PVG's
College of Engineering
Nashik

Department of Computer Engineering

LABORATORY MANUAL

2020-2021

OBJECT ORIENTED PROGRAMMING

& COMPUTER GRAPHICS LABORATORY

SE-COMPUTER ENGINEERING

SEMESTER-I

Subject Code: 210245

TEACHING SCHEME
Lectures: 4 Hrs/Week
Practical: 2 Hrs/Week

EXAMINATION SCHEME

Theory:70 Marks

In-Sem:30 Marks

Practical:25 Marks

Term Work: 25 Marks

-: **Name of Faculty** :-

Prof. A.R.Jain
Asst. Professor, Department of Computer Engineering,

PUNE VIDYARTHI GRIHA'S

COLLEGE OF ENGINEERING, NASHIK.

INDEX

Sr. No	Title	Page No	Date of Conduction	Date of Submission	Sign
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 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.				
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

Assignment No.	Group A(2)

Title: Complex Number

Objectives: To understand operator overloading, constructor and data hiding.

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.

Outcomes: To understand operator overloading concept.

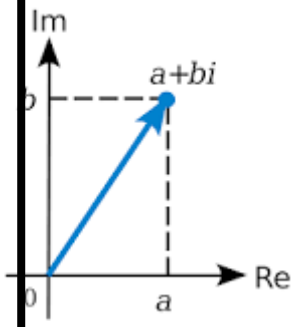
Software Requirements: ubuntu Operating system, Eclipse and Gedit.

Hardware Requirements: P4 System.

Prerequisite: Object oriented concepts.

2.1 Theory:

A **complex number** is a **number** that can be expressed in the form a + bi, where a and b are real **numbers** and i is the imaginary unit, that satisfies the equation $i^2 = -1$. In this expression, a is the real part and b is the imaginary part of the **complex number**.



Operations on Complex number:

1. Addition of Complex Numbers Addition of two complex numbers a + b i and c + d i is

defined asfollows.

(a + b i) + (c + d i) = (a + c) + (b + d) i

This is similar to grouping like terms: real parts are added to real parts and imaginary parts are added to imaginary parts.

2. Subtraction of Complex Numbers

The subtraction of two complex numbers a + b i and c + d i is defined as follows.

(a + b i) - (c + d i) = (a - b) + (b - d) i

3. Mulyiply Complex Numbers

The multiplication of two complex numbers a + b i and c + d i is defined as follows.

(a + b i)(c + d i) = (a c - b d) + (a d + bc) i

However you do not need to memorize the above definition as the multiplication can be carried out using properties similar to those of the real numbers and the added property $i^2 = -1$. (see the examplebelow)

2.2 OOP Features & Concepts used in this practical :

Operator Overloading: 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+(const Box&);

declares the addition operator that can be used to **add** two Box objects and returns final Box object.

Overloadable/Non-overloadableOperators:

Following is the list of operators which can be overloaded:

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

|=	*=	<<=	>>=	[ ]	( )
->	->*	new	new [ ]	delete	delete [ ]

Operator that are not overloaded are follows

scope operator - ::

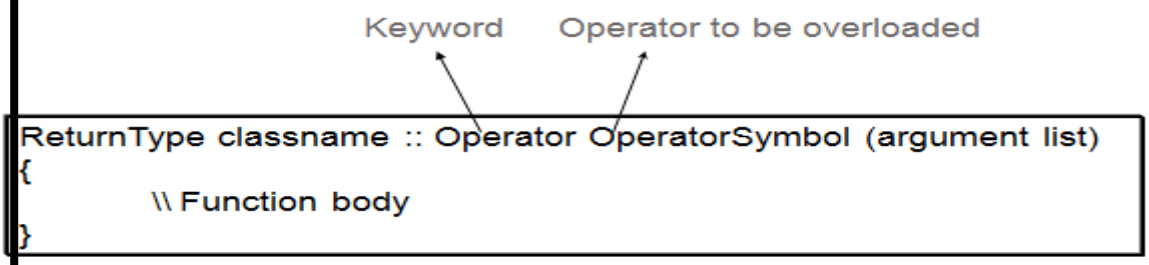
sizeof

member selector - .

member pointer selector - *

ternary operator - ?:

Syntax of operator overloading:



For calling function:

resultant = Object operatorsymbol Object;

resultant will vary depending upon operation done and return type of function.

2.3 Conclusions: Complex number operations are implemented with the operator overloading concept.

Assignment No.	Group A(6)

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

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.

Problem Statement: **:** 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 licence 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.

Outcomes:

Software Requirements: 32-bit Open source Linux or its derivative, Open Source C++ Programming tool like G++/GCC.

Hardware Requirements: Pentium IV, 512 RAM.

New Concepts:

6.1 Theory: 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 creates 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.
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 the another Object of Class .This is used for Copying the values of class object into an 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 Cant 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 ~tiled 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 :

4. A static function can have access to only other static members (fun or var) declared in the same class

5. A static function can be called using the class name instead of its object name

   Class_name :: function_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:

- ✓ They cannot access non static class member data using the member-selection operators (**.** or –>).

- ✓ They cannot be declared as **virtual**.
- ✓ They cannot have the same name as a non static function that has the same argument types.

Ex. Shall we give the example……

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 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.

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:

Data_type * ptr_var;

Eg. int * ptr;

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

We can initialize pointer variable as follows

int p, * ptr ; // declaration

ptr = & a ; // initialization

Pointers to objects:

Consider the following example

item P ; // where item is class & P is object

Similarly, we can define a pointer item_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.

Ex. 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.

6.2 Algorithm:

1. Start
2. Read personnel information such as Name, Roll number, Class, division, Date of Birth, Blood group, Contact address, telephone number, driving licence no. etc
3. Print all information from database.
4. Stop

6.3 Input : Personnel information such as Name, Roll number, Class, division, Date of Birth, Blood group, Contact address, telephone number, driving licence no. Etc

Output : Display personnel information from database.

6.4 Conclusions:

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.

Assignment 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
Class	S.E. (C.E.)
Roll No.	
Date	
Signature	

Assignment 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 DerivedClass : accessSpecifier BaseClass

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 becomes protected members of Derived class.

private:

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

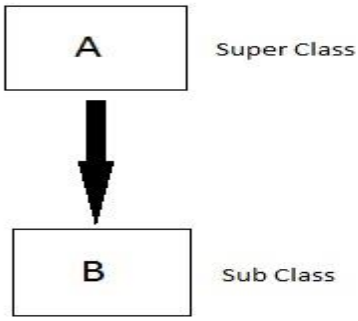
Types of Inheritance

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

1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance
5. 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 Inheritence

```
#include <iostream>
usingnamespacestd;
classVehicle
 {
  public:
```

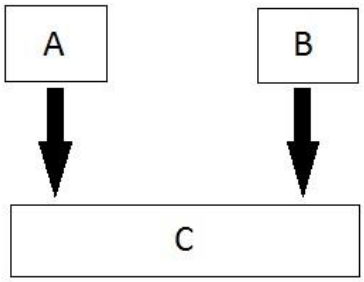
```
    Vehicle()
    {
      cout<< "This is a Vehicle"<<endl;
    }
};
classCar: publicVehicle
{

};
  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 Inheritence

```
#include <iostream>
usingnamespacestd;

classVehicle {
  public:
    Vehicle()
    {
      cout<< "This is a Vehicle"<<endl;
    }
};
```

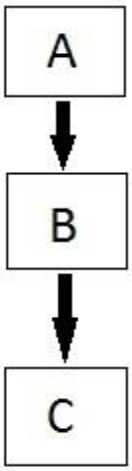
```
classFourWheeler {
  public:
    FourWheeler()
    {
      cout<< "This is a 4 wheeler Vehicle"<<endl;
    }
};

classCar: publicVehicle, publicFourWheeler
{

};

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;

classVehicle
{
  public:
```

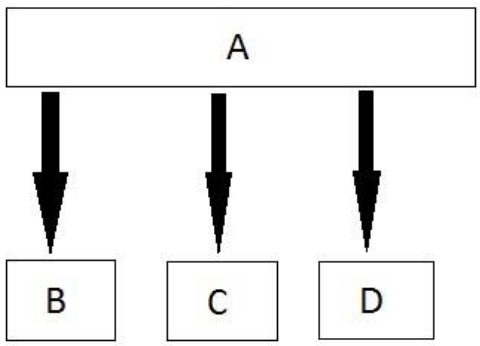
```
    Vehicle()
    {
      cout<< "This is a Vehicle"<<endl;
    }
};
Class fourWheeler : public Vehicle
{  public:
    fourWheeler()
    {
      cout<<"Objects with 4 wheels are vehicles"<<endl;
    }
};
Class Car: public fourWheeler{
   public:
     car()
     {
       cout<<"Car has 4 Wheels"<<endl;
     }
};

int main()
{

    Car obj;
    return0;
}
```

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 inherits from a single base class.




```
// Hierarchical Inheritance

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

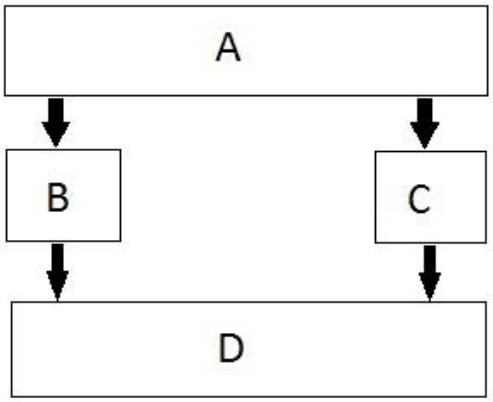
```
classVehicle
{
  public:
    Vehicle()
    {
      cout<< "This is a Vehicle"<<endl;
    }
};

classCar: publicVehicle
{

};
classBus: publicVehicle
{

};

intmain()
{

    Car obj1;
    Bus obj2;
    return0;
}
```

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;
```

```
classVehicle
{
  public:
    Vehicle()
    {
      cout<< "This is a Vehicle"<<endl;
    }
};

classFare
{
    public:
    Fare()
    {
        cout<<"Fare of Vehicle\n";
    }
};

classCar: publicVehicle
{

};
classBus: publicVehicle, publicFare
{

};

int main({

    Bus obj2;
    return0;
}
```

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 ot 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 iz. 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 roe 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 th exception thrown by the try blok. In catch block we take corrective action on throwing exception. If files are openend , we can take corrective action like closing file handles,closing database connetions,saving unsaved work ,etc.

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

SNYTAX:=

```
//normal program code
try{
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

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

Assignment No.	Group B(16)

Title: File Handling In C++

Objectives: To understand File Handling in C++ like opening the file,reading and writing the file,closing the file

Problem Statement:

Implement a program to open, read, write and to close the file in c++. Implement the following operations:

1. Open a file.
2. Read data from file.
3. Write data into file.
4. Close the file.

Outcomes: To understand The Concept Of File Handling.

Software Requirements: ubuntu Operating system, Eclipse and Gedit.

Hardware Requirements: P4 System.

Prerequisite: Object oriented concepts.

16.1 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

In order to create files we have to learn about File I/O i.e. how to write data into a file and how to read data from a file. We will start this section with an example of writing data to a file. We begin as before with the include statement for stdio.h, then define some variables for use in the example including a rather strange looking new type.

#include <stdio.h>

```
#include <stdio.h>
main( )
{
   FILE *fp;
   char stuff[25];
   int index;
   fp = fopen("TENLINES.TXT","w"); /* open for writing */
   strcpy(stuff,"This is an example line.");
   for (index = 1; index <= 10; index++)
   fprintf(fp,"%s Line number %d\n", stuff, index);
   fclose(fp); /* close the file before ending program */
}
```

2. **Reading from File**

When an r is used, the file is opened for reading, a w is used to indicate a file to be used for writing, and an a indicates that you desire to append additional data to the data already in an existing file. Most C compilers have other file attributes available; check your Reference Manual for details. Using the r indicates that the file is assumed to be a text file. Opening a file for reading requires that the file already exist. If it does not exist, the file pointer will be set to NULL and can be checked by the program.

```
#include <stdio.h>
void main()
{
 FILE *fopen(), *fp;
 int c;
 fp = fopen("prog.c","r");
 c = getc(fp) ;
 while (c!= EOF)
 {
     putchar(c);
    c = getc(fp);
 }
 fclose(fp);
}
```

✓ **Write In File**

When a file is opened for writing, it will be created if it does not already exist and it will be reset if it does, resulting in the deletion of any data already there. Using the w indicates that the file is assumed to be a text file.

```
#include <stdio.h>
int main()
{
 FILE *fp;
 file = fopen("file.txt","w");
/*Create a file and add text*/
  fprintf(fp,"%s","This is just an example :)"); /*writes data to the file*/
 fclose(fp); /*done!*/
 return 0;
}
```

4. Close the File

To close a file you simply use the function fclose with the file pointer in the parentheses. Actually, in this simple program, it is not necessary to close the file because the system will close all open files before returning to DOS, but it is good programming practice for you to close all files in spite of the fact that they will be closed automatically, because that would act as a reminder to you of what files are open at the end of each program.

```
fclose(fp);
```

16.3 Conclusions:

File operation is implemented using object oriented programming language features.

16.4 Assignment Questions:

1) what is file?

2) What is database?

3) What is file descriptor?

Assignment No.	Group B(18)

Title: Selection sort using function template.

Aim: Implement C++ program for Selection sort using function template

Objectives: To understand the concept of function template.

Theory:

Function templates

Function templates are special functions that can operate with *generic types*. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type. In C++ this can be achieved using *template parameters*. A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type. The format for declaring function templates with type parameters is:

template<classidentifier>function_declaration;
template<typenameidentifier>function_declaration

For example, to create a template function that returns the greater one of two objects we could use:

```
template <class myType>
myType GetMax (myType a, myType b)
{
    return (a>b?a:b);
}
```

Here we have created a template function with myType as its template parameter. This template parameter represents a type that has not yet been specified, but that can be used in the template function as if it were a regular type. As you can see, the function template GetMax returns the greater of two parameters of this still-undefined type.

Here in this program we have written a function for Selection sort and we have added a template tag before the function so that, the parameter will be of the data type Name. Everything is same except some variable data types. Take a look at the below program, you‟ll get a clear idea.
In the main function we are passing some predefined values into the Selection function by calling the function Selection(a,6) where a is the array containing integers and 6 is the size of array. After passing the values into the function we are displaying the sorted order. You can also rewrite the main function in a way that the user will enter the data and size of the array.

Selection sort

The algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

```
int i,j;
for (j = 0; j < n-1; j++)
{
        int iMin = j;
        for ( i = j+1; i < n; i++)
        {
                if (a[i] < a[iMin])
                {
                        iMin = i;
                }
        }
        if(iMin != j)
        {
                swap(a[j], a[iMin]);
        }
}
```

Selection sort is not difficult to analyze compared to other sorting algorithms since none of the loops depend on the data in the array. Selecting the lowest element requires scanning all n elements (this takes $n - 1$ comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining $n - 1$ elements and so on, for $(n - 1) + (n - 2) + ... + 2 + 1 = n(n - 1) / 2 \in \Theta(n2)$ comparisons. Each of these scans requires one swap for $n - 1$ elements (the final element is already in place).

Algorithm Selection Sort:

Selection(A, N)
Step 1 − Set MIN to location 0
Step 2 − Search the minimum element in the list
Step 3 − Swap with value at location MIN
Step 4 − Increment MIN to point to next element
Step 5 − Repeat until list is sorted

Algorithm:

1. Start
2. Read the numbers as integers or characters.
3. Sort them according to ascending order.
4. Print values as output.
5. Stop

Input:

Integer values, Float values.

Output:

Sorted order of integers as well as floats.

Conclusion:

Hence, we have successfully implemented Selection sort using function template.

GROUP –C

Assignment No.	6(GROUP C)
Title	Write C++ program using STL for sorting and searching user defined records such as 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 code, name, cost, quantity etc) using vector container.
Subject	Object Oriented Programming
Class	S.E. (C.E.)
Roll No.	
Date	
Signature	

Assignment 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 "first-class" 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 builtin function in C++ STL by the name of sort(). This function internally uses IntroSort. In more details it is implemented using hybrid of QuickSort,

HeapSort and InsertionSort.By default, it uses QuickSort but if QuickSort is doing unfair partitioning and taking more than N*logN time, it switches to HeapSort and when the array size becomes really small, it switches to InsertionSort.

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>
 usingnamespacestd;

voidshow(inta[])
{
   for(inti = 0; i < 10; ++i)
      cout<< a[i] << " ";
}

intmain()
{
   inta[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);

   return0;

}
```

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>

usingnamespacestd;

voidshow(inta[], intarraysize)
{
   for(inti = 0; i <arraysize; ++i)
      cout<< a[i] << " ";
}

intmain()
{
   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";

   return0;
}
```

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.Insert

2.Display

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

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 a 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?

Assignment 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
Class	S.E. (C.E.)
Roll No.	
Date	
Signature	

.

Assignment 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>
usingnamespacestd;
```

```
intmain()
{
   // 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];
   return0;
}
```

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

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 do declare a map?
4. Explain Associative mapping with example?

Reference-

OOPL Manual by

(STES’s NBN SINHGAD SCHOOL OF ENGINEERING Ambegaon (Bk), Pune

Department of Computer Engineering)