

JCEI's JAIHIND COLLEGE ENGINEERING, KURAN



Estd. 1996

“PRACTICAL LAB MANUAL”

UNDER THE SUBJECT

OBJECT ORIENTED PROGRAMMING (210247)

SE-AI & DS ENGINEERING

DEPARTMENT OF AI & DS ENGINEERING

SEMESTER-III

Name of Faculty: Prof M. G. Sinalkar

TEACHING SCHEME

Practical: 2Hrs/Week

Lectures: 4Hrs/Week

EXAMINATION SCHEME

Practical Assessment: 25 Marks

Term Work: 25 Marks

JCEI's JAIHIND COLLEGE OF ENGINEERING, KURAN



Estd. 1996

DEPARTMENT OF AI & DS ENGINEERING

CERTIFICATE

This is to certify that the Object Oriented
Programming Lab Manual under the subject

“OBJECT ORIENTED PROGRAMMING (210247)”

SUBMITTED BY

Mr./Miss.

Exam Seat No.

Is a bonafide work carried out by student under the supervision of **Prof. M. G. Sinalkar** and it is submitted towards the partial fulfillment of the requirement of Second Year of AI & DS Engineering.

Prof. M. G. Sinalkar

Subject Teacher

Dept.of Computer And AI&DS Engineering.

Prof. A. A. Khatri

Head of Department

Dept. of Computer AI&DS Engineering.

Dr. D. J. Garkal

Principal

JCEI's JAIHIND COLLEGE OF ENGINEERING, KURAN

INDEX

Sr. No.	Date	TITLE	Sign	Remark
1		Implement a class Complex which represents the Complex Number data type. Implement the following 1. Constructor (including a default constructor which creates the complex number 0+0i). 2. Overload operator+ to add two complex numbers. 3. Overload operator* to multiply two complex numbers. 4. Overload operators << and >> to print and read Complex Numbers.		
2		Develop a program in C++ to create a database of student's information system containing the following information: Name, Roll number, Class, Division, Date of Birth, Blood group, contact address, Telephone number, Driving license no. and other. Construct the database with suitable member functions. Make use of constructor, default constructor, copy constructor, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation operators-new and delete as well as exception handling.		
3		Imagine a publishing company which does marketing for book and audio cassette versions. Create a class publication that stores the title (a string) and price (type float) of publications. From this class derive two classes: book which adds a page count (type int) and tape which adds a playing time in minutes (type float). Write a program that instantiates the book and tape class, allows user to enter data and displays the data members. If an exception is caught, replace all the data member values with zero values.		

4		Write a C++ program that creates an output file, writes information to it, closes the file, open it again as an input file and read the information from the file.		
5		Write a function template for selection sort that inputs, sorts and outputs an integer array and a float array.		
6		Write C++ program using STL for sorting and searching user defined records such as Item records (Item code, name, cost, quantity etc) using vector container.		
7		Write a program in C++ to use map associative container. The keys will be the names of states and the values will be the populations of the states. When the program runs, the user is prompted to type the name of a state. The program then looks in the map, using the state name as an index and returns the population of the state.		

GROUP A

Practical No.1	(GROUP A)
Title	Arithmetic operations on complex numbers using operator overloading.
Subject	Object Oriented Programming
Name	
Class	S.E. (AIDS.E.)
Roll No.	
Date	
Signature	

Practical No: 1

Title: Arithmetic operations on complex numbers using operator overloading.

Problem Statement:

Implement a class Complex which represents the Complex Number data type. Implement the following operations:

1. Constructor (including a default constructor which creates the complex number $0+0i$).
2. Overloaded **operator+** to add two complex numbers.
3. Overloaded **operator*** to multiply two complex numbers.
4. Overloaded **<<** and **>>** to print and read Complex Numbers.

Prerequisites:

Object Oriented Programming

Objectives:

To learn the concept of constructor, default constructor, operator overloading using member function and friend function.

Theory:

Operator Overloading

It is a specific case of polymorphism where different operators have different implementations depending on their arguments. In C++ the overloading principle applies not only to functions, but to operators too. That is, operators can be extended to work not just with built-in types but also classes. A programmer can provide his or her own operator to a class by overloading the built-in operator to perform some specific computation when the operator is used on objects of that class.

An Example of Operator Overloading

Complex a(1.2,1.3); *//this class is used to represent complex numbers*

Complex b(2.1,3); *//notice the construction taking 2 parameters for the real and imaginary part*

Complex c = a+b; *//for this to work the addition operator must be overloaded*

Arithmetic Operators

Arithmetic Operators are used to do basic arithmetic operations like addition, subtraction, multiplication, division, and modulus.

The following table lists the arithmetic operators used in C++.

Operator	Action
+	Addition
-	Subtraction
*	Multiplication
/	Division

%	Modulus
---	---------

With C++ feature to overload operators, we can design classes able to perform operations using standard operators. Here is a list of all the operators that can be overloaded:

Over loadable operators

+ - * / = <> += -= *= /= <<>>
<<= >>= == != <= >= ++ -- % & ^ ! |

~ &= ^= |= && || %= []

5. To overload an operator in order to use it with classes we declare *operator functions*, which are regular functions whose names are the operator keyword followed by the operator sign that we want to overload. The format is:
6. type operator operator-symbol (parameters) { /*...*/ }
7. The **operator** keyword declares a function specifying what *operator-symbol* means when applied to instances of a class. This gives the operator more than one meaning, or "overloads" it. The compiler distinguishes between the different meanings of an operator by examining the types of its operands.

Syntax:

return_type class_name :: operator op(arg_list)

{

//function body

}

where,

8. Return type is the value returned by the specified operation
9. op is the operator to be overload.
10. op is proceeding by the keyword operator.
11. operator op is the function name

Process of the overloading has 3 steps

1. Create a class that define a data types that is used in the overloading operation
2. Declare the operator function operator op () in the public part of the class. It may be either a member function or a friend function.
3. Define the operator function to implement the required operation

e.g.

Overloading Binary operators:

A statement like

C = sum (A, B); // functional notation

This functional notation can be replaced by a natural looking expression

$C = A+B$; // arithmetic notation

by overloading the + operator using an operator+ () function Facilities:

Linux Operating Systems, G++

Algorithm:

Step 1: Start the program

Step 2: Create a class complex

Step 3: Define the default constructor.

Step 4: Declare the operator function which are going to be overloaded and display function

Step 5: Define the overloaded functions such as +, -, /, * and the display function

For Addition:

$$(a+bi) + (x + yi) = ((a+x)+(b+y)i)$$

For Multiplication:

$$(a+bi) * (x + yi) = (((a*x)-(b*y)) + ((a*y) + (x*b))i)$$

Step 6: Create objects for complex class in main () function

Step 7: Create a menu for addition, multiplication of complex numbers and display the result

Step 8: Depending upon the choice from the user the arithmetic operators will invoke the overloaded operator automatically and returns the result

Step 9: Display the result using display function.

Input:

Complex numbers with real and imaginary values for two complex numbers.

Example:

Complex No 1: Real Part :

5 Imaginary part : 4

Complex No 2: Real Part :

5 Imaginary part : 4

Output:

Default constructor value=0+0i

Enter the 1st number

Enter the real part2

Enter the imaginary part4

Enter the 2nd number

Enter the real part4

Enter the imaginary part8

The first number is 2+4i

The second number is 4+8i

The addition is 6+12i

The multiplication is -24+32i

Practical Program:

```
# include<iostream>
using namespace std;
class Complex
{
double real;
double
img;
public:
Complex();
friend istream & operator >> (istream &, Complex &);
friend ostream & operator << (ostream &, const Complex &);
Complex operator + (Complex);
Complex operator * (Complex);
};
Complex::Complex()
{
real = 0;
img = 0;
}
istream & operator >> (istream &, Complex & i)
{
cin >> i.real >> i.img;
return cin;
}
ostream & operator << (ostream &, const Complex & d)
{
cout << d.real << " + " << d.img << "i" << endl;
return cout;
}
Complex Complex::operator + (Complex c1)
{
Complex temp;
temp.real = real + c1.real;
temp.img = img +
c1.img; return temp;
}
Complex Complex::operator * (Complex c2)
{
Complex tmp;
tmp.real = real * c2.real - img * c2.img;
tmp.img = real * c2.img + img * c2.real;
return tmp;
}
int main()
{
Complex C1, C2, C3, C4;
int flag = 1;
char b;
while (flag == 1)
{
cout << "Enter Real and Imaginary part of the Complex Number 1 : \n";
cin >> C1; 2+4i;
cout << "Enter Real and Imaginary part of the Complex Number 2 : \n";
cin >> C2; 3+2i;
int f = 1;
while (f == 1)
{
```

```

cout << "Complex Number 1 : " << C1 << endl; 2+4i;
cout << "Complex Number 2 : " << C2 << endl; 3+2i;
cout << "*****MENU*****" << endl;
cout << "1. Addition of Complex Numbers" << endl;
cout << "2. Multiplication of Complex Numbers" << endl;
cout << "3. Exit\n";
int a;
cout << "Enter your choice from above MENU (1 to 3) : ";
cin >> a;
if (a == 1)
{
C3 = C1+C2;
cout << "Addition : " << C3 << endl;
cout << "Do you want to perform another operation (y/n) : \n";
cin >> b;
if (b == 'y' || b == 'Y')
{
f=1;
}
else
{
cout << "Thanks for using this program!!\n";
flag=0;
f=0;
}
}
else if (a == 2)
{
C4 = C1 * C2;
cout << "Multiplication : " << C4 << endl;
cout << "Do you want to perform another operation (y/n) : \n";
cin >> b;
if (b == 'y' || b == 'Y')
{
f=1;
}
else
{
cout << "Thanks for using this program!!\n";
flag=0;
f=0;
}
}
else
{
cout << "Thanks for using this program!!\n";
flag=0;
f=0;
}
}
}
return 0;

```

Practical Output:

ubuntu@user:~\$ g++ GroupA_Practical1.cpp

ubuntu@user:~\$./a.out

Enter Real and Imaginary part of the Complex Number 1 :

2

3

Enter Real and Imaginary part of the Complex Number 2 :

3

4

Complex Number 1 : $2 + 3i$

Complex Number 2 : $3 + 4i$

*****MENU*****

1. Addition of Complex Numbers

2. Multiplication of Complex Numbers

3. Exit

Enter your choice from above MENU (1 to 3) : 1

Addition : $5 + 7i$

Do you want to perform another operation (y/n) :

y

Complex Number 1 : $2 + 3i$

Complex Number 2 : $3 + 4i$

*****MENU*****

1. Addition of Complex Numbers

2. Multiplication of Complex Numbers

3. Exit

Enter your choice from above MENU (1 to 3) : 2

Multiplication : $-6 + 17i$

Do you want to perform another operation (y/n) :

n

Thanks for using this program!!

Conclusion:

Hence, we have studied concept of operator overloading.

Questions:

1. What is operator overloading?
2. What are the rules for overloading the operators?
3. State clearly which operators are overloaded and which operator are not overloaded?
4. State the need for overloading the operators.
5. Explain how the operators are overloaded using the friend function.
6. What is the difference between “overloading” and “overriding”?
7. What is operator function? Describe the syntax?
8. When is Friend function compulsory? Give an example?

Practical No.2	(GROUP A)
Title	Personnel information system using constructor, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation
Subject	Object Oriented Programming
Name	
Class	S.E. (AIDS.E.)
Roll No.	
Date	
Signature	

Practical No:2

Title: Personnel information system using constructor, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation.

Problem Statement:

Develop an object-oriented program in C++ to create a database of the personnel information system containing the following information: Name, Date of Birth, Blood group, Height, Weight, Insurance Policy, number, Contact address, telephone number, driving license no. etc. Construct the database with suitable member functions for initializing and destroying the data viz constructor, default constructor, copy, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation operators-new and delete as well as exception handling

Prerequisites:

Object Oriented Programming

Objectives:

To learn the concept of constructor, default constructor, copy, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation operators-new and delete.

Theory:

Constructor:

A special method of the class that will be automatically invoked when an instance of the class is created is called as constructor. Following are the most useful features of constructor.

- 1) Constructor is used for Initializing the values to the data members of the Class.
- 2) Constructor is that whose name is same as name of class.
- 3) Constructor gets Automatically called when an object of class is created.
- 4) Constructors never have a Return Type even void.
- 5) Constructor is of Default, Parameterized and Copy

Constructors. The various types of Constructor are as follows: -

Constructors can be classified into 3 types

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor

1. **Default Constructor:** - Default Constructor is also called as Empty Constructor which has no arguments and It is Automatically called when we create the object of class but Remember name of Constructor is same as name of class and Constructor never declared with the help of

Return Type. Means we can't declare a Constructor with the help of void Return Type., if we never Pass or declare any Arguments then this called as the Copy Constructors.

2. **Parameterized Constructor:** - This is another type constructor which has some Arguments and same name as class name but it uses some Arguments So For this, we have to create object of Class by passing some Arguments at the time of creating object with the name of class. When we pass some Arguments to the Constructor then this will automatically pass the Arguments to the Constructor and the values will retrieve by the Respective Data Members of the Class.

3. **Copy Constructor:** - This is also another type of Constructor. In this Constructor we pass the object of class into the Another Object of Same Class. As name Suggests you Copy, means Copy the values of one Object into another Object of Class .This is used for Copying the values of class object into another object of class So we call them as Copy Constructor and For Copying the values We have to pass the name of object whose values we wants to Copying and When we are using or passing an Object to a Constructor then we must have to use the & Ampersand or Address Operator.

Destructor: As we know that Constructor is that which is used for Assigning Some Values to data Members and For Assigning Some Values This May also used Some Memory so that to free up the Memory which is Allocated by Constructor, destructor is used which gets Automatically Called at the End of Program and we doesn't have to Explicitly Call a Destructor and Destructor Can't be Parameterized or a Copy This can be only one Means Default Destructor Which Have no Arguments. For Declaring a Destructor, we have to use ~tilde Symbol in front of Destructor.

Static members

A class can contain static members, either data or functions.

A static member variable has following properties:

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class.
- It is the visible only within the class but its lifetime is the entire program.

Static data members of a class are also known as "class variables", because there is only one unique value for all the objects of that same class. Their content is not different from one object static members have the same properties as global variables but they enjoy class scope. For that reason, and to avoid them to be declared several times, we can only include the prototype (its declaration) in the class declaration but not its definition (its initialization). In order to initialize a static data-member we must include a formal definition outside the class, in the global scope of this class to another. Because it is a unique variable value for all the objects of the same class, it can be referred to as a member of any object of that class or even directly by the class name (of course this is only valid for static members).

A static member function has following properties

- A static function can have access to only other static members (fun or var) declared in the same class
- A static function can be called using the class name instead of its object
name class name :: fun name ;

Static member functions are considered to have class scope. In contrast to non-static member functions, these functions have no implicit this argument; therefore, they can use only static data members, enumerators, or nested types directly. Static member functions can be accessed without using an object

of the corresponding class type. The following restrictions apply to such static functions:

1. They cannot access non-static class member data using the member-selection operators (. or –>).
2. They cannot be declared as virtual.
3. They cannot have the same name as a non-static function that has the same argument types.

E.g. // static members in classes

```
class StaticTest {
private: static int x;

public: static int count()
{ return x;
}
};
int StaticTest::x = 9;
int main()
{
printf_s("%d\n", StaticTest::count());
}
```

Output

9

Friend functions:

In principle, private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not affect friends. Friends are functions or classes declared as such. If we want to declare an external function as friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword friend.

Properties of friend function:

- It is not in the scope of the class to which it has been declared as friend.
- Since it is not in the scope of the class, it cannot be called using the object of that class
- It can be invoked like a normal function w/o the help of any object.
- It can be declared in private or in the public part of the class.
- Unlike member functions, it cannot access the member names directly and has to use an object name and dot operator with each member name.

// friend functions

```
#include <iostream>
using namespace std;
class CRectangle {
int width, height;
public:
void set_values (int, int);
int area () {return (width * height);}
friend CRectangle duplicate
(CRectangle);
};
void CRectangle :: set_values (int a, int b)
{ width = a;
height = b;
}
CRectangle duplicate (CRectangle rectparam)
```

```

{
CRectanglerectres;
rectres.width = rectparam.width*2; rectres.height = rectparam.height*2; return (rectres);
}
int main () {
CRectanglerect, rectb;
rect.set_values (2,3);
rectb = duplicate (rect);
cout<<rectb.area();
return 0;
}

```

The duplicate function is a friend of CRectangle. From within that function we have been able to access the members width and height of different objects of type CRectangle, which are private members. Notice that neither in the declaration of duplicate () nor in its later use in main () have we considered duplicate a member of class CRectangle.

Friend classes

Just as we have the possibility to define a friend function, we can also define a class as friend of another one, granting that second class access to the protected and private members of the first one.

```

// friend class
#include <iostream>
using namespace std;
class CSquare;
class CRectangle
{
int width, height;
public:
int area ()
{return (width * height);}
void convert (CSquare a);
};
class CSquare {
private:
int side;
public:
void set_side (int a)
{side=a;}
friend class CRectangle;
};
void CRectangle::convert (CSquare a) {
width = a.side;
height = a.side;
}
int main () {
CSquaresqr;
CRectanglerect;
sqr.set_side(4);
rect.convert(sqr);
cout<<rect.area();
}

```

```
return 0;
}
```

In this example, we have declared CRectangle as a friend of CSquare so that CRectangle member functions could have access to the protected and private members of CSquare, more concretely to CSquare :: side, which describes the side width of the square..

Pointers:

A pointer is a derived data type that refers to another data variable by storing the variables memory address rather than data.

Declaration of pointer variable is in the following form:

Eg. `int *ptr;`

`Data_type * ptr_var;`

Here ptr is a pointer variable and points to an integer data type.

We can initialize pointer variable as follows

```
int a, *ptr; // declaration
```

```
ptr = &a //initialization
```

Pointers to objects:

Consider the following eg

item X; // where item is class and X is object

Similarly we can define a pointer it_ptr of type item as follows

```
Item *it_ptr ;
```

Object pointers are useful in creating objects at runtime. We can also access public members of the class using pointers.

Eg item X;

```
item *ptr = &X;
```

the pointer 'ptr' is initialized with address of X.

we can access the member functions and data using pointers as

follows `ptr->getdata();`

```
ptr->show();
```

this pointer:

C++ uses a unique keyword called this to represent an object that invokes a member function. this is a pointer that points to the object for which this function was called. This unique pointer is automatically passed to a member function when it is called.

Important notes on this pointer:

- this pointer stores the address of the class instance, to enable pointer access of the members to the member functions of the class.
- this pointer is not counted for calculating the size of the object.
- this pointers are not accessible for static member functions.
- this pointers are not modifiable.

Facilities:

Linux Operating Systems, G++

Algorithm:

1. Start
2. Read personnel information such as Name, Date of Birth, Blood group, Height, Weight, Insurance Policy, number, Contact address, telephone number, driving license no..
3. Print all information from database.
4. Stop

Input:

Personnel information such as Name, Date of Birth, Blood group, Height, Weight, Insurance Policy, number, contact address, telephone number, driving license no.

Output:

Display personnel information from database. The result in following format:

Name DOB Driving License No

1

2

.

.

N

Practical Program:

```
#include<iostream>

#include<string.h>

using namespace std;

class Data
{
    string address;
    long int *telephone;
    long int *dl_no;

public:
    Data();    // default constructor of class Data
    ~Data();   // Destructor of class Data
    void input_student_data();
    void display_student_data();
    friend class Student;
};

class Student
{
    string Name;
    int Roll;
    string Class;
    char *Div;
    string dob;
    char *bg;
    static int count;

public:
    Student();
    ~Student();
```

```

static int get_count()
{
return count;
}

void input_data(Data *);
void display_data(Data *);
};

Data::Data()
{
address = " ";
telephone = new long;
dl_no = new long;
}

Data::~Data()
{
delete telephone;
delete dl_no;
}

void Data::input_student_data()
{
cout<<"Contact Address : ";
cin.ignore();
getline(cin,address);
cout<<"Moblie Phone Number : ";
cin>>*telephone;
cout<<"Driving License Number : ";
cin>>*dl_no;
}

void Data::display_student_data()
{
cout<<"Contact Address : "<<address<<endl;
cout<<"Mobile Phone Number : "<<*telephone<<endl;
}

```

```

cout<<"Driving License Number : "<<*dl_no<<endl;
}
Student::Student()
{

Name = " ";
Roll = 0;
Class = " ";
Div = new char;
dob = "dd/mm/yyyy";
bg = new char[5];
}
Student::~~Student()
{
delete Div;
delete[] bg;
}
inline void Student::input_data(Data *stud1)
{
cout<<"Name : ";
cin.ignore();
getline(cin,Name);
cout<<"Roll Number :
"; cin>>Roll;
cout<<"Class : ";
cin.ignore();
getline(cin,Class);
cout<<"Division : ";
cin>>Div;
cout<<"Date of Birth : ";
cin.ignore();
getline(cin,dob);
cout<<"Blood Group : ";

```



```

cin>>bg;

stud1->input_student_data();

count++;

}

inline void Student::display_data(Data *stud2)
{
    cout<<"Name : "<<Name<<endl;
    cout<<"Roll Number : "<<Roll<<endl;
    cout<<"Class : "<<Class<<endl;
    cout<<"Division : "<<Div<<endl;
    cout<<"Date of Birth : "<<dob<<endl;
    cout<<"Blood Group : "<<bg<<endl;
    stud2->display_student_data();
    cout<<"_____ \n";
}

int Student::count;

int main()
{

    Student *st1[100];
    Data *st2[100];

    int a;
    int s=0;
    char ch;

    do
    {
        cout<<"_____MENU_____ \n";
        cout<<"1. New Student\n";
        cout<<"2. View Database\n";
        cout<<"3.Exit\n";

        cout<<"Enter your choice : "<<endl;

        cin>>a;
    }

```

```

switch (a)
{
    case 1 :
    {

do
    {
st1[s] = new Student;
st2[s] = new Data;
cout<<"Enter the details of student "<<s+1<<" : "<<endl;
st1[s]->input_data(st2[s]);
s++;
cout<<"Do you want to create a new student (y/n): ";
cin>>ch;

} while(ch=='y' || ch=='Y');
}
break;

case 2:
{

cout<<"*****StudentDatabase*****"<<endl;
l; cout<<"Total Number of Students : "<<Student::get_count()<<endl;
cout<<"!!!!!!Thankyou For Using Program!!!!!!"<<endl;
for(int j=0;j<s;j++)
{
st1[j]->display_data(st2[j]);
}

}
break;
case 3:

```

```
{  
    exit(0);  
}  
default:  
cout<<"WRONG CHOICE TRY AGAIN\n";  
}  
}while(ch!=3);  
return 0;  
}
```

Practical Output:

ubuntu@user:~\$ g++ GroupA_Practical2.cpp

ubuntu@user:~\$./a.out

_____MENU_____

1. New Student

2. View

Database Enter

your choice :

3

WRONG CHOICE EXECUTE PROGRAM AGAIN

_____MENU_____

1. New Student

2. View

Database Enter

your choice :

1

Enter the details of student 1 :

Name : OMKAR

Roll Number : 29

Class : SE

COMP Division :

A

Date of Birth : 17/09/2000

Blood Group : B+

Contact Address : Junnar.

Moblie Phone Number : 7028888888

Driving License Number : 3423

Do you want to create a new student (y/n): y

Enter the details of student 2 :

Name : ATHARVA

Roll Number : 27

Class : SE

COMP

Division : A

Date of Birth : 4/12/2000

Blood Group : B+

Contact Address : Junnar

Moblie Phone Number : 8484999999

Driving License Number : 2341

Do you want to create a new student (y/n): n

_____MENU_____

1. New Student

2. View

Database Enter

your choice :

2

*****StudentDatabase*****

*** Name : OMKAR

Roll Number : 29

Class : SE

COMP Division :

A

Date of Birth : 17/09/2000

Blood Group : B+

Contact Address : Junnar.

Mobile Phone Number : 7028888888

Driving License Number : 3423

Name : ATHARVA

Roll Number : 27

Class : SE

COMP Division :

A

Date of Birth : 4/12/2000

Blood Group : B+

Contact Address : Junnar

Mobile Phone Number : 8484999999

Driving License Number : 2341

Total Number of Students : 2

!!!!!!Thankyou For Using Program!!!!

Conclusion:

Hence, we have successfully studied concept of constructor, default constructor, copy constructor, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation operators-new and delete.

Questions:

1. What is static Function?
2. What is friend function? State the advantage of using the friend function.
3. What is friend class? Explain with examples.
4. Explain with examples pointers to object.
5. What is this pointer? Explain with examples.
6. State the advantages of this pointer.
7. What are inline functions?
8. How do we declare member of a class static?
9. What are demerits of friend function?
10. What is concept of constructor, destructor?
11. What are types of constructors?

Practical No.3	(GROUP A)
Title	Creating a class which uses the concept of inheritance, displays data and data members and uses the concept of exception handling.
Subject	Object Oriented Programming
Name	
Class	S.E. (AIDS.E.)
Roll No.	
Date	
Signature	

Practical No:3

Title: Creating a class which uses the concept of inheritance, displays data and data members and uses the concept of exception handling.

Problem Statement:

Imagine a publishing company which does marketing for book and audio cassette versions. Create a class publication that stores the title (a string) and price (type float) of publications. From this class derive two classes: book which adds a page count (type int) and tape which adds a playing time in minutes (type float). Write a program that instantiates the book and tape class, allows user to enter data and displays the data members. If an exception is caught, replace all the data member values with zero values.

Prerequisites:

Object Oriented Programming

Objectives:

To learn the concept of inheritance and exception handling.

Theory:

Inheritance:

Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes. New classes inherit some of the properties and behavior of the existing classes. An existing class that is "parent" of a new class is called a base class. New class that inherits properties of the base class is called a derived class. Inheritance is a technique of code reuse. It also provides possibility to extend existing classes by creating derived classes.

The basic syntax of inheritance is:

Class Derived Class : access Specifier Base Class

There are 3 access specifiers:

Namely public, private and protected.

public:

This inheritance mode is used mostly. In this the protected member of Base class becomes protected members of Derived class and public becomes public.

protected:

In protected mode, the public and protected members of Base class become protected members of Derived class.

private:

In private mode the public and protected members of Base class become private members of Derived class.

Types of Inheritance

In C++, we have 5 different types of Inheritance. Namely,

Single Inheritance

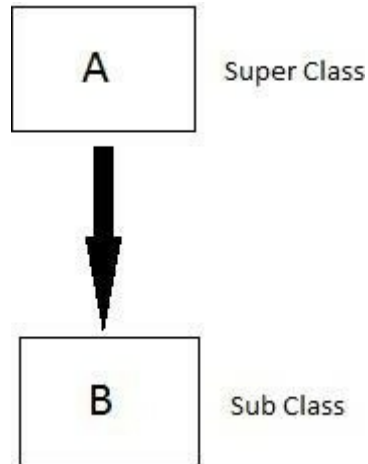
Multiple Inheritance

Hierarchical Inheritance

Multilevel Inheritance
Hybrid Inheritance

Single Inheritance:

In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance.



Syntax:

```
class subclass_name : access_modebase_class
{
//body of subclass
};
```

// Single Inheritance

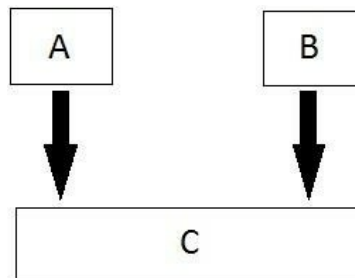
```
#include
<iostream> using
namespacestd; class
Vehicle
{
public:
Vehicle()
{
cout<< "This is a Vehicle"<<endl;
}
};
Class Car: public Vehicle
{
};
int main()
{
Car obj;
return0;
}
```

Output:

This is a Vehicle.

Multiple Inheritance:

In this type of inheritance, a single derived class may inherit from two or more than two base classes.



Syntax:

```
classsubclass_name : access_mode base_class1, access_mode base_class2, ....  
{  
    //body of subclass  
};
```

// Multiple Inheritance

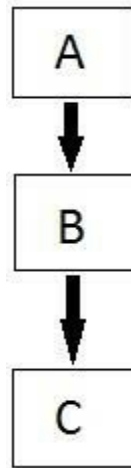
```
#include <iostream>  
using namespace std;  
Class Vehicle  
{ public:  
Vehicle()  
{  
cout<< "This is a Vehicle"<<endl;  
}  
};  
Class FourWheeler  
{ public:  
FourWheeler()  
{  
cout<< "This is a 4 wheeler Vehicle"<<endl;  
}  
};  
Class Car: public Vehicle, public FourWheeler  
{  
};  
int main()  
{  
Car obj;  
return 0;  
}
```

Output:

```
This is a Vehicle  
This is a 4 wheeler Vehicle
```

Multilevel Inheritance:

In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.



// Multilevel Inheritance

```
#include <iostream>
using namespace
std; class Vehicle
{
Public
Vehicle()
{
cout<< "This is a Vehicle"<<endl;
}
};
Class fourWheeler : public Vehicle
{ public:
fourWheeler()
{
cout<<"Objects with 4 wheels are vehicles"<<endl;
}
};
Class Car: public fourWheeler{
public:
car()
{
cout<<"Car has 4 Wheels"<<endl;
}
};
int main()
{
Car obj;
return 0;
}
```

Output:

This is a Vehicle

Objects with 4 wheels are vehicles
Car has 4 Wheels

Hierarchical Inheritance:

In this type of inheritance, multiple derived classes inherit from a single base class.

// Hierarchical Inheritance

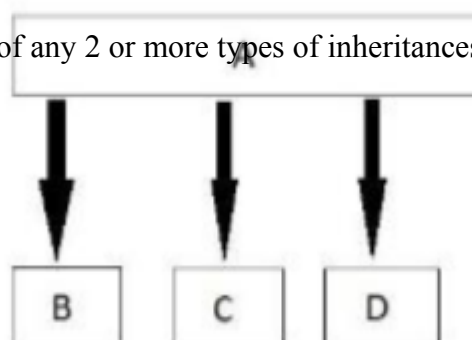
```
#include <iostream>
using namespace
std; class Vehicle
{
public:
Vehicle()
{
cout<< "This is a Vehicle"<<endl;
}
};
//Hybrid Inheritance
#include <iostream>
using namespace std;
class Car: public Vehicle
{
};
class Bus: public Vehicle
{
};
int main()
{
Car obj1;
Bus
obj2;
return 0;
}
```

Output:

This is a Vehicle
This is a Vehicle

Hybrid Inheritance:

Hybrid Inheritance is combination of any 2 or more types of inheritances.



//Hybrid Inheritance

```
#include <iostream>
using namespace
std; class Vehicle
{
public:
Vehicle()
{
cout<< "This is a Vehicle"<<endl;
}
};
class Fare
{
public:
Fare()
{
cout<<"Fare of Vehicle\n";
}
};
class Car: public Vehicle
{
};
class Bus: public Vehicle, public Fare
{
};
int
main({
Bus obj2;
return 0;
}
```

Output:

This is a Vehicle
Fare of Vehicle

Exception Handling:

Exception handling is part of C++ and object oriented programming. they are added in C++ to handle the unwanted situations during program execution. If we do not type the program correctly then it might result in errors. Main purpose of exception handling is to identify and report the runtime error in the program.

Famous examples are divide by zero, array index out of bound error, file not found, device not found, etc.

C++ exception handling is possible with three keywords i.e. try, catch and throw. Exception handling performs the following tasks:-

- Find the problem in the given code. It is also called as hit exception.
- It informs error has occurred. It is called as throwing the exception.
- We receive the error info. It is called as catching the exception.
- It takes the corrective action. It is called as exception handling.

TRY:- It is block code in which there are chances of runtime error. This block is followed by one or more catch block. Most error prone code is added in try block.

CATCH:- This is used to catch the exception thrown by the try block. In catch block we take corrective action on throwing exception. If files are opened , we can take corrective action like closing file handles, closing database connections, saving unsaved work ,etc.

THROW:- Program throws exception when problem occurs. It is possible with throw keyword.

SNYNTAX:

```
//normal program code
{ throw exception
}
catch(argument)
{
...
...
}

//rest of the code
```

// Exception Handling

```
#include <iostream>
using namespace std; int
main()
{
int x = -1; // Some code
cout<< "Before try
\n"; try {
cout<< "Inside try
\n"; if (x < 0)
{
throw x;
cout<< "After throw (Never executed) \n";
}
}
catch (int x ) {
cout<< "Exception Caught \n";
}
cout<< "After catch (Will be executed) \n";
return 0;
}
```

Output:

Before try

Inside try

Exception Caught

After catch (Will be executed)

Facilities:

Linux Operating Systems, G++

Algorithm:

1. Start.
2. Create classes Publication, book and tape.
- 3 . Publication class having data members title, price.
4. Class Book having data members pages and member functions getdata() and pudata().
5. Class Tape having data members minutes and member functions getdata() and pudata().
6. Create an object bof class book and object t of class tape.
7. Stop.

Input:

A class publication that stores the title (a string) and price (type float) of publications. Derives two classes Book and Tape.

Output:

Display title and price from publication class. The result in following format:

Enter Title: OOP
Enter Price: 300
Enter Pages:
250 Enter Title:
POP Enter
Price: 200
Enter Minutes:
60 Title: OOP
Price: 300
Pages:
250 Title:
POP
Price: 200
Minutes: 60

Practical Program:

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

class publication          // declaring class Publication
{
private:
    string title;
    float price;
public:
    void add()
    {
        cout << "\nEnter the Publication information : " << endl;
        cout << "Enter Title of the Publication : ";
        cin.ignore();
        getline(cin, title);
        cout << "Enter Price of Publication : ";
        cin >> price;
    }
    void display()
    {
        cout << "\n_____
"; cout << "\nTitle of Publication : " << title;
        cout << "\nPublication Price : " << price;
    }
};

class book : public publication // declaring class book which inherits class publication in public mode.
{
private:
    int page_count;
public:
    void add_book()
```



```

{
try
{
add();
cout << "Enter Page Count of Book : ";
cin >> page_count;
if (page_count <= 0)
{
throw page_count;
}
}
catch(...)
{
cout << "\nInvalid Page Count!!!";
page_count = 0;
}
}
void display_book()
{
display();
cout << "\nPage Count : " << page_count;
cout << "\n_____ \n";
}
};

class tape : public publication // declaring class tape which inherits class publication in public mode
{
private:
float play_time;
public:
void add_tape()
{
try
{

```

```

add();

cout << "Enter Play Duration of the Tape : ";

cin >> play_time;

if (play_time <= 0)

throw play_time;

}

catch(...)

{

cout << "\nInvalid Play Time!!!";

play_time = 0;

}

}

void display_tape()

{

display();

cout << "\nPlay Time : " << play_time << " min";

cout << "\n_____ \n";

}

};

int main()

{

book b1[10];      // object of class book

tape t1[10];      // object of class tape

int ch, b_count = 0, t_count = 0;

do

{

cout << "\n* * * * PUBLICATION DATABASE SYSTEM * * * *";

cout << "\n_____MENU_____";

cout << "\n1. Add Information to Books";

cout << "\n2. Add Information to Tapes";

cout << "\n3. Display Books Information";

cout << "\n4. Display Tapes Information";

cout << "\n5. Exit";

```

```

cout << "\n\nEnter your choice : ";

cin >> ch;

switch(ch)

{

case 1:

b1[b_count].add_book();

b_count ++;

break;

case 2:

t1[t_count].add_tape();

t_count ++;

break;

case 3:

cout << "\n* * * * BOOK PUBLICATION DATABASE SYSTEM * * * *";

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

{

b1[j].display_book();

}

break;

case 4:

cout << "\n* * * * TAPE PUBLICATION DATABASE SYSTEM * * * *";

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

{

t1[j].display_tape();

}

break;

case 5:

exit(0)

;

}

}while (ch != 5);

return 0;

}

```

Practical Output:

ubuntu@user:~\$ g++ GroupA_Practical3.cpp

ubuntu@user:~\$./a.out

***** PUBLICATION DATABASE SYSTEM *****

_____MENU_____

1. Add Information to Books
2. Add Information to Tapes
3. Display Books Information
4. Display Tapes Information
5. Exit

Enter your choice : 1

Enter the Publication information :

Enter Title of the Publication : Object Oriented Programming

Enter Price of Publication : 350

Enter Page Count of Book : 300

***** PUBLICATION DATABASE SYSTEM *****

_____MENU_____

1. Add Information to Books
2. Add Information to Tapes
3. Display Books Information
4. Display Tapes Information
5. Exit

Enter your choice : 2

Enter the Publication information :

Enter Title of the Publication : Storytel

Enter Price of Publication : 250

Enter Play Duration of the Tape : 120

* * * * * PUBLICATION DATABASE SYSTEM * * * * *

_____MENU_____

1. Add Information to Books
2. Add Information to Tapes
3. Display Books Information
4. Display Tapes Information
5. Exit

Enter your choice : 3

* * * * * BOOK PUBLICATION DATABASE SYSTEM * * * * *

Title of Publication : Object Oriented Programming

Publication Price : 350

Page Count : 300

* * * * * PUBLICATION DATABASE SYSTEM * * * * *

_____MENU_____

1. Add Information to Books
2. Add Information to Tapes
3. Display Books Information
4. Display Tapes Information
5. Exit

Enter your choice : 4

* * * * * TAPE PUBLICATION DATABASE SYSTEM * * * * *

Title of Publication : Storytel

Publication Price : 250

Play Time : 120 min

***** PUBLICATION DATABASE SYSTEM *****

____MENU____

1. Add Information to Books
2. Add Information to Tapes
3. Display Books Information
4. Display Tapes Information
5. Exit

Enter your choice : 5

Conclusion:

Hence, we have successfully studied concept of inheritance and exception handling.

Questions:

1. What is Inheritance?
2. What are types of Inheritance?
3. What is Single Inheritance?
4. What is Multiple Inheritance?
5. What is Hierarchical Inheritance?
6. What is Multilevel Inheritance?
7. What is Hybrid Inheritance?
8. What is Exception handling?
9. What are try catch block of exception handling?

GROUP B

Practical No.4	(GROUP B)
Title	Write a C++ program that creates an output file, writes information to it, closes the file and open it again as an input file and read the information from the file.
Subject	Object Oriented Programming
Name	
Class	S.E. (AIDS.E.)
Roll No.	
Date	
Signature	

Practical No: 4

Title: File handing

Problem Statement:

Write a C++ program that creates an output file, writes information to it, closes the file and open it again as an input file and read the information from the file.

Prerequisites:

Object Oriented Programming

Objectives:

To learn the concept of File handing.

Theory:

Stream:

A stream is a sequence of bytes. It acts as source from which the input data can be obtained or as a destination to which the output data can be sent.

1. Input Stream

Input Streams are used to hold input from a data producer, such as a keyboard, a file, or a network. The source stream that provides data to the program is called the input stream. A program extracts the bytes from the input stream. In most cases the standard input device is the keyboard. With the cin and “extraction” operator (>>) it is possible to read input from the keyboard.

2. Output Stream

Output Streams are used to hold output for a particular data consumer, such as a monitor, a file, or a printer. The destination stream that receives data from the program is called the output stream. A program inserts the bytes into an output stream. By default, the standard output of a program points at the screen. So with the cout operator and the “insertion” operator (<<) you can print a message onto the screen.

iostream standard library provides cin and cout methods for reading from standard input and writing to standard output respectively.

file handling provides three new datatypes:

Data Type	Description
ofstream	This data type represents the output file stream and is used to create files and to write information to files.
ifstream	This data type represents the input file stream and is used to read information from files.
fstream	This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files.

To perform file processing in C++, header file `<iostream>` and `<fstream>` must be included in your C++ source file.

Opening a File

- A file must be opened before you can read from it or write to it.
- Either the ofstream or fstream object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only.
- Following is the standard syntax for open() function which is a member of fstream, ifstream and ofstream objects.

```
void open(const char *filename, ios::openmode mode);
```
- Here, the first argument specifies the name and location of the file to be opened and the second argument of the open() member function defines the mode in which the file should be opened.

Mode Flag	Description
ios::app	Append mode. In this All output to that file to be appended to the end.
ios::ate	Open a file for output and move the read/write control to the end of the file.
ios::in	Open a file for reading.
ios::out	Open a file for writing.
ios::trunc	If the file already exists, its contents will be truncated before opening the file.

- You can combine two or more of these values by OR reading them together.
- For example, if you want to open a file in write mode and want

to truncate it in case it already exists, following will be the syntax:
ofstream outfile;

```
outfile.open("file.dat", ios::out | ios::trunc );
```

Similar way, you can open a file for reading and writing purpose as follows:

```
fstream afile;
```

```
afile.open("file.dat", ios::out | ios::in );
```

Closing a File

When a C++ program terminates it automatically closes flushes all the streams, release all the allocated memory and close all the opened files

- It is always a good practice that a programmer should close all the opened files before program termination.
- Following is the standard syntax for close() function, which is a member of fstream, ifstream, and ofstream objects.
void close();

Writing to a File

- While doing C++ programming, you write information to a file from your program using the stream insertion operator (<<) just as you use that operator to output information to the screen.
- The only difference is that you use an ofstream or fstream object instead of the cout object.

Reading from a File

- You read information from a file into your program using the stream extraction operator (>>) just as you use that operator to input information from the keyboard.
- The only difference is that you use an ifstream or fstream object instead of the cin object.

Example

```
file .read ((char *)&V , sizeof(V)); file . Write ((char *)&V , sizeof(V));
```

- These functions take two arguments. The first is the address of the variable V , and the second is the length of that variable in bytes . The address of variable must be cast to type char * (i.e pointer to character type) .

Facilities:

Linux Operating Systems, G++

Algorithm:

1. Start
2. Create a class
3. Define data members roll number and name.
4. Define accept () to take name and roll number from user.
5. Define display () to display the record.
6. In main () create the object of class and fstream class.

Take a limit from user in n variable

8. Open the file in out mode, call accept () to take record from user, then call write () to write that record into the file and at the end close that file.
9. Open the file in in mode, read the record from the file, call display () function to display the record and at the end close that file.
10. Stop

Input:

how many record you want 3

1 abc

2 pqr

3 xyz

4

Output:

name=abc

Roll=1

name=pqr

Roll=2

name=xyz

Roll=3

Practical Program:

```
#include<iostream>

#include<fstream>

using namespace std;

class Employee      // declaring class employee
{
    string Name;
    int ID;
    double salary;
public:
    void accept()
    {
        cout<<"\n Name : ";
        cin.ignore();
        getline(cin,Name);
        cout<<"\n Id : ";
        cin>>ID;
        cout<<"\n Salary : ";
        cin>>salary;
    }
    void display()
    {
        cout<<"\n Name : "<<Name;
        cout<<"\n Id : "<<ID;
        cout<<"\n Salary : "<<salary<<endl;
    }
};

int main()
{
    Employee o[5];
```

```

fstream f;

int i,n;

f.open("demo2.txt",ios::out);
cout<<"\n Enter the number of employees you want to store : ";
cin>>n;
for(i=0;i<n;i++)
{
cout<<"\n Enter information of Employee "<<i+1<<"\n";
o[i].accept();
f.write((char*)&o[i],sizeof o[i]);
}

f.close();

f.open("demo2.txt",ios::in);
cout<<"\n Information of Employees is as follows : \n";
for(i=0;i<n;i++)
{
cout<<"\nEmployee "<<i+1<<"\n";
f.read((char*)&o[i],sizeof o[i]);
o[i].display();
}
f.close();

return 0;
}

```

Practical Output:

ubuntu@user:~\$ g++ GroupB_Practical4.cpp

ubuntu@user:~\$./a.out

Enter the number of employees you want to store : 2

Enter information of Employee 1

Name : Omkar

Id : 234

Salary : 35000

Enter information of Employee 2

Name : Abhishek

Id : 765

Salary : 48000

Information of Employees is as follows :

Employee 1

Name : Omkar

Id : 234

Salary : 35000

Employee 2

Name : Abhishek

Id : 765

Salary : 48000

Conclusion:

Hence, we have studied concept of File handing

Questions:

1. What is file handling?
2. What are the different benefits of file handling?
3. What is fstream class?
4. How to create object of fsream class?
5. Explain the syntax of read() ?
6. .Explain the syntax of write()?

Practical No.5	(GROUP B)
Title	Write a function template selection sort. Write a program that inputs, sorts and outputs an integer array and a float array.
Subject	Object Oriented Programming
Name	
Class	S.E. (AIDS.E.)
Roll No.	
Date	
Signature	

Practical No: 5

Title: Function Template

Problem Statement:

Implement a function template selection Sort. Write a program that inputs, sorts and outputs an integer array and a float array.

Prerequisites:

Object Oriented Programming

Objectives:

To learn the concept of Template.

Theory:

Templates

Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.

A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept. There is a single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector <int> or vector <string>.

You can use templates to define functions as well as classes, let us see how do they work:

Function Template:

The general form of a template function definition is shown here:

```
template <class type> ret-type func-name(parameter list)
{
// body of function
}
```

Here, type is a placeholder name for a data type used by the function. This name can be used within the function definition.

The following is the example of a function template that returns the maximum of two values:

```
#include
<iostream>
#include <string>
using namespace
std;
template <typename T>
inline T const& Max (T const& a, T const& b)
{
return a < b ? b:a;
}
int main ()
```

{

```
int i = 39; int j = 20;
cout<< "Max(i, j): " << Max(i, j) <<endl; double f1 =13.5; double f2 =20.7;
cout<< "Max(f1, f2): " << Max(f1, f2) <<endl; string s1 = "Hello"; string s2 = "World";
cout<< "Max(s1, s2): " << Max(s1, s2) <<endl; return 0;
}
```

If we compile and run above code, this would produce the following result:

```
Max(i, j): 39
Max(f1, f2): 20.7
Max(s1, s2): World
```

Class Template:

Just as we can define function templates, we can also define class templates. The general form of a generic class declaration is shown here:

```
template <class type> class class-name
{
.
.
.
}
```

Here, type is the placeholder type name, which will be specified when a class is instantiated.

You can define more than one generic data type by using a comma-separated list.

Following is the example to define class Stack<T> and implement generic methods to push and pop the elements from the stack:

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <string>
#include <stdexcept>
using namespace std;
template <class T> class Stack
{
private:
vector<T>elems; // elements
public:
void push(T const&); //push element
void pop(); // pop element
T top()const; // return top element bool empty()const
{ // return
true if empty.
return elems.empty();
};
};
template <class T>
void Stack<T>::push (T const&elem)
{
// append copy of passed element
elems.push_back(elem);
}
```

```

template<class T>
void Stack<T>::pop ()
{
if (elems.empty())
{
throw out_of_range("Stack<>::pop(): empty stack");
}
// remove last element
elems.pop_back();
}
template <class T>
T Stack<T>::top () const
{
if (elems.empty())
{
throw out_of_range("Stack<>::top(): empty stack");
}
// return copy of last element
return elems.back();
}
int main()
{
try
{
Stack<int> intStack; // stack of ints Stack<string>stringStack; // stack
of strings
// manipulate int stack int Stack.push(7);
cout<<intStack.top() <<endl;
// manipulate string stack stringStack.push("hello");
cout<<stringStack.top() <<std::endl; stringStack.pop();
stringStack.pop();
}
catch (exception const& ex)
{
cerr<< "Exception: " <<ex.what()
<<endl; return -1;
}
}
}
If we compile and run above code, this would produce the following result: 7
hello
Exception: Stack <> :: pop(): empty stack

```

Selection Sort:

Selection sort is a sorting algorithm, specifically an in-place comparison sort. It has $O(n^2)$ time complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited

How selection sort works?

Example



For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value



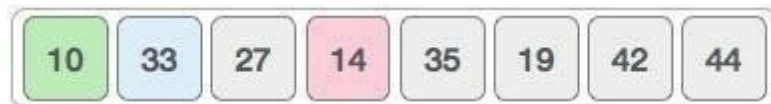
So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of sorted list.



For the second position, where 33 is residing, we start scanning the rest of the list in linear Manner.



We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.



After two iterations, two least values are positioned at the beginning in the sorted manner. The same process is applied on the rest of the items in the array. Pictorial depiction of entire sorting process is as follows:



Facilities:

Linux Operating Systems, G++

Algorithm:

1. Start
2. Declare the template parameter T.
3. Define template function for selection sort.
4. In main () Define two arrays, one for integer and another for float. and take a input for both the arrays and call sorting function template to sort the number.
5. Stop

Input:

Selection sort Integer Element

Enter how many elements

you want 5

Enter the Integer element

7

5

8

9

3

Float Element

Enter how many elements

you want 5

Enter the

float element

3.8

9.4

5.5

2.2

6.7

Output:

Sorted list= 3 5 7 8 9

Sorted list=2.2 3.8 5.5 6.7 9.4

Practical Program:

```
#include<iostream>

using namespace std;

int n;

#define size 10

template<class T>

void sel(T A[size])

{

int i,j,min;

T temp;

for(i=0;i<n-1;i++)

{

min=i;

for(j=i+1;j<n;j++)

{

if(A[j]<A[min])

min=j;

}

temp=A[i];

A[i]=A[min];

A[min]=temp;

}

cout<<"\nSorted array:";

for(i=0;i<n;i++)

{

cout<<" "<<A[i];

}

}

int main()

{

int A[size];

float B[size];
```

```

int i;

int ch;

do

{

cout<<"\n* * * * SELECTION SORT SYSTEM * * * * ";

cout<<"\n_____MENU_____";

cout<<"\n1. Integer Values";

cout<<"\n2. Float Values";

cout<<"\n3. Exit";

cout<<"\n\nEnter your choice : ";

cin>>ch;


switch(ch)

{

case 1:

cout<<"\nEnter total no of int elements:";

cin>>n;

cout<<"\nEnter int elements:";

for(i=0;i<n;i++)

{

cin>>A[i];

}

sel(A);

break;


case 2:

cout<<"\nEnter total no of float elements:";

cin>>n;

cout<<"\nEnter float elements:";

for(i=0;i<n;i++)

{

cin>>B[i];

}

```

```
sel(B);
```

```
break;
```

```
case 3:
```

```
exit(0);
```

```
}
```

```
}while(ch!=3);
```

```
return 0;
```

```
}
```

Practical Output:

ubuntu@user:~\$ g++ GroupB_Practical5.cpp

ubuntu@user:~\$./a.out

***** SELECTION SORT SYSTEM *****

_____MENU_____

1. Integer Values
2. Float Values
3. Exit

Enter your choice : 1

Enter total no of int elements:5

Enter int elements:

5
4
6
7
2

Sorted array: 2 4 5 6 7

***** SELECTION SORT SYSTEM *****

_____MENU_____

1. Integer Values
2. Float Values
3. Exit

Enter your choice : 2

Enter total no of float elements:5

Enter float elements:

3.4

2.7

3.0

2.5

1.5

Sorted array: 1.5 2.5 2.7 3 3.4

* * * * * SELECTION SORT SYSTEM * * * * *

_____MENU_____

1. Integer Values

2. Float Values

3. Exit

Enter your choice : 3

Conclusion:

Hence, we have studied concept of Function Template.

Questions:

1. What is template?
2. What is Function template?
3. What is Class template?
4. Explain template with function overloading.
5. Explain template with non-type argument.

GROUP C

Practical No.6	(GROUP C)
Title	Write a function template selection sort. Write a program that inputs, sorts and outputs an integer array and a float array.
Subject	Object Oriented Programming
Name	
Class	S.E. (AIDS.E.)
Roll No.	
Date	
Signature	

Practical No:6

Title: Personnel information system using sorting and searching for STL and vector container.

Problem Statement:

Write C++ program using STL for sorting and searching user defined records such as personal records (Name, DOB, Telephone number etc) using vector container. **OR**
Write C++ program using STL for sorting and searching user defined records such as Item records (Item code, name, cost, quantity etc) using vector container.

Prerequisites:

Object Oriented Programming

Objectives:

To learn the concept STL, searching, sorting and vector container.

Theory:

STL:

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. A working knowledge of template classes is a prerequisite for working with STL.

STL has four components

- Algorithms
- Containers
- Functions
- Iterators

Algorithms

- The algorithm defines a collection of functions especially designed to be used on ranges of elements. They act on containers and provide means for various operations for the contents of the containers.
- Algorithm
- Sorting
- Searching
- Important STL Algorithms
- Useful Array algorithms
- Partition Operations
- Numeric

Containers

- Containers or container classes store objects and data. There are in total seven standard “firstclass” container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors.

Sequence Containers: implement data structures which can be accessed in a sequential manner.

- vector
- list
- deque
- arrays
- forward_list(Introduced in C++11)
- Container Adaptors: provide a different interface for sequential containers.
- queue
- priority_queue
- stack
- Associative Containers: implement sorted data structures that can be quickly searched ($O(\log n)$ complexity).
- set
- multiset
- map
- multimap
- Unordered Associative Containers : implement unordered data structures that can be quickly searched
- unordered_set
- unordered_multiset
- unordered_map
- unordered_multimap

Functions

- The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters to be passed.

Iterators

- As the name suggests, iterators are used for working upon a sequence of values. They are the major feature that allow generality in STL.

Utility Library

- Defined in header <utility>.
- pair

Sorting:

It is one of the most basic functions applied to data. It means arranging the data in a particular fashion, which can be increasing or decreasing. There is a built in function in C++ STL by the name of sort(). This function internally uses Intro Sort. In more details it is implemented using hybrid of Quick Sort, Heap Sort and Insertion Sort. By default, it uses Quick Sort but if Quick Sort is doing unfair partitioning and taking more than $N \cdot \log N$ time, it switches to Heap Sort and when the array size becomes really small, it switches to Insertion Sort.

The prototype for sort is :

sort(startaddress, endaddress)

startaddress: the address of the first element of the array

endaddress: the address of the next contiguous location of the last element of the array.
So actually sort() sorts in the range of [startaddress , endaddress]

//Sorting

```
#include <iostream>
#include <algorithm>
using namespace std;
void show (int a [])
{
    for(int i = 0; i < 10; ++i)
        cout<< a[i] << " ";
}
int main ()
{
    int a [10] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    cout<< "\n The array before sorting is: ";
    show(a);
    sort(a, a+10);
    cout<< "\n\n The array after sorting is: "; show(a);
    return 0;
}
```

The output of the above program is:

The array before sorting is: 1 5 8 9 6 7 3 4 2 0

The array after sorting is: 0 1 2 3 4 5 6 7 8 9

Searching:

It is a widely used searching algorithm that requires the array to be sorted before search is applied. The main idea behind this algorithm is to keep dividing the array in half (divide and conquer) until the element is found, or all the elements are exhausted.

It works by comparing the middle item of the array with our target, if it matches, it returns true otherwise if the middle term is greater than the target, the search is performed in the left sub-array.

If the middle term is less than target, the search is performed in the right sub-array.

The prototype for binary search is :

binary_search(startaddress, endaddress, valuetofind)
startaddress: the address of the first element of the array.
endaddress: the address of the last element of the array.
valuetofind: the target value which we have to search for

//Searching

```
#include <algorithm>
#include <iostream>
using namespace std;
void show(inta[], intarraysize)
{
    for(int i = 0; i < arraysiz; ++i)
        cout<< a[i] << " ";
}
int main()
{
    inta[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    intasize = sizeof(a) / sizeof(a[0]);
    cout<< "\n The array is : ";
}
```

```

show(a, asize);
cout<< "\n\nLet's say we want to search for 2 in the array";
cout<< "\n So, we first sort the array";
sort(a, a + asize);
cout<< "\n\n The array after sorting is : ";
show(a, asize);
cout<< "\n\nNow, we do the binary search";
if(binary_search(a, a + 10, 2))
cout<< "\nElement found in the array";
else
cout<< "\nElement not found in the array";
cout<< "\n\nNow, say we want to search for 10";
if(binary_search(a, a + 10, 10))
cout<< "\nElement found in the array";
else
cout<< "\nElement not found in the array";
return 0;

```

Output:

The array is: 1 5 8 9 0 6 7 3 4 2 0
 Let's say we want to search for 2 in the array

So, we first sort the array
 The array after sorting is: 0 1 2 3 4 5 6 7 8 9
 Now, we do the binary search
 Element found in the array
 Now, say we want to search for 10

Element not found in the array

Facilities:

Linux Operating Systems, G++

Algorithm:

1. Start.
2. Give a header file to use 'vector'.
3. Create a vector naming 'personal_records'.
4. Initialize variables to store name, birth date and telephone number.
5. Using iterator store as many records you want to store using predefined functions as push_back().
6. Create another vector 'item_record'
7. Initialize variables to store item code, item name, quantity and cost.
8. Using iterator and predefined functions store the data.
9. Using predefined function sort (), sort the data stored according to user requirements.
10. Using predefined function search, search the element from the vector the user wants to check.
11. Display and call the functions using a menu.
12. End.

Input:

Personnel information such as name, DOB, telephone number.

Output:

Display personnel information from database. The result in following format:

***** Menu *****

1. I

nsert

2.Displa

y

3.Search

4.Sort

5.Delete

6.Exit

Enter your choice:1

Enter Item Name: bat

Enter Item Quantity:2

Enter Item Cost:50

Enter Item Code:1

Practical Program:

```
#include <iostream> //standard input output stream header file

#include <algorithm> //The STL algorithms are generic because they can operate on a variety of data structures

#include <vector> //The header file for the STL vector library is vector.


using namespace std;


class Item // creating class Item
{
public:
    char
    name[10]; int
    quantity; int
    cost;
    int code;


    bool operator==(const Item& i1) //Boolean operators allow you to create more complex conditional statements
    {
        if(code==i1.code) //operator will return 1 if the comparison is true, or 0 if the comparison is false
            return 1;
        return 0;
    }

    bool operator<(const Item& i1)
    {
        if(code<i1.code) //operator will return 1 if the comparison is true, or 0 if the comparison is false
            return 1;
        return 0;
    }
};


vector<Item> o1;


void print(Item &i1);
```



```

void display();
void insert();
void search();
void dlt();

bool compare(const Item &i1, const Item &i2)
{
    return i1.cost < i2.cost;
}

int main()
{
    int ch;

    do
    {
        cout<<"\n* * * * * Menu * * * * *";

        cout<<"\n1.Insert";

        cout<<"\n2.Display";

        cout<<"\n3.Search";

        cout<<"\n4.Sort";

        cout<<"\n5.Delete";

        cout<<"\n6.Exit";

        cout<<"\nEnter your choice : ";

        cin>>ch;

        switch(ch)
        {
            case 1:
                insert();
                break;

            case 2:
                display();
                break;

            case 3:
                search();
                break;

```

```

case 4:

sort(o1.begin(),o1.end(),compare);

cout<<"\n\n Sorted on Cost : ";

display();

break;

case 5:

dlt();

break;

case 6:

exit(0);

}

}while(ch!=7);

return 0;

}

void insert()

{

Item i1;

cout<<"\nEnter Item Name : ";

cin>>i1.name;

cout<<"\nEnter Item Quantity : ";

cin>>i1.quantity;

cout<<"\nEnter Item Cost : ";

cin>>i1.cost;

cout<<"\nEnter Item Code : ";

cin>>i1.code;

o1.push_back(i1);

}

void display()

{

for_each(o1.begin(),o1.end(),print);

}

void print(Item &i1)

{

```

```

cout<<"\n";
cout<<"\nItem Name : "<<i1.name;
cout<<"\nItem Quantity : "<<i1.quantity;
cout<<"\nItem Cost : "<<i1.cost;
cout<<"\nItem Code : "<<i1.code;
cout<<"\n\n";
}

void search()
{
vector<Item>::iterator p;
Item i1;
cout<<"\nEnter Item Code to search : ";
cin>>i1.code;
p=find(o1.begin(),o1.end(),i1);
if(p==o1.end())
{
cout<<"\nNot found!!!";
}
else
{
cout<<"\nFound!!!";
}
}

void dlt()
{
vector<Item>::iterator p;
Item i1;
cout<<"\nEnter Item Code to delete : ";
cin>>i1.code;
p=find(o1.begin(),o1.end(),i1);
if(p==o1.end())
{
cout<<"\nNot found!!!";
}
}

```

```
}  
else  
{  
o1.erase(p);  
cout<<"\nDeleted!!!";  
}  
}
```

Practical Output:

```
ubuntu@user:~$ g++ GroupC_Practical6.cpp
```

```
ubuntu@user:~$ ./a.out
```

```
***** Menu *****
```

```
1.Insert
```

```
2.      D
```

```
isplay
```

```
3.Search
```

```
4.Sort
```

```
5.Delete
```

```
6.Exit
```

```
Enter your choice : 1
```

```
Enter Item Name : Pen
```

```
Enter Item Quantity : 3
```

```
Enter Item Cost : 15
```

```
Enter Item Code : 100
```

```
***** Menu *****
```

```
1.Insert
```

```
2.Display
```

```
3.Search
```

```
4.Sort
```

```
5.Delete
```

```
6.Exit
```

```
Enter your choice : 1
```

```
Enter Item Name : Pencil
```

Enter Item Quantity : 5

Enter Item Cost : 25

Enter Item Code : 200

***** Menu *****

1.Insert

2.Display

3.Search

4.Sort

5.Delete

6.Exit

Enter your choice : 1

Enter Item Name : Eraser

Enter Item Quantity : 3

Enter Item Cost : 15

Enter Item Code : 300

***** Menu *****

1.Insert

2.Display

3.Search

4.Sort

5.Delete

6.Exit

Enter your choice : 2

Item Name : Pen

Item Quantity : 3

Item Cost : 15

Item Code : 100

Item Name :

Pencil Item

Quantity : 5 Item

Cost : 25 Item

Code : 200

Item Name : Eraser

Item Quantity : 3

Item Cost : 15

Item Code : 300

* * * * * Menu * * * * *

1.Insert

2.Display

3.Search

4.Sort

5.Delete

6.Exit

Enter your choice : 3

Enter Item Code to search : 400

Not found!!!

* * * * * Menu * * * * *

1.Insert

2.Display

3. S
earch

4.Sort

5.Delete

6.Exit

Enter your choice : 3

Enter Item Code to search : 100

Found!!!

* * * * * Menu * * * * *

1.Insert

2.Display

3.Search

4.Sort

5.Delete

6.Exit

Enter your choice : 4

Sorted on Cost :

Item Name : Pen

Item Quantity : 3

Item Cost : 15

Item Code : 100

Item Name :

Eraser Item

Quantity : 3 Item

Cost : 15

Item Code : 300

Item Name :

Pencil Item

Quantity : 5 Item

Cost : 25 Item

Code : 200

* * * * * Menu * * * * *

1.Insert

2.Display

3. S

earch

4.Sort

5.Delete

6.Exit

Enter your choice : 5

Enter Item Code to delete : 100

Deleted!!!

* * * * * Menu * * * * *

1.Insert

2.Display

3.Search

4.Sort

5.Delete

6.Exit

Enter your choice : 2

Item Name : Eraser

Item Quantity : 3

Item Cost : 15

Item Code : 300

Item Name :

Pencil Item

Quantity : 5 Item

Cost : 25 Item

Code : 200

* * * * * Menu * * * * *

1.Insert

2.Display

3.Search

4.Sort

5.Delete

6.Exit

Enter your choice : 6

Conclusion:

Hence, we have successfully studied the concept of STL(Standard Template Library) and how it makes many data structures easy. It briefs about the predefined functions of STL and their uses such as search() and sort()

Questions:

1. What is STL?
2. What are four components of STL?
3. What is Sorting?
4. What is Searching?
5. What vector container?

Practical No.7	(GROUP C)
Title	Write a program in C++ to use map associative container. The keys will be the names of states and the values will be the populations of the states. When the program runs, the user is prompted to type the name of a state. The program then looks in the map, using the state name as an index and returns the population of the state
Subject	Object Oriented Programming
Name	
Class	S.E. (AIDS.E.)
Roll No.	
Date	
Signature	

Practical No:7

Title: To use map associative container.

Problem Statement:

Write a program in C++ to use map associative container. The keys will be the names of states and the values will be the populations of the states. When the program runs, the user is prompted to type the name of a state. The program then looks in the map, using the state name as an index and returns the population of the state.

Prerequisites:

Object Oriented Programming

Objectives:

To learn the concept of map associative container

Theory:

Map Associative Container:

Map associative container are associative containers that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have same key values.

map::operator[]

This operator is used to reference the element present at position given inside the operator. It is similar to the at() function, the only difference is that the at() function throws an out-of-range exception when the position is not in the bounds of the size of map, while this operator causes undefined behaviour.

Syntax :

mapname[key]

Parameters :

Key value mapped to the element to be fetched.

Returns :

Direct reference to the element at the given key value

Examples:

```
Input: map mymap;  
mymap['a'] = 1;  
mymap['a'];  
Output : 1  
Input : map mymap;  
mymap["abcd"] = 7;
```

```
mymap["abcd"];  
Output : 7
```

//Program

```
#include <map>  
#include <iostream>  
#include <string>  
using namespace std;  
int main()  
{  
    // map declaration  
    map<int,string>mymap;  
    // mapping integers to strings  
    mymap[1] = "Hi";  
    mymap[2] = "This";  
    mymap[3] = "is";  
    mymap[4] = "NBN";  
    // using operator[] to print string  
    // mapped to integer 4  
    cout<<mymap[4];  
    return 0;  
}
```

Output:

NBN

Facilities:

Linux Operating Systems, G++

Algorithm:

1. Start.
2. Give a header file to map associative container.
3. Insert states name so that we get values as population of that state.
4. Use populationMap.insert().
5. Display the population of states.
6. End.

Input:

Information such as state name to map associative container.

Output:

Size of population Map: 5
Brasil: 193 million
China: 1339 million
India: 1187 million
Indonesia: 234 million
Pakistan: 170 million
Indonesia's populations is 234 million

Practical Program:

```
#include <iostream>

#include <map>

#include <string>

#include <utility>

using namespace std;

int main()

{

typedef map<string,int> mapType;

mapType populationMap;

populationMap.insert(pair<string, float>("Maharashtra", 125));

populationMap.insert(pair<string, float>("Uttar Pradesh", 225));

populationMap.insert(mapType::value_type("Bihar", 120));

populationMap.insert(mapType::value_type("West Bengal", 100));

populationMap.insert(make_pair("Madhya Pradesh", 90));

populationMap.insert(make_pair("Tamil Nadu", 80));

populationMap.insert(make_pair("Rajasthan", 78));

populationMap.insert(make_pair("Andhra Pradesh", 53));

populationMap.insert(make_pair("Odisha", 47));

populationMap.insert(make_pair("Kerala", 38));

populationMap.insert(make_pair("Telangana", 37));

populationMap.insert(make_pair("Assam", 35));

populationMap.insert(make_pair("Jharkhand", 38));

populationMap.insert(make_pair("Karnataka", 68));

populationMap.insert(make_pair("Gujarat", 70));

populationMap.insert(make_pair("Punjab", 31));

populationMap.insert(make_pair("Chhattisgarh", 30));

populationMap.insert(make_pair("Haryana", 29));

populationMap.insert(make_pair("UT Delhi", 19));

populationMap.insert(make_pair("UT Jammu and Kashmir", 14));

populationMap.insert(make_pair("Uttarakhand", 12));

populationMap.insert(make_pair("Himachal Pradesh", 8));

populationMap.insert(make_pair("Tripura", 04));
```



```

populationMap.insert(make_pair("Meghalaya", 4));
populationMap.insert(make_pair("Manipur", 3));
populationMap.insert(make_pair("Nagaland", 2));
populationMap.insert(make_pair("Goa", 2));
populationMap.insert(make_pair("Arunachal Pradesh", 2));
populationMap.insert(make_pair("UT Puducherry", 2));
populationMap.insert(make_pair("Mizoram", 1));
populationMap.insert(make_pair("UT Chandigarh", 1));
populationMap.insert(make_pair("Sikkim", 1));
populationMap.insert(make_pair("UT Dadra and Nagar Haveli and Daman and Diu", 1));
populationMap.insert(make_pair("UT Andaman and Nicobar Islands", 1));
populationMap.insert(make_pair("UT Lakshadweep", 0.0003));
populationMap.insert(make_pair("UT Ladakh", 0.00006));
mapType::iterator iter = --populationMap.end();
populationMap.erase(iter);

cout << "Total state and UT of India with Size of populationMap: " << populationMap.size() << '\n';
for (iter = populationMap.begin(); iter != populationMap.end(); ++iter)
{
    cout << iter->first << ":" << iter->second << " million\n";
}

char c;
do
{
    string state;

    cout<<"\nEnter that state you want to know the population of: ";
    cin>>state;

    iter = populationMap.find(state);
    if( iter != populationMap.end() )
        cout << state <<"s populations is "
        << iter->second << " million\n";
    else
        cout << "State is not in populationMap" << '\n';
    cout<<"Do you wish to continue?(y/n).:";

```

```
cin>>c;  
}while(c=='y'||c=='Y');  
populationMap.clear()  
; return 0;  
}
```

Practical Output:

ubuntu@user:~\$ g++ GroupC_Practical7.cpp

ubuntu@user:~\$./a.out

Total state and UT of India with Size of populationMap: 35

Andhra Pradesh:53 million

Arunachal Pradesh:2 million

Assam:35 million

Bihar:120 million

Chhattisgarh:30 million

Goa:2 million

Gujarat:70 million

Haryana:29 million

Himachal Pradesh:8 million

Jharkhand:38 million

Karnataka:68 million

Kerala:38 million

Madhya Pradesh:90 million

Maharashtra:125 million

Manipur[:3 million

Meghalaya:4 million

Mizoram:1 million

Nagaland:2 million

Odisha:47 million

Punjab:31 million

Rajasthan:78 million

Sikkim:1 million

Tamil Nadu:80 million

Telangana:37 million

Tripura:4 million

UT Andaman and Nicobar Islands:1 million

UT Chandigarh:1 million

UT Dadra and Nagar Haveli and Daman and Diu:1 million

UT Delhi:19 million

UT Jammu and Kashmir:14 million

UT Ladakh:0 million

UT Lakshadweep:0 million

UT Puducherry:2 million

Uttar Pradesh:225 million

Uttarakhand:12 million

Enter that state you want to know the population of: Maharashtra

Maharashtra's populations is 125 million

Do you wish to continue?(y/n):y

Enter that state you want to know the population of: pune

State is not in populationMap

Do you wish to continue?(y/n):y

Enter that state you want to know the population of: Sikkim

Sikkim's populations is 1 million

Do you wish to continue?(y/n):n

Conclusion:

Hence, we have successfully studied the concept of map associative container.

Questions:

1. What is an associative container in C++?
2. What is map in C++?
3. How to declare a map?
4. Explain Associative mapping with example?