A.Y 2021-2022	**Pimpri Chinchwad Education Trust's** **Pimpri Chinchwad College Of Engg. & Research.**	

LABORATORY MANUAL

Subject: Object Oriented Programming Laboratory

[SUBJECT CODE: 210247]

CLASS: S.E. Computer Engineering

YEAR: 2021 – 2022

Semester: I

PREPARED BY:

Prof. Vijay Kotkar
SUBJECT TEACHER

APPROVED BY:

Dr.A.A.Chaugule
H.O.D. [Comp]

List of Experiments

Sr. No.	Name of Experiment	Page No.
1	Design a class 'Complex 'with data members for real and imaginary part. Provide default and parameterized constructors. Write a program to perform arithmetic operations of two complex numbers using operator overloading. i. Addition and subtraction using friend functions ii. Multiplication and division using member functions	
2	Develop an object oriented program in C++ to create a database of student information system containing the following information: Name, Roll number, Class, division, Date of Birth, Blood group, 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 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 class publications that sores 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 members values with zero values.	
4	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.	
5	Write a function template selection sort. Write a program that inputs, sorts and outputs an integer array and a float array.	
6	Write C++ program using STL for Sorting and searching with user-defined records such as Person Record (Name, birth date, telephone no), item record (item code, item name, quantity and cost.	
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 as an index and returns the population of	

Vision – Mission of the Institute

Vision

To be a Premier institute of technical education and research to serve the need of society and all the stakeholders.

Mission

To establish state-of-the-art facilities to create an environment resulting in individuals who are technically sound having professionalism, research and innovative aptitude with high moral and ethical values.

Vision – Mission of the Computer Department

Vision

To strive for excellence in the field of Computer Engineering and Research through Creative Problem Solving related to societal needs

Mission:

1. Establish strong fundamentals, domain knowledge and skills among the students with analytical thinking, conceptual knowledge, social awareness and expertise in the latest tools & technologies to serve industrial demands
2. Establish leadership skills, team spirit and high ethical values among the students to serve industrial demands and societal needs
3. Guide students towards Research and Development, and a willingness to learn by connecting themselves to the global society.

Program Educational Objectives:

PO	Program Educational Outcomes
PEO1	To prepare globally competent graduates having strong fundamentals, domain knowledge, updated with modern technology to provide the effective solutions for engineering problems.
PEO2	To prepare the graduates to work as a committed professional with strong professional ethics and values, sense of responsibilities, understanding of legal, safety, health, societal, cultural and environmental issues.
PEO3	To prepare committed and motivated graduates with research attitude, lifelong learning, investigative approach, and multidisciplinary thinking.
PEO4	To prepare the graduates with strong managerial and communication skills to work effectively as individual as well as in teams.

Program Specific Outcomes

PSO	Program Specific Outcomes
PSO1	The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality project.
PSO2	The ability to understand, analyze and develop computer programs in the areas related to algorithms, software testing, application software, web design, data analytics, IOT and networkingfor efficient design of computer-based systems.
PSO3	The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies and to generate IPR &Deliver a quality project.

Program Outcomes

PO	Program Outcomes
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**:Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Objectives:

C207.1. To understand basics of object oriented programming

Course Outcomes:

.

CO	Statements
C207.1	U**nderstand** and **apply** the concepts like inheritance, polymorphism, exception handling and generic structures for implementing reusable programming codes
C207.2	**Analyze** the concept of file and **apply** it while storing and retrieving the data from secondary storages

EXPERIMENT NO.

(Reference for software related practical)

1. Aim

2. Objectives

3. Software's used

4. Theory

5. Flowchart

6. Algorithm

6. Procedure

7. Input

8. Output

9. Result

10. Conclusion

Experiment No: 1

AIM: Design a class 'Complex 'with data members for real and imaginary part. Provide default and parameterized constructors. Write a program to perform arithmetic operations of two complex numbers using operator overloading.

iii. Addition and subtraction using friend functions
iv. Multiplication and division using member functions

OBJECTIVES:

- To understand the concept of operator overloading
- To understand the concept of friend function
- To understand the concept of member function

SOFTWARE USED:

Linux Operating Systems, GCC

THEORY:

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

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

Box operator op (operator object);

1.Unary Operator:

The **unary operators** operate on a single operand and following are the examples of **Unary operators**: The increment (++) and decrement (--) **operators**. The **unary** minus (-) **operator**. The logical not (!) **operator**.

The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--. Following example explain how minus (-) operator can be overloaded for prefix as well as postfix usage.

```
#include <iostream>
using namespace std;
class Distance
{
private:
int feet;             // 0 to infinite
int inches;           // 0 to 12
public:
// required constructors
Distance(){
feet = 0;
inches = 0;
}
Distance(int f, int i){
feet = f;
inches = i;
}
// method to display distance
void displayDistance()
{
cout <<"F: "<< feet <<" I:"<< inches <<endl;
}
// overloaded minus (-) operator
Distance operator- ()
{
```

```
feet = -feet;
inches = -inches;
return Distance(feet, inches);
}
};
int main()
{
Distance D1(11, 10), D2(-5, 11);

-D1;                    // apply negation
D1.displayDistance();   // display D1

-D2;                    // apply negation
D2.displayDistance();   // display D2

return 0;
}
```

2.Binary Operator:

Overloading with a single parameter is called **binary operator overloading**. Similar to **unary operators**, **binary operators** can also be **overloaded**. **Binary operators**require two operands, and they are **overloaded** by using member functions and friend functions.

Example:

```
using namespace std;
class temp
{
complex operator + (complex c2)
        {
                complex c3;
                c3.x = x + c2.x;
                c3.y = y + c2.y;
```

```
        return c3;
    }
};
```

ALGORITHM:

1. Start.
2. Create class complex with data members x and y and member functions accept(), display().
3. Initialize 3 objects c1, c2, c3 and k in main().
5. Define default constructor to initialize variables to 0+0i.
6. Define operator overloaded functions to add, subtract, multiply and divide two complex numbers.
7. Call the operator overloaded functions.
8. Use c3=c1+c2; to invoke the overloaded + operator.
9. Use c3=c1-c2; to invoke the overloaded - operator.
10. Use c3=c1/c2; to invoke the overloaded / operator.
11. Use c3=c1*c2; to invoke the overloaded * operator.
12. After performing the required operations call display().
13. Stop.

INPUT:

3+2i (ordinary Number and Imaginary Number)

2+3i

OUTPUT:

5+5i

CONCLUSION:

Thus we studied concepts of friend function and member function.

PROGRAM:

```
#include<iostream>
#include<stdio.h>
#include<conio.h>
using namespace std;
class complex
{
        float x;
        float y;
        public:
                complex operator+(complex);
                complex operator-(complex);
                complex operator*(complex);
                complex operator/(complex);
                complex();
                complex(float,float);
                void display();
                void getdata();
};
complex::complex()
{
x=0;
y=0;
```

```
}
complex::complex(float a, float b)
{
        x=a;
        y=b;
}
complex complex::operator+(complex c)
{
complex temp;
        temp.x=x+c.x;
        temp.y=y+c.y;
        return(temp);
}
complex complex::operator-(complex c)
{
        complex temp1;
        temp1.x=x-c.x;
        temp1.y=y-c.y;
        return(temp1);
}
complex complex::operator*(complex c)
{
        complex temp2;
```

```
        temp2.x=(x*c.x)-(y*c.y);
        temp2.y=(y*c.x)+(x*c.y);
        return(temp2);
}
complex complex::operator/(complex c)
{
        complex temp3;
        temp3.x=((x*c.x)+(y*c.y))/((c.x*c.x)+(c.y*c.y));
        temp3.y=((y*c.x)-(x*c.y))/((c.x*c.x)+(c.y*c.y));
        return(temp3);
}
void complex::getdata()
{
cout<<"Enter real part";
        cin>>x;
        cout<<"Enter imaginary part";
        cin>>y;
}
void complex::display()
{
cout<<x<<"+"<<y<<"i\n";
}
int main()
```

```
{
complex c1, c2, c3, c4, c5, c6;
        cout<<"\n Enter first number";
        c1.getdata();
        cout<<"\n Enter second number";
        c2.getdata();
        c3=c1+c2;
        c4=c1-c2;
        c5=c1*c2;
        c6=c1/c2;
        cout<<"\n The first number is";
        c1.display();
        cout<<"\n The second number is";
        c2.display();
        cout<<"\n The addition is";
        c3.display();
        cout<<"\n The subtraction is";
        c4.display();
        cout<<"\n The multiplication is";
        c5.display();
        cout<<"\n The division is";
        c6.display();
}
```

OUTPUT:

Enter first no

Enter real part 3

Enter imaginary part 5

Enter second no

Enter real part2

Enter imaginary part 6

The first no is3+5i

The second no is2+6i

The addition is5+11i

The subtraction is1+-1i

The multiplication is-24+28i

The division is0.9+-0.2i

Experiment No: 2

AIM: Develop an object oriented program in C++ to create a database of student information system containing the following information: Name, Roll number, Class, division, Date of Birth, Blood group, 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 constructor, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation operators-new and delete as well as exception handling..

OBJECTIVES:

1. To learn the concept of constructor and destructor in C++.
2. To study the representation, implementation and applications of data structures.
3. To study implementation of data structures using OOP concepts.
4. To compare the benefits of static and dynamic data structures.

SOFTWARE USED:

Linux Operating Systems, GCC

CONCEPTS:

Class, functions, constructor, destructor, new operator, delete operator, friend class, inline function, this pointer.

THEORY:

1.Constructor:

Definition: A constructor is a member function of a class with the same name as that of its class name. A constructor is defined like other member functions of a class. It can be defined either inside or outside the class definition.

Constructors are used to initialise the data members of a class.

There are three types of constructors: default constructors, parameterised constructors and copy constructors.

i. Default constructor:

A default constructor is a constructor that accepts no parameters. When a user defined class does not contain an explicit constructor, the compiler automatically supplies a default constructor, having no arguments. A default constructor is invoked whenever an object of the class is created.

Syntax:

```
class_name()
{… initialisation…}
```

Example:

```
class abc
{
public:
abc(){…}
};
int main()
{
abc obj; //default constructor called
return 0;
}
```

ii. Parameterised constructor:

A constructor that accepts parameters is called a parameterised constructor. A parameterised constructor is invoked only when the object created specifies the arguments when it is declared.

Syntax:

```
class_name(parameter list)
{…initialisation…}
```

Example:

```
class abc
{
```

```
public:
abc(int a, int b)
{…}
};
void main()
{
abc obj(5,4);
}
```

2.Destructor:

Definition: A destructor is also a member function whose name is the same as the class name but it is preceded by a tilde (~). A destructor takes no arguments, and no return types can be specified for it- not even void. It is called automatically by the complier whenever an object is destroyed. A destructor cleans up the storage (memory area of an object) that is no longer accessible.

Syntax:

```
~class_name()
{…}
```

Example:

```
class abc
{
public:
~abc(){cout<<"Destructor invoked";}
};
void main()
{
abc obj1;
…. //automatically obj1 is destructed using destructor ~abc()
}
```

3.'new' operator:

Definition: The new operator is used at the time of dynamic memory allocation and object construction.

Syntax:

```
class_name *object_name=new class_name
```

data_type *variable_name=new data_type

Example:

```
calc *obj=new calc;
int *a[]=new int;
```

4.'delete' operator:

Definition: The delete operator is used at the time of object deletion to free up memory space occupied by objects of a class which are no longer required in the class.

Syntax:

delete object_name;

Example:

delete obj;

5.this pointer:

Definition: The 'this' pointer is used in a function or a constructor to refer to a data member of a class having the same name as a local variable of the function or constructor. The function and constructor have to be defined inside the class to be able to access private data members of a class.

Syntax:

this->variable_name=variable_name

Example:

```
class hello
{ int a;
public:
int access();
};
int hello::access()
{
int a=10;
this->a=a;
```

```
cout<<a;
return 0;
}
```

ALGORITHM:

1. Start.
2. Create a class data, having data members string name, address, dob, lic, cldiv, int roll, long int phone, char bg, static int c and member functions getdata(), show(), static member function getcount(), default constructor data(), parameterised constructor data(int, long int, string, string, string, string, string) and destructor ~data()
3. initialize variables to zero values in default constructor, i.e.

   name=”-“

   address=”-“

   dob=”-“

   lic=”-“

   roll=0

   phone=0

   bg=’-‘
4. create object of class data obj, which calls default constructor, using new operator
5. obj.show() will display values of default constructor
6. delete obj, which will call destructor ~data()
7. create object of class data obj1(23, 9822794182, ”Dhruvatara”, ”Kalyani Nagar”, ”22.02.97”, “SE A”, “AMIFN32”)
8. obj1.show() will display values of parameterised constructor
9. delete obj1, which will call destructor ~data()
10. Ask the user the size of their database and store in num
11. Declare data oba[num]
12. Create a for loop from i=0 to i<num and call oba[i].getdata() to accept database.
13. Create a loop from i=0 to i<num and call oba[i].show() to display database
14. Call getcount() to display number of objects created
15. Check for exception handling concept

16. Stop.

INPUT:

Enter personal data like name, blood group, address, date of birth, class, division, license number, policy number, roll number, phone etc.

OUTPUT:

We get output in default and parameterize constructor, destructor.

CONCLUSION: -

Thus we have learned how to use basic concepts of object oriented programming like allocation of memory using new and delete, constructors, static members, etc.

Experiment No: 3

AIM: Imagine a publishing company which does marketing for book and audio cassette versions. Create class publications that sores 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 members' values with zero values

OBJECTIVES:

1. To learn the concept of constructor and destructor in C++.
2. To study the representation, implementation and applications of datastructures.
3. To study implementation of data structures using OOP concepts.
4. To compare the benefits of static and dynamic data structures.

SOFTWARE USED:

Linux Operating Systems, GCC

CONCEPT:

1. Member Functions.

2. New operator in constructor

THEORY:

Member Functions: -

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.

```
class Box
{
  public:
    double length;      // Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box

    double getVolume(void)
```

```
    {
      return length * breadth * height;
    }
};
```

If you like you can define same function outside the class using **scope resolution operator, ::** as follows:

```
double Box::getVolume(void)
{
  return length * breadth * height;
}
```

Let us take previously defined class to access the members of the class using a member function instead of directly accessing them:

```
class Box

{

  public:

    double length;         // Length of a box

    double breadth;        // Breadth of a box

    double height;         // Height of a box

    double getVolume(void);// Returns box volume

};
```

Member functions can be defined within the class definition or separately using **scope resolution operator, ::**. Defining a member function within the class definition declares the function **inline**, even if you do not use the inline specifier. So either you can define **Volume()** function as below:

```
class Box
```

Here, only important point is that you would have to use class name just before :: operator. A member function will be called using a dot operator (**.**) on an object where it will manipulate data related to that object only as follows:

```
Box myBox;          // Create an object
myBox.getVolume();  // Call member function for the object
```

Let us put above concepts to set and get the value of different class members in a class:

```
#include <iostream>
using namespace std;
class Box
{
  public:
    double length;        // Length of a box
    double breadth;       // Breadth of a box
    double height;        // Height of a box

    // Member functions declaration
    double getVolume(void);
    void setLength( double len );
    void setBreadth( double bre );
    void setHeight( double hei );
};
// Member functions definitions
double Box::getVolume(void)
{
  return length * breadth * height;
}

void Box::setLength( double len )
{
  length = len;
}
void Box::setBreadth( double bre )
{
  breadth = bre;
}
```

```
void Box::setHeight( double hei )
{
   height = hei;
}

// Main function for the program
int main( )
{
  Box Box1;               // Declare Box1 of type Box
  Box Box2;               // Declare Box2 of type Box
  double volume = 0.0;     // Store the volume of a box here
  // box 1 specification
  Box1.setLength(6.0);
  Box1.setBreadth(7.0);
  Box1.setHeight(5.0);

  // box 2 specification
  Box2.setLength(12.0);
  Box2.setBreadth(13.0);
  Box2.setHeight(10.0);

  // volume of box 1
  volume = Box1.getVolume();
  cout <<"Volume of Box1 : "<< volume <<endl;

  // volume of box 2
  volume = Box2.getVolume();
  cout <<"Volume of Box2 : "<< volume <<endl;
  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

ALGORITHM:

1. Start
2. Create class books
3. Initialize data members variables publisher, title, author, price, stock, num, cost, cont and member functions accept(), display() and search().
4. In main() initialize array of objects, ti, auth, cont, k, num, i.
5. Display menu and accordingly take input from user using accept function of the objects and display using display function.
6. Search for the book user wants by calling search() and display price.
7. Stop.

INPUT:

Details about books such as author, title, price, publisher and stock position

OUTPUT:

Display required book's details.

Search required book if author name and title is given

CONCLUSION:

Hence, we have successfully studied concept of new and delete operator.

Experiment No: 4

AIM: 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.

OBJECTIVES:

- To understand the concept of file handling
- To understand the inbuilt file handling functions.

SOFTWARE USED:
Linux Operating Systems, GCC

THEORY:

So far, we have been using the iostream standard library, which provides cin and cout methods for reading from standard input and writing to standard output respectively.

Operations : Creating a file and output some data

Sr.No	Data Type & Description
1	**ofstream** This data type represents the output file stream and is used to create files and to write information to files.
2	**ifstream** This data type represents the input file stream and is used to read information from files.
3	**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.

Opening a File

A file must be opened before you can read from it or write to it. Either **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.

Sr.No	Mode Flag & Description
1	**ios::app** Append mode. All output to that file to be appended to the end.
2	**ios::ate** Open a file for output and move the read/write control to the end of the file.
3	**ios::in** Open a file for reading.
4	**ios::out** Open a file for writing.
5	**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**ing them together. For example if you want to open a file in write mode and want to truncate it in case that already exists, following will be the syntax −

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

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

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

Closing a File

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

ALGORITHM:

1. Start.
2. Create a file pointer object in main program.
3. With pointer object open a file in writing mode and write the content into it.
4. Close a file.
5. With pointer object open a file in reading mode and read the content from the file.
6. Close a file.
7. Stop.

INPUT:

Text file

OUTPUT:1) Writing the content into file

2) reading content from the file.

CONCLUSION:

Thus we studied concepts of file handling and its operation to perform reading the content from the file and writing the content into the file.

Experiment No: 5

AIM: Write a function template selection sort. Write a program that inputs, sorts and outputs an integer array and a float array.

OBJECTIVE:

1. To understand the concept of template.
2. To understand the uses of template.

SOFTWARE USED:

Linux Operating Systems, GCC

THEORY:

C++ templates provide a way to re-use source code. C++ provides two kinds of templates:

1. Class templates
2. Function templates.

Use function templates to write generic functions that can be used with arbitrary types. For example, one can write searching and sorting routines which can be used with any arbitrary type. C++ Function templates are those functions which can handle different data types without separate code for each of them. For a similar operation on several kinds of data types, a programmer need not write different versions by overloading a function. It is enough if he writes a C++ template based function. This will take care of all the data types.

Class Templates

A class template definition looks like a regular class definition, except it is prefixed by the keyword template. Once code is written as a C++ class template, it can support all data types.
Declaration of C++ class template should start with the keyword template. A parameter should be included inside angular brackets. The parameter inside the angular brackets, can be either the keyword class or typename. This is followed by the class body declaration with the member data and member functions.

```
template <class T>

class class_name
{
      // class member specification

// with anonymous type T wherever appropriate

};
```

T is a type parameter and it can be any type.

Defining member functions - C++ Class Templates:
If the functions are defined outside the template class body, they should always be defined with the full template definition. Other conventions of writing the function in C++ class templates are the same as writing normal c++ functions.

Advantages of C++ Class Templates:

- One C++ Class Template can handle different types of parameters.
- Compiler generates classes for only the used types. If the template is instantiated for int type, compiler generates only an int version for the c++ template class.
- Templates reduce the effort on coding for different data types to a single set of code.
- Testing and debugging efforts are reduced.

Function templates

There is lot of occasions, where we might need to write the same functions for different data types. A favorite example can be addition of two variables. The variable can be integer, float or double. The requirement will be to return the corresponding return type based on the input type. If we start writing one function for each of the data type, then we will end up with 4 to 5 different functions, which can be a night mare for maintenance. C++ templates come to our

rescue in such situations. When we use C++ function templates, only one function signature needs to be created. The C++ compiler will automatically generate the required functions for handling the individual data types.

Eg : Add function.

```
Template <class T> T Add(Ta, Tb)

//C++functiontemplatesample
 {
  returna+b;
 }
```

This c++ function template definition will be enough. Now when the integer version of the function, the compiler generates an Add function compatible for integer data type and if float is called it generates float type and so on. Here T is the type name. This is dynamically determined by the compiler according to the parameter passed. The keyword *class* means, the parameter can be of any type. It can even be a class.

ALGORITHM:

template <class t1, class t2, class t3> void ADD (t1 *C,t2 *A,t3 *B)

Addition: ADD (A,B,C,M,N)

Let A be an M*N matrix array and Let B be an M*N matrix array.

This algorithm stores the A+B in an M*N matrix array C.

1. Repeat Steps 2 to 3 for I = 1 to M:
2. Repeat Step 3 for J = 1 to N:
3. Set C[I,J] = A[I,J] + B[I,J]
4. Return.

template <class t1, class t2, class t3> void SUB (t1 *C,t2 *A,t3 *B)

Subtraction : SUB (A,B,C,M,N)

Let A be an M*N matrix array and Let B be an M*N matrix array.

This algorithm stores the A-B in an M*N matrix array C.

1. Repeat Steps 2 to 3 for I = 1 to M:

2. Repeat Step 3 for J = 1 to N:
3. Set C[I,J] = A[I,J] - B[I,J]
4. Return.

template <class t1, class t2, class t3> void MATMUL (t1 *C,t2 *A,t3 *B)

Multiplication: MATMULT(A,B,C,M,N,L)

Let A be an M*N matrix array and Let B be an N*L matrix array.

This algorithm stores the A*B in an M*L matrix array C.

1. Repeat Steps 2 to 5 for I = 1 to M:
2. Repeat Steps 3 to 5 for J = 1 to L:
3. Set C[I][J] = 0. [Initializes C[I][J]]
4. Repeat step 5 for K = 1 to N:
5. C[I,J] = C[I,J] + A[I,K] * B[K,J].
6. Return

INPUT:

Two matrices with different data types as integers and floating point values

OUTPUT:

Integer sort with sorted array and float sort with sorted array.

CONCLUSION:

In this practical we have implemented template. We have used concept of templates.

Hence we have successfully studied all the concept of templates in object oriented programming language.

PROGRAM:

```
//selection using function templates

#include <iostream>
using namespace std;

template <class T>
void sort()
{
        int i, j;
        T temp;
        T n[5];
        cout<<"\n Enter five numbers : ";
        for(i=0;i<5;i++)
        {
                cin>>n[i];
        }
        for(i=0;i<4;i++)
        {
                for(j=i;j<5;j++)
                {
                        if(n[i]>n[j])
                        {
                                temp=n[i];
                                n[i]=n[j];
                                n[j]=temp;
                        }
                }
        }
        cout<<"\n The array in the sorted order is : "<<endl;
        for(i=0;i<5;i++)
        {
                cout<<"\t"<<n[i];
        }
}
int main()
{
        int choice;
        char ans;
        do
        {
                cout<<"\n 1. Integer  sort. \n 2. Float sort.";
                cout<<"\n Enter the input you want to sort : ";
                cin>>choice;
```

```
                switch(choice)
                {
                        case 1 : sort<int>();
                                 break;
                        case 2 : sort<float>();
                                 break;
                        case 3 : cout<<"\n Invalid choice.";
                                 break;
                }
                cout<<"\n Do u wish to continue (Y/N)?";
                cin>>ans;
        }while(ans=='Y' || ans=='y');
        return 0;
}
```

OUTPUT:

```
1. Integer  sort.
2. Float sort.
Enter the input you want to sort : 1

Enter five numbers : 100 98 110 20 65

The array in the sorted order is :
        20      65      98      100     110
Do u wish to continue (Y/N)? y

1. Integer  sort.
2. Float sort.
Enter the input you want to sort : 2

Enter five numbers : 66.5 99.1 12.4 26.9 1.50

The array in the sorted order is :
        1.5     12.4    26.9    66.5    99.1
Do u wish to continue (Y/N)? n
```

Experiment No: 6

AIM: Write C++ program using STL for Sorting and searching with user-defined records such as Person Record (Name, birth date, telephone no), item record (item code, item name, quantity and cost.

OBJECTIVES:

1. To learn the concept of constructor and destructor in C++.
2. To study the representation, implementation and applications of datastructures.
3. To study implementation of data structures using OOP concepts.
4. To compare the benefits of static and dynamic data structures.

SOFTWARE USED:

Linux Operating Systems, GCC

THEORY:

Searching

Consider searching for a given value **v** in an array of size N. There are 2 basic approaches: **sequential search** and **binary search**.

Sequential Search

Sequential search involves looking at each value in turn (i.e., start with the value in array[0], then array[1], etc). The algorithm quits and returns true if the current value is v; it quits and returns false if it has looked at all of the values in the array without finding v. Here's the code:

```
public static boolean sequentialSearch(Object[] A, Object v) {
for (int k = 0; k < A.length; k++) {
if (A[k].equals(v)) return true;
}
return false;
```

```
}
```

If the values are in **sorted** order, then the algorithm can sometimes quit and return false without having to look at all of the values in the array: v is not in the array if the current value is **greater** than v. Here's the code for this version:

```
public static boolean sortedSequentialSearch(Comparable[] A, Comparable v) {
// precondition: A is sorted (in ascending order)
for (int k = 0; k < A.length; k++) {
if (A[k].equals(v)) return true;
if (A[k].compareTo(v) > 0) return false;
}
return false;
}
```

The worst-case time for a sequential search is always O(N).

Binary Search

When the values are in sorted order, a better approach than the one given above is to use **binary search**. The algorithm for binary search starts by looking at the middle item x. If x is equal to v, it quits and returns true. Otherwise, it uses the relative ordering of x and v to eliminate half of the array (if v is less than x, then it can't be stored to the right of x in the array; similarly, if it is greater than x, it can't be stored to the left of x). Once half of the array has been eliminated, the algorithm starts again by looking at the middle item in the remaining half. It quits when it finds v or when the entire array has been eliminated.

Here's the code for binary search:

```
public static boolean binarySearch(Comparable[] A, Comparable v) {
// precondition: A is sorted (in ascending order)
return binarySearchAux(A, 0, A.length - 1, v);
}
```

```
private static boolean binarySearchAux(Comparable[] A, int low, int high, int v) {
// precondition: A is sorted (in ascending order)
// postcondition: return true iff v is in an element of A in the range
//              A[low] to A[high]
if (low > high) return false;
int middle = (low + high) / 2;
if (A[middle].equals(v)) return true;
if (v.compareTo(A[middle]) < 0) {
// recursively search the left part of the array
return binarySearchAux(A, low, middle-1, v);
}
else {
// recursively search the right part of the array
return binarySearchAux(A, middle+1, high, v);
}
}
```

The worst-case time for binary search is proportional to $\log_2 N$: the number of times N can be divided in half before there is nothing left. Using big-O notation, this is O(log N). Note that binary search in an array is basically the same as doing a lookup in a perfectly balanced binary-search tree (the root of a balanced BST is the middle value). In both cases, if the current value is not the one we're looking for, we can eliminate half of the remaining values.

Sorting

Sorting is nothing but storage of data in sorted order, it can be in ascending or descending order. The term Sorting comes into picture with the term Searching. There are so many things in our real life that we need to search, like a particular record in database, roll numbers in merit list, a particular telephone number, any particular page in a book etc.

Sorting arranges data in a sequence which makes searching easier. Every record which is going to be sorted will contain one key. Based on the key the record will be sorted. For example, suppose we have a record of students, every such record will have the following data:

- Roll No.
- Name
- Age
- Class

Here Student roll no. can be taken as key for sorting the records in ascending or descending order. Now suppose we have to search a Student with roll no. 15, we don't need to search the complete record we will simply search between the Students with roll no. 10 to 20.

Sorting Efficiency

There are many techniques for sorting. Implementation of particular sorting technique depends upon situation. Sorting techniques mainly depends on two parameters. First parameter is the execution time of program, which means time taken for execution of program. Second is the space, which means space taken by the program.

Types of Sorting Techniques

There are many types of Sorting techniques, differentiated by their efficiency and space requirements. Following are some sorting techniques which we will be covering in next sections.

- Bubble Sort
- Insertion Sort
- Selection Sort
- Quick Sort
- Merge Sort
- Heap Sort

Insertion Sorting

It is a simple Sorting algorithm which sorts the array by shifting elements one by one. Following are some of the important characteristics of Insertion Sort.

- It has one of the simplest implementation
- It is efficient for smaller data sets, but very inefficient for larger lists.

- Insertion Sort is adaptive, that means it reduces its total number of steps if given a partially sorted list, hence it increases its efficiency.
- It is better than Selection Sort and Bubble Sort algorithms.
- Its space complexity is less, like Bubble Sorting, inerstion sort also requires a single additional memory space.
- It is **Stable**, as it does not change the relative order of elements with equal keys

Sorting using Insertion Sort Algorithm

```
int a[6] = {5, 1, 6, 2, 4, 3};
int i, j, key;
for(i=1; i<6; i++)
{
key = a[i];
j = i-1;
while(j>=0 && key < a[j])
{
a[j+1] = a[j];
j--;
}
a[j+1] = key;
}
```

Selection Sorting

Selection sorting is conceptually the most simplest sorting algorithm. This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continues in this way until the entire array is sorted.

How Selection Sorting Works

In the first pass, the smallest element found is 1, so it is placed at the first position, then leaving first element, smallest element is searched from the rest of the elements, 3 is the smallest, so it is

then placed at the second position. Then we leave 1 nad 3, from the rest of the elements, we search for the smallest and put it at third position and keep doing this, until array is sorted.

Sorting using Selection Sort Algorithm

```
void selectionSort(int a[], int size)
{
int i, j, min, temp;
for(i=0; i < size-1; i++ )
{
min = i;   //setting min as i
for(j=i+1; j < size; j++)
{
if(a[j] < a[min])   //if element at j is less than element at min position
{
min = j;    //then set min as j
}
}
temp = a[i];
a[i] = a[min];
a[min] = temp;
}
}
```

ALGORITHM:

1. Start
2. Enter your choice
3. In that choice we have enter either personal record or item record
4. First we have to choice personal record
5. In that there is again they give some choices like enter details, display, search and then sort
6. We have to enter details in that enter number of member present in record
7. Then display it and sort it
8. Again search for some record which is present in that records
9. Again this steps repeat for entering item record
10. And finally we give output
11. Stop

INPUT:

Enter name, date of birth and phone number for some records in personal record and in item record enter item name, item code, cost, and quantity for some records.

OUTPUT:

We give output of personal record as display that record then search for one record and also sort it like that we give item record.

CONCLUSION:

We have studied successfully the concepts of searching and sorting in object oriented programming language.

PROGRAM:

```
#include <iostream>
#include<list>
using namespace std;
class record
{
        list<string>name,dob,phone,ni;
        list<string>::iterator it1,it2,it3,j,k,l,c,n;
        list<string>code;
        list<int>number;
        list<float>cost;
list<int>::iterator no,j1;
list<float>::iterator f,i;
public:
        void getp();
        void display();
        void searchp(string);
        void sortp();
        void checkempty();
        void getlist();
        void displayit();
void searchlist();
void sortitem();

};
void record::getp()
{
        int count;
        string n,d,p;
        cout<<"Enter the number of members in record:"<<endl;
        cin>>count;
        for(int i=1;i<=count;i++)
        {
                cout<<"Enter name:"<<endl;
                cin>>n;
                name.push_back(n);
                cout<<"Enter date of birth:"<<endl;
                cin>>d;
                dob.push_back(d);
                cout<<"Enter phone number:"<<endl;
                cin>>p;
                phone.push_back(p);
        }
}
void record::searchp(string data)
```

```
{
        int flag=0;
        it1=name.begin();
        it2=dob.begin();
        it3=phone.begin();
        while(it1!=name.end()&&it2!=dob.end()&&it3!=phone.end())
        {
                if(*it1==data)
                {
                        cout<<"Record found!"<<endl;
                        cout<<"Corresponding D.O.B: "<<*it2<<endl;
                        cout<<"Corresponding phone number: "<<*it3<<endl;
                        flag=1;
                        break;
                }
                if(*it2==data)
                {
                        cout<<"Record found!"<<endl;
                        cout<<"Corresponding name "<<*it1<<endl;
                        cout<<"Corresponding phone number: "<<*it3<<endl;
                        flag=1;
                        break;
                }
                if(*it3==data)
                {
                        cout<<"Record found!"<<endl;
                        cout<<"Corresponding name: "<<*it1<<endl;
                        cout<<"Corresponding D.O.B: "<<*it2<<endl;
                        flag=1;
                        break;
                }
                it1++;
                it2++;
                it3++;
        }
        if(flag==0)
                cout<<"Record not found."<<endl;
}
void record:: display()
{
        it1=name.begin();
        it2=dob.begin();
        it3=phone.begin();
        while(it1!=name.end())
        {
                cout<<*it1<<"\t"<<*it2<<"\t"<<*it3<<endl;
                it1++;
```

```
                it2++;
                it3++;
        }
}
void record::sortp()
{
string temp;
it1=name.begin();
        it2=dob.begin();
        it3=phone.begin();
        j=it1;
        k=it2;
        l=it3;
        j++;
        k++;
        l++;
        while(it1!=name.end())
{
while(j!=name.end())
{
if(*it1>*j)
{
temp=*it1;
*it1=*j;
*j=temp;
temp=*it2;
*it2=*k;
*k=temp;
temp=*it3;
*it3=*l;
*l=temp;
}
j++;
k++;
l++;
}
it1++;
it2++;
it3++;
}
}
void record::getlist()
{
cout<<"Enter the number of items:"<<endl;
int c,no;
string n;
float f;
```

```
cin>>c;
for(int i=1;i<=c;i++)
{
cout<<"Enter item name:"<<endl;
cin>>n;
ni.push_back(n);
cout<<"Enter item code:"<<endl;
cin>>n;
code.push_back(n);
cout<<"Enter cost:"<<endl;
cin>>f;
cost.push_back(f);
cout<<"Enter the quantity:"<<endl;
cin>>no;
number.push_back(no);
}
}
void record::displayit()
{
c=code.begin();
n=ni.begin();
no=number.begin();
f=cost.begin();
while(c!=code.end())
{
cout<<*c<<"\t"<<*n<<"\t"<<*no<<"\t"<<*f<<endl;
c++;
n++;
no++;
f++;
}
}
void record::sortitem()
{
string temp;
int tempno;
float tempf;
c=code.begin();
n=ni.begin();
no=number.begin();
f=cost.begin();
i=f;
j1=no;
k=c;
l=n;
i++;
j1++;
```

```
k++;
l++;
while(f!=cost.end())
{
while(i!=cost.end())
{
if(*f>*i)
{
tempf=*f;
*f=*i;
*i=tempf;

temp=*n;
*n=*l;
*l=temp;

temp=*c;
*c=*k;
*k=temp;

tempno=*no;
*no=*j1;
*j1=tempno;
}
i++;
j1++;
k++;
l++;
}
f++;
n++;
no++;
c++;
}
}
void record::searchlist()
{
        string key;
        cout<<"Enter the item code:"<<endl;
        cin>>key;
        c=code.begin();
        n=ni.begin();
        no=number.begin();
        f=cost.begin();
        while(c!=code.end())
        {
                if(key==*c)
```

```
                {
                        cout<<"Item available!"<<endl;
                        cout<<"Item name: "<<*n<<endl;
                        cout<<"Item quantity: "<<*no<<endl;
                        cout<<"Item cost: "<<*f<<endl;
                }
                c++;
                n++;
                no++;
                f++;
        }
}
int main()
{
        record obj;
        string key;
        int ch,chr;
        char x='y';
        do
        {
        cout<<"1. Personal record\n2. Item record\nEnter choice:\n";
        cin>>ch;
        do
        {
        if(ch==1)
        {
                cout<<"1. Enter details\n2. Display\n3. Search entry\n4. Sort records\nEnter
choice\n";
                cin>>chr;
                switch(chr)
                {
                case 1:
                        obj.getp();
                        obj.display();
                        break;
                case 2:
                        obj.display();
                        break;
                case 3:
                        cout<<"Enter either name, d.o.b or phone number you want to find\n";
                        cin>>key;
                        obj.searchp(key);
                        break;
                case 4:
                        obj.sortp();
                        obj.display();
                        break;
```

```
                default:
                        cout<<"Wrong choice"<<endl;
                }
        }
        else if(ch==2)
        {
                cout<<"1. Enter details\n2. Display\n3. Search entry\n4. Sort records\nEnter
choice\n";
                cin>>chr;
                switch(chr)
                {
                case 1:
                        obj.getlist();
                        obj.displayit();
                        break;
                case 2:
                        obj.displayit();
                        break;
                case 3:
                        obj.searchlist();
                        break;
                case 4:
                        obj.sortitem();
                        obj.displayit();
                        break;
                default:
                        cout<<"Wrong choice"<<endl;
                }
        }
        else
        {
                cout<<"Wrong choice"<<endl;
                break;
        }
        cout<<"Do you wish to continue? Y or N\n";
        cin>>x;
        }while(x=='y'||x=='Y');
        cout<<"Do you wish to select another type of record? Y or N\n";
        cin>>x;
        }while(x=='y'||x=='Y');
        return 0;
}
```

OUTPUT:

1. Personal record

```
2. Item record
Enter choice:
1
1. Enter details
2. Display
3. Search entry
4. Sort records
Enter choice
1
Enter the number of members in record:
2
Enter name:
prachi
Enter date of birth:
24.05.97
Enter phone number:
8856088982
Enter name:
Rutuja
Enter date of birth:
20.02.95
Enter phone number:
9456838299
prachi  24.05.97        8856088982
Rutuja 20.02.95         9456838299
Do you wish to continue? Y or N
y
1. Enter details
2. Display
3. Search entry
4. Sort records
Enter choice
2
prachi  24.05.97        8856088982
Rutuja 20.02.95         9456838299
Do you wish to continue? Y or N
y
1. Enter details
2. Display
3. Search entry
4. Sort records
Enter choice
3
Enter either name, d.o.b or phone number you want to find
prachi
```

```
Record found!
Corresponding D.O.B: 24.05.97
Corresponding phone number: 8856088982
Do you wish to continue? Y or N
y
1. Enter details
2. Display
3. Search entry
4. Sort records
Enter choice
4
Rutuja 20.02.95        9456838299
prachi  24.05.97        8856088982
Do you wish to continue? Y or N
n
Do you wish to select another type of record? Y or N
y
1. Personal record
2. Item record
Enter choice:
2
1. Enter details
2. Display
3. Search entry
4. Sort records
Enter choice
1
Enter the number of items:
3
Enter item name:
books
Enter item code:
c1053
Enter cost:
1000
Enter the quantity:
4
Enter item name:
pen
Enter item code:
E237
Enter cost:
200
Enter the quantity:
20
```

```
Enter item name:
bag
Enter item code:
A901
Enter cost:
450
Enter the quantity:
1
c1053  books  4      1000
E237   pen    20     200
A901   bag    1      450
Do you wish to continue? Y or N
y
1. Enter details
2. Display
3. Search entry
4. Sort records
Enter choice
2
c1053  books  4      1000
E237   pen    20     200
A901   bag    1      450
Do you wish to continue? Y or N
y
1. Enter details
2. Display
3. Search entry
4. Sort records
Enter choice
3
Enter the item code:
E237
Item available!
Item name: pen
Item quantity: 20
Item cost: 200
Do you wish to continue? Y or N
y
1. Enter details
2. Display
3. Search entry
4. Sort records
Enter choice
4
E237   pen    20     200
```

```
c1053  books  4       1000
A901   bag    1       450
Do you wish to continue? Y or N
n
Do you wish to select another type of record? Y or N
n
```

Experiment No:7

AIM: 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.

OBJECTIVES:

- To understand the concept of Standard template library function
- To understand the function related to map associative container.

SOFTWARE USED:

Linux Operating Systems, GCC

THEORY:

Maps 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.

Some basic functions associated with Map:
begin() – Returns an iterator to the first element in the map
end() – Returns an iterator to the theoretical element that follows last element in the map
size() – Returns the number of elements in the map
max_size() – Returns the maximum number of elements that the map can hold
empty() – Returns whether the map is empty
pair insert(keyvalue, mapvalue) – Adds a new element to the map
erase(iterator position) – Removes the element at the position pointed by the iterator
erase(const g)– Removes the key value 'g' from the map
clear() – Removes all the elements from the map

Creating objects:

Maps are associative containers that store elements formed by a combination of a key value and a mapped value, following a specific order.

In a `map`, the key values are generally used to sort and uniquely identify the elements, while the mapped values store the content associated to this key. The types of key and mapped value may differ, and are grouped together in member type `value_type`, which is a pair type combining both:

```
typedef pair<const Key, T>value_type;
```

For inserting the values:

```
map<int, int> gquiz1;

   // insert elements in random order
   gquiz1.insert(pair<int, int>(1, 40));
   gquiz1.insert(pair<int, int>(2, 30));
   gquiz1.insert(pair<int, int>(3, 60));
   gquiz1.insert(pair<int, int>(4, 20));
   gquiz1.insert(pair<int, int>(5, 50));
   gquiz1.insert(pair<int, int>(6, 50));
   gquiz1.insert(pair<int, int>(7, 10));
```

ALGORITHM:

1. Start.
2. Create an class state with member function accept() and display().
3. Create an object of map global.
4. Menu for accepting data and to find the population for the particular state.
5. With object access the member function.
6.Repeat the process.
7. Stop.

INPUT:

State and population value from the user side.

OUTPUT:1) Displaying the values related to state with population

CONCLUSION:

Thus we studied concepts of standard template container Map and its operation related to it.