `write()` from `A::B::D` or from `A::C::D`.
- That means the programming language does not allow us to represent the concept as given in the design.
- C++ allows the solving of this problem by using **virtual inheritance**. This process is also called as **disinheritance**.
- In order to prevent the compiler from giving error due to multiple inheritance we use the keyword **virtual**. That means base class is made virtual.

4.9 : Virtual Base Class

Q.15 Explain virtual base class and virtual function with example.

[SPPU : June-22, Marks 6]

Ans. : Virtual base class : In order to prevent the compiler from giving an error due to ambiguity in multiple path or multiple inheritance, we use the keyword **virtual**. That means when we inherit from the base class in both derived classes, the base class is made virtual. The code that illustrates the concept of virtual base class is as given below -

C++ Program

```
#include <iostream.h>
class base {
public:
    int i;
};
class derived1:virtual public base
{
public:
    int j;
};
class derived2:virtual public base
{
public:
    int k;
};
//derived3 is inherited from derived1 and derived2
//but only one copy of base class is inherited.
class derived3:public derived1,public derived2
{
public:
    int sum()
    {
        return i+j+k;
    }
}
```

```
};
void main()
{
    derived3 obj;
    obj.i=10;
    obj.j=20;
    obj.k=30;
    cout<<" The sum is = "<<obj.sum();
}
```

Output

The sum is = 60

Virtual function : Refer Q.18.

4.10 : Introduction to Polymorphism

Q.16 What is polymorphism ?

[SPPU : Dec.-22, Marks 6]

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. The programmer does not have to know the exact type of the object in advance hence exact behavior of the object is determined at the run time.
- Polymorphism is concerned with the application of specific implementation or use of abstract base class.

4.11 : Pointers to Objects

Q.17 Write a C++ program to demonstrate pointer to object

Ans. : • Following is a simple C++ program in which a pointer to object variable is used to access the value of the class.

Program to display the contents using the pointer to object.

```
...../
#include <iostream>
using namespace std;
class Test
```

```
int a;
public:
Test(int b)
{
    a = b;
}
int getVal()
{
    return a;
}
```

```
int main()
```

```
Test obj(100), *ptr_obj;
ptr_obj = &obj;
```

```
cout << "Value obtained using pointer to object is ..." << endl;
cout << ptr_obj->getVal() << endl;
return 0;
```

Address of obj is stored in pointer variable.

Output

0

Program explanation

- In above program, the object to the class Test is obj. One pointer variable is created named ptr_obj. This is actually a pointer to the object. This pointer can access the public function of the class Test. Hence we are calling the function getVal of the class Test using the pointer to the object. Thus member function can be accessed using pointer.

4.12 : Virtual Functions

Q.18 What is virtual function ? Explain with example.

☞ [SPPU : Dec.-22, Marks 6]

Ans. : **Definition** : "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. The virtual function can be redefined in the derived class. Thus using virtual functions we can have **one interface, multiple functions** performing **different task**. This feature is called **polymorphism**. The virtual function within the base class defines the form of the interface to the function. Each redefinition of the virtual function by a derived class indicate some different task related to derived class.
- When a virtual function is redefined by a derived class, the keyword virtual is not needed. The virtual function written in base class acts as **interface** and the function defined in derived classes act as different forms of the same function. This property of virtual function brings the **runtime polymorphism**. The following program shows the property -

```
#include <iostream>
using namespace std;
class base
{
public:
    int i;
    base(int x)//constructor
    {
```

```

    i = x;
}
virtual void function()
{
    cout << "Using base version of function():";
    cout << i << "\n";
}
};
class derived1 : public base
{
public:
//constructor for object creation
derived1(int x):base(x){ }
void function()
{
    cout << "Using derived1's version of function():";
    cout << i+i << "\n";
}
};
class derived2 : public base
{
public:
derived2(int x):base(x){ } //constructor for object creation
void function()
{
    cout << "Using derived2's version of function():";
    cout << i*i << "\n";
}
};
int main()
{
    base *p;
    int num;
    cout << "\n Enter Some number ";
    cin >> num;

```

The function is with keyword virtual

```

base obj(num);
derived1 d1_obj(num);
derived2 d2_obj(num);
p=&obj;//base object's address
p->function();//base class function
p = &d1_obj;//derived1 object's address
p->function();//derived1 class function()
p = &d2_obj;//derived2 object's address
p->function();//derived2 class function()
return 0;
}

```

Output

Enter Some number 5

Using base version of function(): 5

Using derived1's version of function(): 10

Using derived2's version of function(): 25

Q.19 Differentiate compile time and run time polymorphism.

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.
----	---	--

4.13 : Pure Virtual Functions

Q.20 Write short note on - Pure virtual function.

Ans. : • 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
- Following program demonstrate the use of pure virtual function.

The class A is an abstract class in which the pure virtual function Display() is declared. This function is overridden by the two derived classes - class B and class C.

```

/*****
Program for demonstrating the pure virtual function
*****/
#include <iostream>
using namespace std;
class A
{
public:
    virtual void Display()=0;
};

class B : public A
{
public:
    void Display()
    {
        cout << "\n In Derived Class B" << endl;
    }
};

```

Pure Virtual Function

```

class C : public A
{
public:
    void Display()
    {
        cout << "\n In Derived Class C" << endl;
    }
};

int main()
{
    A *p;
    B b;
    C c;
    p = &b;
    p->Display(); //using pointer the method of derived class is
                // accessed

    p = &c;
    p->Display();
    return 0;
}

```

Output

In Derived Class B

In Derived Class C

Program explanation : The base class A derived two classes namely B and C. In class A, the pure virtual function Display is defined by assigning = 0.

The Display function is overridden in class B and class C.

4.14 : Abstract Base Class

Q.21 What is abstract base class ? Explain with suitable example.

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.**

For example

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

class area
{
    double dim1, dim2;
public:
    void setarea(double d1, double d2)
    {
        dim1 = d1;
        dim2 = d2;
    }
    void getdim(double &d1, double &d2)
    {
        d1 = dim1;
        d2 = dim2;
    }
    virtual double getarea() = 0; // pure virtual function
};

class square : public area
{
public:
    double getarea()
    {
        double d1, d2;
```

```
        getdim(d1, d2);
        return d1 * d2;
    }
};

class triangle : public area
{
public:
    double getarea()
    {
        double d1, d2;
        getdim(d1, d2);
        return 0.5 * d1 * d2;
    }
};

int main()
{
    area *p;
    square s;
    triangle t;
    int num1, num2;
    cout << "\n Enter The two dimensions for calculating area of
square";
    cin >> num1 >> num2;
    s.setarea(num1, num2);
    p = &s;
    cout << "Area of square is : " << p->getarea() << "\n";
    cout << "\n Enter The two dimensions for calculating area of
triangle";
    cin >> num1 >> num2;
    t.setarea(num1, num2);
    p = &t;
    cout << "Area of Triangle is: " << p->getarea() << "\n";
    return 0;
}
```

Output

Enter The two dimensions for calculating area of square 10 20
Area of square is : 200

Enter The two dimensions for calculating area of triangle 6
8

Area of Triangle is: 24

4.15 : Polymorphic Class

Q.22 What is polymorphic class ? Explain.

Ans. : • Polymorphic class is a class that defines one or more virtual functions. Virtual function is a member function that is declared in a base class and redefined by the derived class.

Example program : Refer program in Q.18.

4.16 : Virtual Destructors

Q.23 Write short note on - Virtual destructor.

Ans. : • Destructor is basically used to de-allocate the memory allocated for the objects. Thus use of destructor is to clean up the memory allocated for the class members. The destructor is denoted using ~ symbol.

- When a class is derived from a base class then on calling the destructor, it does not destruct the memory of derived class. This problem can be fixed up by making the base class destructor virtual.

```
#include <iostream>
using namespace std;
class Base
{
public:
Base()
{
```

```
    cout << "\n Calling Base class Constructor";
}
virtual ~Base()           //Note that this is virtual constructor
{
    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;
    return 0;
}
```

Output

Calling Base class Constructor
Calling Derived class Constructor
Calling Derived class Destructor
Calling Base class Destructor

4.17 : Early and Late Binding

Q.24 Explain the following - Early binding and late binding.

Ans. :

1. **Early binding** : This is a type of binding in which the **function call** is associated with the **function definition** at the **compile time** only. Hence this type of binding is also called as early binding. Following example illustrates this concept -

```
.....
Program to demonstrate the static binding concept
...../
```

```
#include <iostream>
using namespace std;
class Base
{
public :
    void display()
    {
        cout << "\n In base class" << endl;
    }
};
```

```
class Derived : public Base
{
public :
    void display()
    {
        cout << "\n In derived class" << endl;
    }
};
```

```
int main()
{
    Base b;
```

```
Derived d;
d.display();
d.d.display();
return 0;
}
```

Output

In base class

In derived class

2. **Late binding** : C++ provides facility to specify that the compiler should match function calls with the correct definition at the **run time**; this is called **late binding** or **dynamic binding**. This type of binding is called the late binding because the compiler can not resolve the call to appropriate function late until the run time.

Late binding is achieved using **virtual functions**.

Following program illustrates this concept -

```
.....
Program to demonstrate the dynamic binding concept
...../
```

```
#include <iostream>
using namespace std;
class Base
{
public :
    virtual void display()
    {
        cout << "\n In base class" << endl;
    }
};
```

```
class Derived : public Base
{
public :
    void display()
```



```

{
    cout << "\n In derived class" << endl;
}

};

int main()
{
    Base *ptr;
    Base b;
    Derived d;
    ptr = &b;
    ptr->display();
    ptr = &d;
    ptr->display();
    return 0;
}

```

Output

In base class

In derived class

Q.25 Give the difference between early binding and late binding.

[SPPU : Dec.-22, Marks 5]

Ans. :

Sr. No.	Early binding	Late binding
1.	Static binding happens at the compile time.	Late binding happens at run time.
2.	Static binding is also called as early binding.	Late binding is also called as dynamic binding.
3.	There is no use of virtual function in this type of binding.	The virtual function is used in dynamic binding.
4.	It is more efficient than the late binding as extra level of indirection is involved in late binding.	It more flexible than the early binding.

4.18 : Container Classes and Contained Classes

Q.26 Explain the term - Container class and contained class.

Ans. : • When an object of one class is present inside another class or in other words, if the object of one class will be a member of another class, then that type of relationship between the classes is called containership.

- The relationship between these classes is called **has-a** relationship.
- Following program illustrates the concept of container class and contained class -

C++ Program

```

#include <iostream>
using namespace std;

```

class Engine

```

{
    public:
        void display()
        {
            cout << "\nThis is Engine class";
        }
};

```

// Container class

class Car {

```

    // creating object of Engine - contained object
    Engine e;

```

public:

// constructor

Car()

{

```

    cout << "\n This is Car Class";
    e.display();

```

}

};

```
int main()
{
    // creating object of Car
    Car c;
}
```

Output

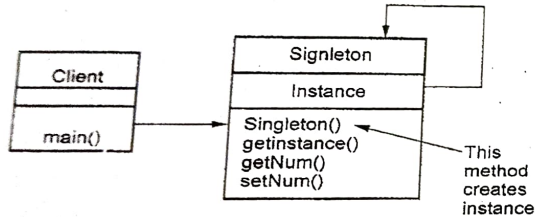
This is Car Class
This is Engine class

4.19 : Singleton Class

Q.27 Explain the concept of singleton class with the help of C++ program.

Ans. : • Singleton class is a type of class that has only one instance and it provides global point of access to it.

- The concept of singleton class is illustrated by following figure -



C++ Program

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

```
{
    static Singleton *instance;
    int num;
    Singleton()
    {
        num = 0;
```

This Function creates an instance of Singleton class. This is a actually a constructor

```
};
public:
    static Singleton *getInstance()
    {
        instance = new Singleton;
        return instance;
    }
    int getNum()
    {
        return this->num;
    }
    void setNum(int num)
    {
        this->num=num;
    }
};
```

This Function returns an instance of a singleton class

This Function returns the current value of a number

This Function sets the value of a num variable

```
};
//initialize the pointer of instance to zero
Singleton *Singleton::instance = 0;
int main()
{
    Singleton *obj = obj->getInstance();
    cout << "\nBefore: " << obj->getNum();
    obj->setNum(111);
    cout << "\nAfter: " << obj->getNum();
    return 0;
}
```

Output

Before: 0

After: 111

5

Unit V

Templates, Namespaces and Exception Handling

Part I : Templates

5.1 : Introduction to Templates

Q.1 Explain the term - 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.

```
int max(int left, int right)
{
    if(left<right)
        return right;
    else
        return left;
}
```

```
double max(double left,double right)
{
    if(left<right)
        return right;
    else
        return left;
}
```

- The above two functions are nearly identical but the function on the left side is for comparing the two integer numbers and the function on the right hand side is for comparing the two double type numbers. Now we want the generic programming algorithm which will perform the functionality of both the data type elements. Hence we can replace int and double by a type T. Here is an example -

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

```
template <class T> //T helps us to take any type of data
T max(T left,T right)
```

```
{
    if(left<right)
        return right;
    else
        return left;
}
int main()
{
    int int1,int2;
    cout<<"\n\t Comparison of Two integer numbers";
    cout<<"\n Enter first integer number";
    cin>>int1;
    cout<<"\n Enter second integer number";
    cin>>int2;
    int max1=max(int1,int2);//passing two integer values
    cout<<" The Maximum integer number is" <<max1<<endl;
    double d1,d2;
    cout<<"\n\t Comparison of Two double numbers";
    cout<<"\n Enter first double number";
    cin>>d1;
    cout<<"\n Enter second double number";
    cin>>d2;
    double max2=max(d1,d2);// passing two double values
    cout<<"The maximum double number is" <<max2<<endl;
    return 0;
}
```

5.2 : Function Template and Class Template

Q.2 Write a C++ program involving a function template.

[SPPU : June-22, Dec.-22, Marks 5]

Ans. : • In function template, the same functional code with different data type elements can be handled.

The syntax of function template is as follows -

```
template <class name_of_data_type>
```

```
name_of_data_type function_name (name_of_data_type id1,...
name_of_data_type id2)
```

For example :

```
template <class T>
T min(T a, T b)
```

Here `template` is a keyword used to represent the template. Then inside the angular brackets keyword `class` is followed by the data type name `T`. The compiler will replace `T` by the appropriate data types.

Function templates are implemented like regular functions, except they are prefixed with the keyword `template`. Here is a sample with a function template.

C++ Program

```
#include <iostream>
using namespace std;
//min returns the minimum of the two elements
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;
}
```

Output

```
min('p', 't') = p
min(10.3, 67.2) = 10.3
```

Q.3 Explain class template and function template with example.

[SPPU : June-22, Dec.-22, Marks 5]

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

The syntax of class template declaration is

```
template < class Type >
class classname
{
    ... //body of class
    ...
};
```

In above example `Type` can be of any data type. Then template class member function is defined. The complete program using class template is as given below.

```
#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, 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();
    return 0;
}

```

Output

```

maximum(100,60) = 100

```

```

maximum('p','t') = t

```

In above program, **Compare** is a class in which two variable a and b are declared for comparison. The function max is used to find the maximum number among the two. The T is used to indicate the data type. If we create an object of type integer by

```

Compare <int> obj1(100,60)

```

then two integer values will be compared. Similarly one can compare two characters, two real values by declaring appropriate objects.

Function template : Refer Q.2.

Q.4 Explain class template using multiple parameters. Write a C++ program.

Ans. : Refer Q.3.

[SPPU : June-22, Marks 7]

Q.5 What is the difference between functional template and class template ?

Ans. : • Function templates are those functions which can handle different data types without separate code for each of them. For a similar operation on several kinds of data types, a programmer need not write different functions.

- Using class template we can write a class whose members take template parameters as types.

5.3 : Function Overloading vs. Function Templates

Q.6 Give the difference between function overloading and function template.

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.

Q.7 Write a function template for finding the minimum value contained in array.

Ans. :

```

#include <iostream.h>
using namespace std;

```

```

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;
}

int main()
{
    int a[10];
    int i;
    for(i=0;i<5;i++)
    {
        cout<<"Enter the integer values "<<endl;
        cin>>a[i];
    }
    cout<<"\n The minimum element of an array is:
        "<<min(a)<<endl;
    float b[10];
    for( i=0;i<5;i++)
    {
        cout<<"Enter the real values "<<endl;
        cin>>b[i];
    }
    cout<<"\n The minimum element of an array is:
        "<<min(b)<<endl;
    return 0;
}

```

Output

```

Enter the integer values
10
Enter the integer values
4

```

```

Enter the integer values
11
Enter the integer values
5
Enter the integer values
12
    The minimum element of an array is: 4
Enter the real values
10.10
Enter the real values
5.5
Enter the real values
11.11
Enter the real values
6.6
Enter the real values
7.7
The minimum element of an array is : 5.5.

```

Part II : Namespaces**5.4 : Introduction to Namespaces****Q.8** Explain the concept of namespaces.

[SPPU : June-22, Dec.-22, Marks 5]

Ans. : • Namespaces are used to group the entities like class, variables, objects, function under a name.

• The namespaces help to divide global scope into sub-scopes, where each sub-scope has its own name.

• **Example :**

```

#include <iostream>
using namespace std;

```

```

namespace ns1
{
    int a = 5;
}

namespace ns2
{
    char a[] = "Hello";
}

int main ()
{
    cout << ns1::a << endl;
    cout << ns2::a << endl;
    return 0;
}

```

Output

5
Hello

Q.9 What is std namespace ?

Ans. : • All the files in the C++ standard library declare all of its entities within the **std** namespace. That is why in the C++ program the **std namespace** is declared as follows -

using namespace std;

- This statement is used in the program which uses any entity declared in **iostream**.

5.5 : Rules of Namespaces**Q.10 Enlist the rules of namespaces.**

Ans. : • Following are the rules of namespaces -

- 1) The namespace must be defined using the keyword **namespace**.

- 2) The namespace definition must appear at the global scope, or it must be present inside another namespace (i.e. nested)
- 3) The definition of namespace must not be terminated with semicolon.
- 4) It is not allowed to create an instance of namespace.
- 5) There can be unnamed namespace as well.
- 6) The namespace definition is valid over multiple files. There is no need to redefine it across multiple files.
- 7) The namespace declaration do not have access specifiers (public, private)

Part III : Exception Handling**5.6 : Basics of Exception Handling****Q.11 What is exception handling ?**

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

- The purpose of exception handling mechanism is to detect and handle the exceptional situations so that appropriate action can be taken.

Q.12 Give 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.

5.7 : Exception Handling Mechanism

Q.13 Explain the exception handling mechanism.

Ans. : • 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**.
- Following figures illustrates the concept of exception handling mechanisms -

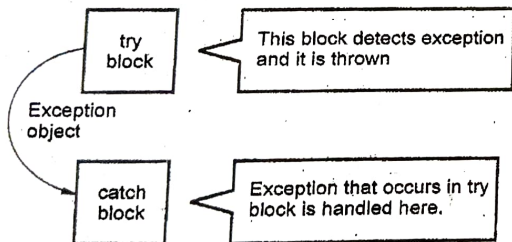


Fig. Q.13.1 (a) try-catch mechanism

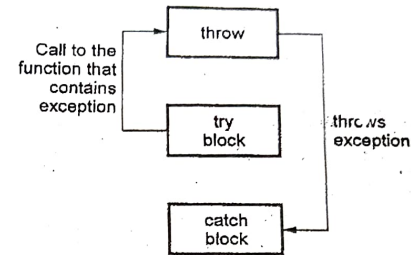


Fig. Q.13.1 (b) throw-try-catch mechanism

- Mainly exception handling is done with the help of three keywords : *try*, *catch* and *throw*. The *try* and *catch* block is as shown below -

```
try
{
    throw exception;
    // exception is some value
    // the portion of the code that is to be monitored for error
    // detection
}
catch( argument)
{
    //catch block softly handles the exception
}
```

- The *try* block contains the portion of the program that is to be examined for error detection. If an exception (i.e. error) occurs in this block then it is thrown using *throw*. Using *catch* the exception is caught and processed. If *try* block contains all the code included in main then effectively the complete program must be scanned for errors. Any exception is caught by the catch block. This catch block should be immediately followed by the *try* block.

5.8 : Throwing and Catching Mechanism

Q.14 Explain exception handling mechanism in C++. Write a C++ to handle divide by zero exception. [SPPU : June-22, Marks 7]

Ans. : Exception handling mechanism : Refer Q.13.

C++ Program for handling divide by zero exception :

- The purpose of exception handling is to handle abnormal events. The following program illustrates that in division operation the denominator value should not be zero and if at all it is zero then how to handle such error.

```

/*****
Program to handle divide by zero error using exception handling
mechanism
*****/
#include <iostream>
using namespace std;

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
Cant divide by zero.

5.9 : Specifying Exceptions

Q.15 Write short note on - Exception specification.

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

void f() throw(int,double)

Here, 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 double type handler.

An exception specification with an empty throw, as in

void f() throw()

tells the compiler that the function does not throw any exceptions.

For example -

```

/*****
Program for demonstrating exception specification
*****/

```

```

#include <iostream.h>
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;
    }
}

```

Output

Handling the double value 12.34

5.10 : Multiple Exceptions

Q.16 Explain how to use multiple catch statements ? Give C++ program.

Ans. : • The exception handler is declared by the keyword *catch* which is responsible to handle the exception thrown by the *try* block. Normally the code within the catch statement attempts to rectify the errors by taking appropriate action. 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;
    return 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

5.11 : Exceptions with Arguments**Q.17** How to use exceptions with arguments ? Explain.

[SPPU : Dec.-22, Marks 6]

Ans. : • The exception with argument helps the programmer to know the cause of exception. Following program illustrates this concept -

Example

```
#include <iostream>
#include <string.h>
using namespace std;
class Person
{
private:
    double age;
public:
    char description[20];
    Person()
    {
        age=0;
        description[0]='\0';
    }
    Person(double val, char *desc)
    {
        age=val;
        strcpy(description,desc);
    }
    void input_data()
    {
        cout << "\n Enter age: ";
        cin >> age;
        if(age<0)
            throw Person(age,"Negative value of Age!");
        else
            cout << "You have entered the age as: " << age;
    }
};

int main()
{
    Person p;
    try
```

Passing
Arguments
to Exception

```

{
    p.input_data();
}
catch(Person obj)
{
    cout << "\n Exception occurred!!!";
    cout << "\n Error Description: " << obj.description;
}
return 0;
}

```

Output

Enter age: -35

Exception occurred!!!

Error Description: Negative value of Age!

5.12 : C++ Streams

Q.18 What is stream ? Explain the types of streams available in C++.

[SPPU : June-22, Dec.-22, Marks 5]

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. For example, at the start of a program, the standard input stream "cin" is connected to the keyboard and the standard output stream "cout" is connected to the screen. In fact, input and output streams such as "cin" and "cout" are examples of stream objects. In C++ the entire I/O (Input/Output) system operates through streams. A stream is connected to a physical device by the C++ input output system (Popularly known as I/O system).

- When C program begins execution three predefined streams are automatically opened and those are stdin, stdout, stderr and when C++ program begins four streams get opened - cin, cout, cerr and clog.

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.

5.13 : Stream Classes

Q.19 Write a short note on - Stream classes.

[SPPU : Dec.-22, Marks 5]

Ans. : • The stream can represent file, console, block of memory or hardware device. The *iostream* library provides the common set of functions for handling these streams. The general representation is as shown in following Fig. Q.19.1.

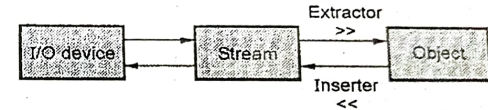


Fig. Q.19.1 Stream

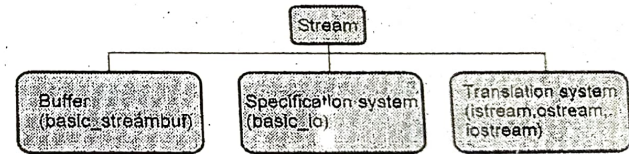


Fig. Q.19.2 Basic level of stream class

- 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. This system converts the objects to a sequence of characters. (Refer Fig. Q.19.2)
- This can be presented by following Fig. Q.19.3 of class hierarchy.

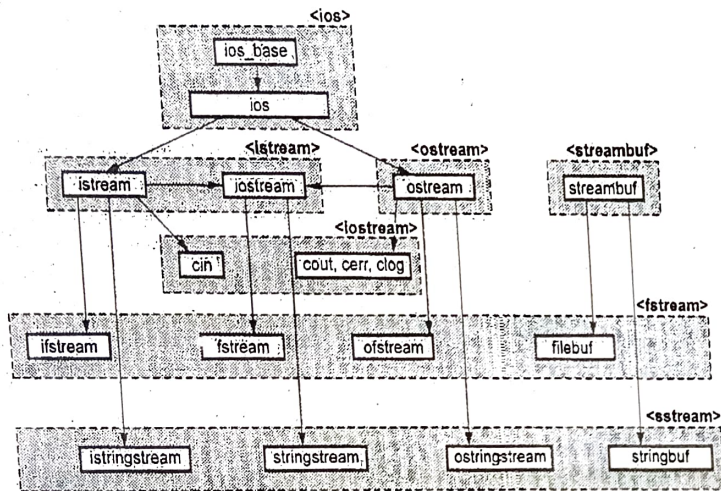


Fig. Q.19.3 Stream class hierarchy

5.14 : Unformatted I/O

Q.20 Write a C++ program for demonstrating getline and write function.

Ans. : The `getline` function is used to read the file line by line and the `write` statement is used to write the contents either to the file or to the console. Following C++ program reads the content from the file and displays each line. Finally the total number of lines count is also displayed.

```
#include <iostream>
#include <fstream>
#include <string>
```

```
using namespace std;
int main()
{
    int count=0;
    char data[20];
    ifstream input;
    cout << "Reading the contents from the file" << endl;
    input.open("Sample.dat");
    input.getline(data,20,'\n');
    int n = strlen(data);
    while(input)
    {
        cout.write(data,n);
        cout << endl;
        count++;
        input.getline(data,20);
        n = strlen(data);
    }
    cout << "\n Number of lines in the file = " << count << endl;
    return 0;
}
```

Output

Reading the contents from the file

Statement 1

Statement 2

Statement 3

Number of lines in the file = 3

5.15 : Formatted I/O and I/O Manipulators

Q.21 Enlist and explain formatted I/O with example.

Ans. : The `ios` class declares the values that are used to set or clear the format flags -

Following are the format flags along with their meaning.

Flags	Meaning
skipws	If this flag is set then leading white spaces are discarded (skipped). On clearing this flag the leading white spaces are not discarded.
left	On setting this flag the output becomes left justified.
right	On setting this flag the output becomes right justified.
internal	If it is set then numeric value is padded to fill the field between any sign or base character and number.
oct	It is set means numeric values are outputted in octal.
dec	It is set means numeric values are outputted in decimal.
hex	It is set means numeric values are outputted in hex.
showbase	If set then it shows base indicator(for example 0X for hex).
showpos	It shows + sign before positive numbers.
showpoint	On set it shows decimal point and trailing zeros for all floating point numbers.
scientific	On set it shows exponential format on floating point numbers.
fixed	On set it shows fixed format on floating point numbers.
boolalpha	Booleans can be set using <i>true</i> or <i>false</i> .
uppercase	On setting this flag upper case A-F are used for hex values and E for scientific values.

- The **oct**, **dec** and **hex** fields are collectively referred as **basefield**. The **left**, **right** and **internal** fields can be referred as **adjustfield**, similarly **scientific** and **fixed** can be referred as **floatfield**.
- The **setf()** function is used to set the flag and **unsetf()** function is used to clear the flag.

For example, if we want the text to be displayed on the console should be right justified then -

```
cout.setf(ios::right);
cout<<"Twinkle Twinkle Little Stars";
```

- Following is a program that shows how several format flags can be used.

```
#include <iostream>
using namespace std;
int main()
{
    cout << 100.75 << " hello India " << 100 << "\n";
    cout << 10 << " " << -10 << "\n";
    cout << 100.0 << "\n\n";
    cout.unsetf(ios::dec);

    cout.setf(ios::hex|ios::scientific);
    cout << 100.75 << " hello India " << 100 << "\n";
    cout.setf(ios::showpos);
    cout << 10 << " " << -10 << "\n";
    cout.setf(ios::showpoint | ios::fixed);
    cout << 100.0;
    return 0;
}
```

Output

```
100.75 hello India 100
10 -10
100

1.0075e+02 hello India 64
a fffffff6
+100.000000
```

Q.22 What are I/O manipulators ? Write a C++ programs to demonstrate it. [SPPU : Dec.-22, Marks 5]

Ans. : • Manipulators are the operators in C++ that are used for formatting the output. Various commonly used manipulators are **setw**, **endl** and **setfill**.

- The `setw` is used to set the minimum field width. We can set the right justified numbers using `setw` operator. Note that the `setw` function is defined in `<iomanip.h>` file.

C++ Program

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int a=4444,b=22,c=333;
    cout << "Using setw(30)\n";
    cout << setw(30) << a << "\n";
    cout << setw(30) << b << "\n";
    cout << setw(30) << c << "\n";
    cout << "Using setw(20)\n";
    cout << setw(20) << a << "\n";
    cout << setw(20) << b << "\n";
    cout << setw(20) << c << "\n";
    cout << "Using setw(10)\n";
    cout << setw(10) << a << "\n";
    cout << setw(10) << b << "\n";
    cout << setw(10) << c << "\n";
    return 0;
}
```

Output

```
Using setw(30)
                4444
                22
                333

Using setw(20)
                4444
                22
                333
```

```
Using setw(10)
        4444
        22
        333
```

The functionality of `endl` is similar to `"\n"` i.e. newline character.

C++ Program

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    cout << "This is first line" << endl;
    cout << "This is second line" << endl;
    cout << "This is third line" << endl;
    cout << "This is forth line" << endl;
    return 0;
}
```

Output

```
This is first line
This is second line
This is third line
This is forth line
```

The `setfill` manipulator is used to fill the fields entirely using some character specified within it.

C++ Program

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int a=4444,b=22,c=333;
    cout << setw(10) << setfill('*') << a << endl;
```

```

cout << setw(10) << setfill('*') << b << endl;
cout << setw(10) << setfill('*') << c << endl;
return 0;
}

```

Output

```

*****4444
*****22
*****333

```

END... ✍

Unit VI

6

Working with Files

6.1 : Classes for File Stream Operations

Q.1 Name and explain the classes used for file stream operations.

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`.

6.2 : Opening and Closing Files

Q.2 Write and explain the file open function.

Ans. : • The file operations are associated with the object of one of the classes : `ifstream`, `ofstream` or `fstream`. Hence we need to create an object of the corresponding class.

- The file can be opened by the function called `open()`. The syntax of file open is
`Open(filename, mode)`
- The `filename` is a null terminated string that represents the name of the file that is to be opened. The mode is optional parameter and is used with the flags as given below.

<code>ios::in</code>	Open for input operation.
<code>ios::out</code>	Open for output operation.
<code>ios::binary</code>	Open for binary operations.
<code>ios::ate</code>	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.
<code>ios::app</code>	The output operations are appended to the file. This is an appending mode. That means contents are inserted at the end of the file.
<code>ios::trunc</code>	The contents of pre existing file get destroyed and it is replaced by new one.

- We can use `open()` function using the above given syntax as -
`ofstream obj;`
`obj.open("sample.bin", ios::out | ios::binary)`
- That means the file `sample.bin` is opened for output operation in binary mode. Thus we can combine the flags using OR operator (`|`).
- The `is_open()` is a boolean function that can be used to check whether the file is open or not.

For example :

```
if(obj.is_open())
{
cout << "File is Successfully opened for operations";
}
```

Q.3 What is file mode ? Explain any four modes supported by C++.
 [SPPU : June-22, Dec.-22, Marks 10]

Ans. : Refer Q.2

Q.4 What is the difference between opening a file with constructor function and `open()` function. [SPPU : June-22, Dec.-22, Marks 10]

Ans. : 1. Opening a file using constructor function

- Using the constructor the object is created. In the same way using the file stream constructor the **file stream object** is created. Suppose we want to open the file for reading then we make use of `ifstream` class and create an object. Using this object the desired file is opened and is available for reading data. For example
`ifstream myfile("input.txt", ios::in);`
- The above statement creates `myfile` object of input stream class `ifstream`.
- Opening a file using Constructor function is better to be used when a single file is to be opened using single stream.

2. Opening a file using `open` function

- The `open` function of single stream is used to open the desired file. For example
`ifstream myfile;` //This line creates an object of `ifstream`
`myfile.open("input1.txt", ios::in);`
`myfile.close();`
`myfile.open("input2.txt", ios::in);`
`myfile.close();`
`myfile.open("input3.txt", ios::in);`
`myfile.close();`
- When we want to open multiple files one after the other then use `open` function.

Q.5 Explain file functions for text file and binary file operations.

[SPPU : June-22, Dec.-22, Marks 10]

Ans. : 1) Open file function : Refer Q.2.

2) Close file function : To close the file the member function `close()` is used. The `close` function takes no parameter and returns no value.

For example

```
obj.close ();
```

3) **Read and write functions** : We can make use of two functions namely : **read** and **write** for handling the binary file format.

The syntax of read and write functions will be -

```
input_obj.read((char *) &variable, sizeof(variable));
```

memory block
size of block

```
output_obj.write ((char*) & variable, size of (variable));
```

- The memory block must be type cast to pointer to character type This block acts as a buffer in which read contents can stored or the data to be written in the file can be stored.

6.3 : Detecting End_Of_File (EOF)

Q.6 Explain how to detect end of the file ?

Ans. : • For finding end of the file we use `eof()` function. This function is a member function of `ios` class. If end of file is encountered then it returns a non-zero value.

For example

```
if (seqfile.eof () != 0)
```

```
{
    cout << " You are at the end of the file";
}
```

Q.7 Write a program using the `open ()`, `eof ()` and `getline ()` functions to open and read file contents line by line.

Ans. :

```
#include <iostream>
```

```
#include <fstream>
```

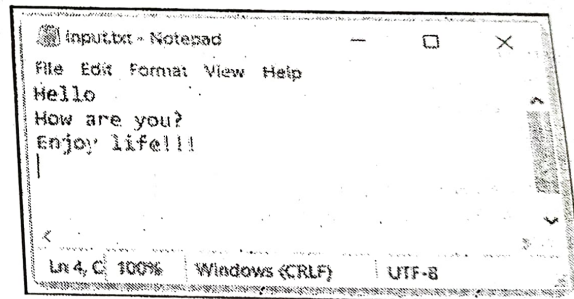
```
using namespace std;
```

```
int main(void)
```

```
{
    char filename[70];
```

```
ifstream ifs;
char FirstLine[70];
char SecondLine[70];
char ThirdLine[70];
cout << "Enter the filename to be opened : ";
cin >> filename;
ifs.open(filename);
if(!ifs.eof())
{
    cout << "\nThe first line... \n";
    ifs.getline(FirstLine, 70);
    cout << FirstLine << "\n";
    cout << "The second line... \n";
    ifs.getline(SecondLine, 70);
    cout << SecondLine << endl;
    cout << "The third line... \n";
    ifs.getline(ThirdLine, 70);
    cout << ThirdLine << endl;
}
}
```

Sample Input file(Input.txt)



Output

```

E:\atost\exe
Enter the filename to be opened: input.txt
The first line...
Hello
The second line...
How are you?
The third line...
Enjoy life!!!

Process returned 0 (0x0)   execution time : 3.912 s
Press any key to continue.

```

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;
}

```

Output

The Contents of the file are...

Statement 1

Statement 2

Statement 3

Statement 4

Statement 5

Statement 6

Statement 7

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;
return 0;
}

```

Q.10 Write a program that reads an array of number from file and creates another two files store the odd number in one file and even numbers in another file.

Ans. :

```

#include<iostream>
#include<fstream>
using namespace std;
int main()
{
    ofstream out_obj;
    int a[10]={1,2,3,4,5,6,7,8,9,10};
    int b[10];
    out_obj.open("input.dat");
    out_obj.write((char *)&a,sizeof(a));
    out_obj.close();
    ifstream in_obj;

```

```

ofstream fp1,fp2;
in_obj.open("input.dat");
in_obj.read((char *)&b,sizeof(b)); //storing the values file in b[]
fp1.open("Even.dat");
fp2.open("Odd.dat");
for(int i=0;i<10;i++)
{
    if((b[i]%2)==0)
        fp1<<b[i]<<" "; //writing to even file
    else
        fp2<<b[i]<<" "; //writing to odd file
}
in_obj.close();
fp1.close();
fp2.close();
ifstream fp;
char ch;
fp.open("Even.dat");
cout<<"\n The contents of even file are ..."<<endl;
while(fp) //reading even file
{
    fp.get(ch);
    cout<<ch;
}
fp.close();
fp.open("Odd.dat");
cout<<"\n The contents of odd file are ..."<<endl;
while(fp)//reading odd file
{
    fp.get(ch);
    cout<<ch;
}
fp.close();
return 0;
}

```

Output

The contents of even file are ...

2 4 6 8 10

The contents of odd file are ...

1 3 5 7 9

Q.11 Write a program which copies the content of one file to a new file by removing unnecessary spaces between words.

Ans. :

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

```
int main()
{
    ifstream infile;
    ofstream outfile;
    infile.open("Sample.dat");
    outfile.open("Myoutput.dat");
    char output;
    if(infile == NULL)
    {
        cout << "Error" << endl;
        exit(0);
    }
    while(infile)
    {
        infile.get(output);
        if(output != ' ')
        {
            outfile << output;
        }
    }
}
```

```
infile.close();
outfile.close();
ifstream fp;
fp.open("Myoutput.dat");
cout << "\n Following are the contents of Myoutput.dat file...\n";
while(fp)
{
    fp.get(output);
    cout.put(output);
}
fp.close();
}
```

<Sample.dat>

Statement 1 Statement 2 Statement 3
return 0;

Output

Following are the contents of Myoutput.dat file...
Statement1Statement2Statement3

Q.12 Using file handling methods of C++ write a program and explain how to merge the contents of two files into one file.

Ans. : Following is a simple C++ program in which we have created two files namely - emp.dat and dept.dat. The contents of both the files are read and merged into the third file named Combined.dat.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    //Reading from the files
    char Data[80];
    ifstream in_obj;
    in_obj.open("emp.dat");
```

```

ofstream out_obj;
out_obj.open("Combined.dat");
while(in_obj)
{
    in_obj.getline(Data,80);
    out_obj<<Data;
    out_obj<<"\n";
}
cout<<"\n The contents of emp.dat file are written...\n";
in_obj.close();
in_obj.open("dept.dat");
cout<<"\n The contents of dept.dat file are written...\n";
while(in_obj)
{
    in_obj.getline(Data,80);
    out_obj<<Data;
    out_obj<<"\n";
}
in_obj.close();
out_obj.close();
ifstream fp;
fp.open("Combined.dat");
cout<<"\n Following are the contents of Combined.dat file...\n";
while(fp)
{
    fp.getline(Data,80);
    cout<<"\n"<<Data;
}
fp.close();
return 0;
}

```

Program explanation :

In above program, initially the **emp.dat** file is opened in a read mode and the **Combined.dat** is opened in writing mode. The contents are read from the **emp.dat** and written in the **Combined.dat**. Similarly, the **dept.dat** file

is opened in read mode and written in **Combined.dat**. All these files are then closed. Finally only **Combined.dat** file is opened in read mode and the merged contents are displayed on the console.

6.4 : File Pointers and Manipulators

Q.13 Explain the seek and tell operations of file handling in detail.

[SPPU : Dec.-22, Marks 6]

Ans. : • The file pointers are 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 positions.

While performing file operations, we must be able to reach at any desired position inside the file. For this purpose there are two commonly used functions -

1) seek : The seek operation is using two functions **seekg** and **seekp**.

- **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

2) 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).

```

## 6.5 : Updating File

Q.14 : Write a C++ program for performing file handling operations on Employee database.

Ans. :

Program for performing various operations on Sequential File organization.

```

.....
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
class EMP_CLASS
{
typedef struct EMPLOYEE
{
 char name[10];
 int emp_id;
 int salary;
}Rec;
Rec Records;
public:
 void Create();
 void Display();
 void Update();
 void Delete();
 void Append();
 int Search();
};
void EMP_CLASS::Create()
{
 char ch='y';

```

This record structure is for the sequential file  
For example

| name | emp_id | salary |
|------|--------|--------|
| AAA  | 10     | 1000   |
| BBB  | 20     | 2000   |
| CCC  | 30     | 3000   |

```

fstream seqfile;
seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
do
{
 cout << "\n Enter Name: ";
 cin >> Records.name;
 cout << "\n Enter Emp_ID: ";
 cin >> Records.emp_id;
 cout << "\n Enter Salary: ";
 cin >> Records.salary;
 //then write the record containing this data in the file
 seqfile.write((char*)&Records,sizeof(Records));
 cout << "\nDo you want to add more records?";
 cin >> ch;
}while(ch=='y');
seqfile.close();
}
void EMP_CLASS::Display()
{
 fstream seqfile;
 int r,m,i;
 seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
 //positioning the pointer in the file at the beginning
 seqfile.seekg(0,ios::beg);
 cout << "\n The Contents of file are ..." << endl;
 //read the records sequentially
 while(seqfile.read((char *)&Records,sizeof(Records)))
 {
 if(Records.emp_id!=-1)
 {
 cout << "\nName: " << Records.name;
 cout << "\nEmp_ID: " << Records.emp_id;
 cout << "\nSalary: " << Records.salary;
 cout << "\n";
 }
 }
}

```

User must enter the desired data in the members of the structure Records

```

}
int last_rec=seqfile.tellg();//last record position
//formula for computing total number of objects in the file
n=last_rec/(sizeof(Rec));
cout<<"\n\n Total number of objects are "<<n<<"(considering
logical deletion)";
seqfile.close();
}

```

```
void EMP_CLASS::Update()
```

```

{
 int pos;
 cout<<"\n For updation,";
 fstream seqfile;
 seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
 seqfile.seekg(0,ios::beg);
//obtaining the position of desired record in the file
 pos=Search();
 if(pos==-1)
 {
 cout<<"\n The record is not present in the file";
 return;
 }
}

```

```
//calculate the actual offset of the desired record in the file
```

```

int offset=pos*sizeof(Rec);
seqfile.seekp(offset);//seeking the desired record for modification
cout<<"\n Enter the values for updation...";
cout<<"\n Name: ";cin>>Records.name;
cout<<"\n Emp_Id: ";cin>>Records.emp_id;
cout<<"\n Salary: ";cin>>Records.salary;
seqfile.write((char*)&Records,sizeof(Records))<<flush;
seqfile.seekg(0);
seqfile.close();
cout<<"\n The record is updated!!!";
}

```

New values for the  
updation of record

```
void EMP_CLASS::Delete()
```

```

{
 int id,pos;
 cout<<"\n For deletion,";
 fstream seqfile;
 seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
 seqfile.seekg(0,ios::beg);//seeking for reading purpose
 pos=Search();//finding pos. for the record to be deleted
 if(pos!=-1)
 {
 cout<<"\n The record is not present in the file";
 return;
 }
//calculate offset to locate the desired record in the file
 int offset=pos*sizeof(Rec);
 seqfile.seekp(offset);//seeking the desired record for deletion
 strcpy(Records.name,"");
 Records.emp_id=-1;
 Records.salary=-1;
 seqfile.write((char*)&Records,sizeof(Records))<<flush;
 seqfile.seekg(0);
 seqfile.close();
 cout<<"\n The record is Deleted!!!";
}

```

It's a logical deletion

```
void EMP_CLASS::Append()
```

```

{
 fstream seqfile;
 seqfile.open("EMP.DAT",ios::ate|ios::in|ios::out|ios::binary);
 seqfile.seekg(0,ios::beg);
 int i=0;
 while(seqfile.read((char*)&Records,sizeof(Records)))
 {
 i++;//going through all the records
 // for reaching at the end of the file
 }
 //instead of above while loop
}

```



```

//we can also use seqfile.seekg(0,ios::end)
//for reaching at the end of the file
seqfile.clear();//turning off EOF flag
cout<<"\n Enter the record for appending";
cout<<"\nName: ";cin>>Records.name;
cout<<"\nEmp_ID: ";cin>>Records.emp_id;
cout<<"\nSalary: ";cin>>Records.salary;
seqfile.write((char*)&Records,sizeof(Records));
seqfile.seekg(0);//reposition to start(optional)
seqfile.close();
cout<<"\n The record is Appended!!!";
}

int EMP_CLASS::Search()
{
 ifstream seqfile;
 int id,pos;
 cout<<"\n Enter the Emp_ID for searching the record ";
 cin>>id;
 seqfile.open("EMP.DAT",ios::ate|ios::in|ios::out|ios::binary);
 seqfile.seekg(0,ios::beg);
 pos=-1;
 int i=0;
 while(seqfile.read((char *)&Records,sizeof(Records)))
 {
 if(id==Records.emp_id)
 {
 pos=i;
 break;
 }
 i++;
 }
 return pos;
}

void main()
{

```

```

EMP_CLASS List;
char ans='y';
int choice,key;
clrscr();
do
{
 cout<<"\n Main Menu "<<endl;
 cout<<"\n 1.Create";
 cout<<"\n 2.Display";
 cout<<"\n 3.Update";
 cout<<"\n 4.Delete";
 cout<<"\n 5.Append";
 cout<<"\n 6.Search";
 cout<<"\n 7.Exit";
 cout<<"\n Enter your choice ";
 cin>>choice;
 switch(choice)
 {
 case 1:List.Create();
 break;
 case 2:List.Display();
 break;
 case 3:List.Update();
 break;
 case 4:List.Delete();
 break;
 case 5:List.Append();
 break;
 case 6:key=List.Search();
 if(key<0)
 else
 cout<<"\n Record is not present in the file";
 cout<<"\n Record is present in the file";
 break;

```

```

case 7:exit(0);
}
cout<< "\n\t Do you want to go back to Main Menu?";
cin>>ans;
}while(ans=='y');
}

```

## 6.6 : Error Handling During File Operations

Q.15 Explain error handling during file operations.

[SPPU : June-22, Dec.-22, Marks 5]

Ans. : • When error occurs in file handling, flags are set in the state according to the general category of the error. Flags and their error categories are summarized in the following table.

| state flag   | Purpose                                                                                                       |
|--------------|---------------------------------------------------------------------------------------------------------------|
| ios::goodbit | This state flag indicates that there is no error with streams. In this case the status variables has value 0. |
| ios::badbit  | This state flag indicates that the stream is corrupted and no read/write operation can be performed.          |
| ios::failbit | This state flag indicates that input/output operation failed.                                                 |
| ios::eofbit  | This state flag indicates that the input operation reached end of input sequence.                             |

- When error occurs while performing file I/O operations, the appropriate message is displayed and then the file terminates.
- Following program illustrates how to use appropriate error messages on corresponding read and write error-causing situations.

```

#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 (los)
 {
 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 (los)
 {
 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 (lis)
 {
 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
 // another array 'array2'

 if (lis)
 {
 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;


```

**Output**

Writing the contents to the file...

Reading the contents from the file ...

10 20 30 40 50 0 0 0 0 0

END... 

**JUNE - 2022 [5869]-249**

**Solved Paper**

Course 2019

Time :  $2\frac{1}{2}$  Hours]

[Maximum Marks : 70

Instructions to the candidates :

- 1) Answer Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8
- 2) Neat diagrams must be drawn wherever necessary.
- 3) Figures to the right side indicate full marks.
- 4) Use of calculator is allowed.
- 5) Assume suitable data, if necessary.

Q.1 a) What is operator overloading ? Why it is necessary to overload an operator ? (Refer Q.1 of Chapter - 3)

b) Write a program in C++ to use scope resolution operator. (Refer Q.23 of Chapter - 1)

c) What is friend function ? What are the merits and demerits using the friend function ? (Refer Q.18 of Chapter - 2)

OR

Q.2 a) What are the rules for over loading operators. (Refer Q.3 of Chapter - 3)

b) Which operators can not be overloaded ? Write steps overload + operator so that it can add two complex numbers. (Refer Q.10 of Chapter - 3)

c) Write down program to overload unary operators ? (Any operators). (Refer Q.8 of Chapter - 3)

Q.3 a) Explain virtual base class and virtual function with example. (Refer Q.15 of Chapter - 4)

b) What does inheritance mean in C++ ? What are different forms of inheritance ? Give example of each.  
(Refer Q.1 and Q.4 of Chapter - 4) [6]

c) Discuss the ways in which inheritance promotes software reuse, saves time during program development and helps prevent errors.  
(Refer Q.10 of Chapter - 4) [6]

OR

Q.4 a) What is the ambiguity that arises in multiple inheritance ? How it can be overcome. Explain with example.  
(Refer Q.14 of Chapter - 4) [6]

b) What are types of inheritance ? Explain in detail.  
(Refer Q.8 of Chapter - 4) [6]

c) Discuss the role of access specifiers in inheritance and show their visibility when they are inherited as public, private protected.  
(Refer Q.12 of Chapter - 4) [6]

Q.5 a) Write a C++ program involving a function template.  
(Refer Q.2 of Chapter - 5) [5]

b) Explain exception handling mechanism in C++ ? Write a program in C++ to handle divide by zero exception.  
(Refer Q.14 of Chapter - 5) [7]

c) Explain class template and function template with example.  
(Refer Q.3 of Chapter - 5) [5]

OR

Q.6 a) Explain class template using multiple parameters ? Write a program in C++. (Refer Q.4 of Chapter - 5) [7]

b) Explain namespace with example.  
(Refer Q.8 of Chapter - 5) [5]

c) What is stream ? Explain types of streams available in C++.  
(Refer Q.18 of Chapter - 5) [5]

Q.7 a) What is file mode ? Explain any four modes supported by C++.  
(Refer Q.3 of Chapter - 6) [6]

b) Explain error handling during file operations.  
(Refer Q.15 of Chapter - 6) [5]

c) What is the difference between opening a file with construction function and open ( ) function. (Refer Q.4 of Chapter - 6) [6]

OR

Q.8 a) Write a program using the open ( ), eof ( ) and getline ( ) functions to open and read file contents line by line.  
(Refer Q.7 of Chapter - 6) [6]

b) Explain file functions for text file and binary file operations.  
(Refer Q.5 of Chapter - 6) [7]

c) Explain file opening modes in detail.  
(Refer Q.2 of Chapter - 6) [4]

DECEMBER - 2022 [5925]-219

Solved Paper

Course 2019

Time : 2  $\frac{1}{2}$  Hours]

[Maximum Marks : 70

Q.1 a) What are the rules for overloading operators ?  
(Refer Q.3 of Chapter - 3) [4]

b) Write down a C++ program to implement operator overloading for complex class. (Refer Q.10 of Chapter - 3) [8]

c) Explain friend function with example.  
(Refer Q.13 of Chapter - 3) [6]

OR

Q.2 a) What is operator overloading ? Write a program to overload unary operator. (Refer Q.1 and Q.8 of Chapter - 3) [8]

b) Write down a C++ program for copy constructor for string class. [6]

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

class String {
private:
 char* s;

public:
 String(const char* str = NULL); // constructor
 ~String() { delete[] s; } // destructor
 void print() { cout << s << endl; }
};

String::String(const char* str)
{
 int size = strlen(str);
 s = new char[size + 1];
 strcpy(s, str);
}

int main()
{
 String str1("India");
 String str2 = str1;
 cout << "\n Original String: ";
 str1.print();
 cout << "\n Printing String using Copy Constructor: ";
 str2.print();
 return 0;
}
```

c) Differentiate friend function with normal function of the class.

Ans. :

- Friend function helps to access private and protected data while normal function is a group of statements that performs a specific task.
- The "friend" keyword is used to define the friend function whereas normal functions do not use "friend" keyword

Q.3 a) Explain containment and inheritance along with examples. (Refer Q.26 and Q.5 of Chapter - 4)

b) What is virtual function? Explain how to achieved run time polymorphism. (Refer Q.18 of Chapter - 4)

c) Explain function over loading and function overriding in detail. (Refer Q.12 of Chapter - 1 and Q.18 of Chapter - 4)

OR

Q.4 a) What does inheritance mean in C++? Give an example of each. (Refer Q.1 and Q.5 of Chapter - 4)

b) What is polymorphism? Explain with example to achieved run time polymorphism. (Refer Q.16 and Q.18 of Chapter - 4)

c) Write copy constructor for employee class, in which objects of string class and date class are the data members.

Ans. :

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

```
class Date
{
private:
 int day, month, year;
```

```

public:
 // constructor
 Date(){}
 Date(int d, int m, int y)
 {
 day = d;
 month = m;
 year = y;
 }
 void print_date()
 {
 cout<<"Date: ";
 cout<<day<<"."<<month<<"."<<year;
 }

 // copy constructor
 Date(const Date& new_date)
 {
 day = new_date.day;
 month=new_date.month;
 year=new_date.year;
 }
};

class Employee
{
private:
 string name;
 Date birth_date;

public:
 // constructor

```

```

Employee(string n, Date bd){
 name = n;
 birth_date = bd;
}

void print_emp_info()
{
 cout<<"Employee Name: "<<name<<endl;
 birth_date.print_date();
}

// copy constructor
Employee(const Employee& other)
{
 name=other.name;
 birth_date=other.birth_date;
}
};

int main()
{
 Date dob(1, 1, 2023);
 Employee emp1("AAA ZZZ", dob);
 cout<<"\n\n ***** Before Copy Constructor *****"<<endl;
 emp1.print_emp_info();
 Employee emp2 = emp1; // calling copy constructor
 cout<<"\n\n ***** After Copy Constructor *****"<<endl;
 emp2.print_emp_info();
 return 0;
}

```

Output

\*\*\*\*\* Before Copy Constructor \*\*\*\*\*

Employee Name: AAA ZZZ

Date: 1-1-2023

\*\*\*\*\* After Copy Constructor \*\*\*\*\*

Employee Name: AAA ZZZ

Date: 1-1-2023

Q.5 a) What is a user defined exception ? Write down the scenario where we require use define exceptions.

(Refer Q.17 of Chapter - 5)

[6]

b) What is namespace ? To demonstrate namespace with example.

(Refer Q.8 of Chapter - 5)

[6]

c) Explain class template and function template with example.

(Refer Q.3 and Q.2 of Chapter - 5)

[5]

OR

Q.6 a) What is stream ? Explain types of streams available in C++.

(Refer Q.18 of Chapter - 5)

[6]

b) Explain namespace in C++ with example ?

(Refer Q.8 of Chapter - 5)

[6]

c) Compare late binding and early binding.

(Refer Q.25 of Chapter - 4)

[5]

Q.7 a) Explain error handling during file operations.

(Refer Q.15 of Chapter - 6)

[6]

b) Write a program using put ( ) to write characters to a file until user enters a dollar sign.

[6]

Ans. :

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
char c;
```

```
ofstream myfile("output.txt"); // Open output file for writing
```

```
if (myfile.is_open())
```

```
{
```

```
cout << "Enter some text, press $ to quit:\n";
```

```
while (cin.get(c) && c != '$')
```

```
;
```

```
myfile.put(c); // Write character to file
```

```
}
```

```
myfile.close(); // Close output file
```

```
cout << "Text written to output.txt\n";
```

```
}
```

```
return 0;
```

```
}
```

c) Write a note on file operating modes.

(Refer Q.3 of Chapter - 6)

[5]

OR

Q.8 a) Explain manipulators for file handling in C++.

(Refer Q.22 of Chapter - 5)

[5]

b) What is file pointer ? Write a note on file opening and file closing. (Refer Q.13, Q.4 and Q.5 of Chapter - 6)

[6]

c) Explain stream classes hierarchy for file handling in C++.

(Refer Q.19 of Chapter - 5)

[6]

END...✍