PUNE INSTITUTE OF COMPUTER TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

LABORATORY MANUAL

AY 2018-19

214448 : OBJECT ORIENTED PROGRAMMING LABORATORY

SE-IT

SEMESTER-I

TEACHING SCHEME	EXAMINATION SCHEME
Lectures: 3 Hrs/Week	Theory: 50 Marks
Practical: 2 Hrs/Week	On-Line: 50 Marks
Practical: 50 Marks	Term Work: 25 Marks

:::|| Compiled By ||:::

Mr. Abhinay G. Dhamankar (PICT,IT)

214448: OBJECT ORIENTED PROGRAMMING LABORATORY

Prerequisites :

Principles of Programming Languages, Fundamentals of Data Structures

Course Objectives :

1. Employ a problem-solving strategy to breakdown a complex problem into a series of simpler tasks.
2. Execute problem-solving actions appropriate to completing a variety of sub problems.
3. Apply analytical and logical thinking to extract facts from a problem description and determine how they relate to one another and to the problems to be solved.
4. Design and implement an object oriented solution to solve a real life problem.
5. Develop problem-solving and programming skills using OOP concept.

Course Outcomes : After studying this subject student should be able to

1. Develop and implement algorithms for solving simple problems using modular programming concept.
2. Abstract data and entities from the problem domain, build object models and design software solutions using object-oriented principles and strategies.
3. Discover, explore and apply tools and best practices in object-oriented programming.
4. Develop programs that appropriately utilize key object-oriented concepts
5. Create a data base using files

Note: While performing the assignments following care should be taken

1. Proper indenting, coding styles, commenting, naming conventions should be followed.
2. Avoid using global variables as far as possible
3. Faculty should prepare a lab manual including standard test cases & should be available for reference to students.
4. Student should submit term work in the form of a journal based on the above assignments.
5. Practical examination will be based on the term work. Questions will be asked during the examination to judge the understanding of the practical performed at the time of examination.
6. Candidate is expected to know the theory involved in the experiment.

LIST OF ASSIGNMNETS

LAB EXPT.NO	PROBLEM STATEMENT
1.	**Constructor and Destructor** WAP to generate Monthly Weather Report using **Constructor, Destructor**. Create a class named weather report that holds a daily weather report with data members day_of_month, hightemp, lowtemp,a mount_rain and amount_snow. Use different types of constructors to initialize the objects. Also include a function that prompts the user and sets values for each field so that you can override the default values. Write a menu driven program in C++ with options to enter data and generate monthly report that displays average of each attribute.
2.	**Dynamic Memory Allocation using new and delete operator** A Book shop maintains the inventory of books that are being sold at the shop. The list includes details such as title, author, publisher, price and available stock. Write a program in C++ which will have a class called books with suitable member functions for i. Add ii. Update iii. Search a book iv. Purchase a book (update the stock and display the total cost) v. Record number of successful/unsuccessful transactions **(use static data members to keep count of transactions)** Use **new** operator in **constructors** to allocate memory space required.
3.	**Operator Overloading** WAP to 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.
4.	**Inheritance** Design a base class with name, date of birth, blood group and another base class consisting of the data members such as height and weight. Design one more base class consisting of the insurance policy number and contact address. The derived class contains the data members' telephone numbers and driving license number. Write a menu driven program to carry out the following things: **i.** Build a master table **ii.** Display Record **iii.** Insert a record **iv.** Delete record **v.** Edit record **vi.** Search for a record
5.	**Multiple Inheritance** Create employee bio-data using following classes i. Personal record ii. Professional record

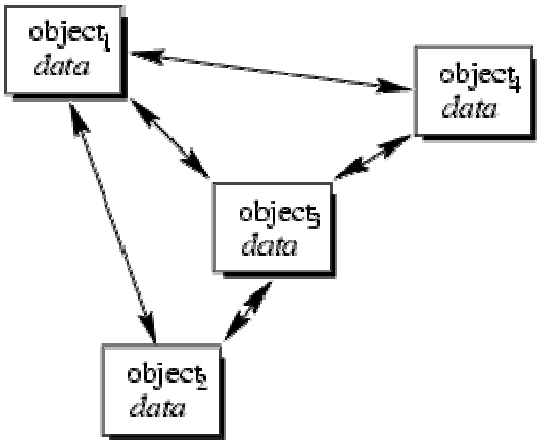
	iii. Academic record Assume appropriate data members and member function to accept required data & print bio-data. Create bio-data using **multiple inheritance** using C++.
6.	**Inheritance using Virtual function** Create a base class shape with two double type values and member functions to input the data and compute_area() for calculating area of figure. Derive two classes' triangle and rectangle. Make **compute_area()** as a **virtual function** and redefine this function in the derived class to suit their requirements. Write a program that accepts dimensions of triangle/rectangle and display calculated area.
7.	**Exception handling** Write a program in C++ which includes the code for following operations : i. A function to read two double type numbers from keyboard ii. A function to calculate the division of these two numbers iii. A **try block** to detect and throw an exception if the condition "divide-by-zero"occurs iv. Appropriate **catch block** to handle the **exceptions** thrown
8.	**Generic Programming (Template**) Write a program in C++ using **function/class template** to read two matrices of different data types such as integers and floating point values and perform simple arithmetic operations on these matrices separately and display it.
9.	**File Operation in C++** 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
10.	**File handling In C++(Sequential file)** Write a program in C++ to implement **sequential file** for students' database and perform following operations on it **i.** Create **ii.** Display **iii.** Add **iv.** Delete **v.** Modify

OOPL : Prerequisite

THEORY:

OOP : (Definition) is a programming language model organized around "objects" rather than "actions" and data rather than logic. The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling.

Object-oriented programming. Objects of the program interact by sending messages to each other.



WHY OBJECT ORIENTED APPROACH?

Major factor in the invention of Object-Oriented approach is to remove some of the flaws encountered with the procedural approach. In OOP, data is treated as a critical element and does not allow it to flow freely. It bounds data closely to the functions that operate on it and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects. A major advantage of OOP is code reusability.

Some important features of Object Oriented programming are as follows:

1. Emphasis on data rather than procedure
2. Programs are divided into Objects
3. Data is hidden and cannot be accessed by external functions
4. Objects can communicate with each other through functions
5. New data and functions can be easily added whenever necessary
6. Follows bottom-up approach

CONCEPTS OF OOP

- Objects
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism

OBJECTS

Objects are the basic run-time entities in an object-oriented system. Programming problem is analyzed in terms of objects and nature of communication between them. When a program is executed, objects interact with each other by sending messages. Different objects can also interact with each other without knowing the details of their data

CLASSES

A class is a collection of objects of similar type. It is a user defined data type. An object is called variable of class type. Once a class is defined, any number of objects can be created which belong to that class.

```
class student {
        // attributes:
           char name, DOB;

        //methods:
           Totol ();
           Avg();
           Display();
 };
```

DATA ABSTRACTION AND ENCAPSULATION

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes. Storing data and functions in a single unit (class) is encapsulation. Data cannot be accessible to the outside world and only those functions which are stored in the class can access it.

INHERITANCE

Inheritance is the process by which objects can acquire the properties of objects of other class. In OOP, inheritance provides reusability, like, adding additional features to an existing class without modifying it. This is achieved by deriving a new class from the existing one. The new class will have combined features of both the classes.

POLYMORPHISM

Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends on the data types used in the operation. Polymorphism is extensively used in implementing Inheritance.

ADVANTAGES OF OOP

Object-Oriented Programming has the following advantages over conventional approaches:

1. OOP provides a clear modular structure for programs which makes it good for defining abstract data types where implementation details are hidden and the unit has a clearly defined interface.
2. OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
3. OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer. This is particularly useful for developing graphical user interfaces.

Here are the few things you will notice about C++ which you cannot find in classis C Language.

1. Improved Comments Style in C++
2. The Keywords const & volatile
3. The scope operator
4. The iostream library
5. File stream operations
6. Variable definitions
7. Definition AND Declaration

One of the first object-oriented computer languages was called Smalltalk. C++ and Java are the most popular object-oriented languages today.

FAQ:

1. What are the main differences between object-oriented programming and the other programming techniques?
2. What are the added features in C++ ?
3. Benefits of OOP
4. What is Namespace?
5. Difference between “C structure” and “C++ structure”?
6. What is the difference between “calloc” and “malloc”?
7. What is difference between object and class?
8. Define class and object.
9. What is data abstraction and Data encapsulation?
10. What is difference between procedure oriented and object oriented languages?
11. Applications of OOP?

Subject Coordinator	**Head of Department (I.T)**
Mr. Abhinay G. Dhamankar	**Dr. B. A. Sonkamble**

Assignment No 1:
CONSTRUCTOR, DESTRUCTOR

AIM:

Create a class named weather report that holds a daily weather report with data member's day_of_month, hightemp, lowtemp, amount_rain and amount_snow. The constructor initializes the fields with default values: 99 for day_of_month, 999 for hightemp,-999 for lowtemp and 0 for amount_rain and amount_snow. Include a function that prompts the user and sets values for each field so that you can override the default values. Write a program that creates a monthly report.

OBJECTIVE:

Understand decomposition of a problem into a number of entities called object and then build data and function around these objects.

THEORY:

Constructors

A constructor is a special member function whose task is to initialize the objects of it's class. This is the first method that is run when an instance of a type is created. A constructor is invoked whenever an object of it's associated class is created. If a class contains a constructor, then an object created by that class will be initialized automatically. We pass data to the constructor by enclosing it in the parentheses following the class name when creating an object

Why you should define a constructor

Uninitialized member fields have *garbage* in them. This creates the possibility of a serious bug (e.g., an uninitialized pointer, illegal values, inconsistent values ...).

Characteristics of a constructor:

1. They should be declared in the public section
2. They are invoked automatically when the objects are created.
3. They cannot be inherited
4. They cannot be virtual

A constructor is similar to a function, but with the following differences.

1. No return type.
2. No return statement.

Types of Constructor

1. Default Constructor
2. Parameterized constructor
3. Copy constructor
4. Constructor with default arguments

INPUT:

day_of_month, hightemp, lowtemp, amount_rain and amount_snow

OUTPUT:

Display the result in following format:

	hightemp	lowtemp	amount_rain	amount_snow
1				
2				
.				
.				
---	---	---	---	---
Average				

FAQ:

1. What is dynamic initialization of a variable?
2. What are the different types of polymorphism?
3. Difference between a “assignment operator” and a “copy constructor”
4. What are the benefits of OOP?
5. Explain Destructor?
6. Explain copy constructor
7. Give example of a constructor with default values.
8. Give pictorial representation of object from your program.
9. Memory allocation for static members.
10. Memory allocation for array of objects.
11. Difference between class variables and static variables
12. Explain types of constructor and uses with example.
13. What is object and member function.
14. What is dynamic constructor .Explain with examples?
15. Difference between constructor and destructor.

PRACTICE ASSIGNMENTS:

1. Create a class BANK that holds Account holder information Accname, AccNo, Balance. The constructor initializes the fields with default values. Include a function that prompts the user and sets values for each field so that you can override the default values. Write a program that creates a report for 'N' number of Account Holder.

2. Create a class CRICKET that holds Name of Player, Century Scored, Runs Scored, Average, Economy, Wickets, and Catches etc. The constructor initializes the fields with default values. Include a function that prompts the user and sets values for each field so that you can override the default values. Write a program that creates a report for 'N' number of Cricketers.

3. Create a class named STUDENT that holds student information Stdname, StdRollno, Stdclass, Marks of subject, Percentage. The constructor initializes the fields with default values. Include a function that prompts the user and sets values for each field so that you can override the default values. Write a program that creates a report for 'N' number of students

Subject Coordinator
Mr. Abhinay G. Dhamankar

Head of Department (I.T)
Dr. B. A. Sonkamble

Assignment No 2:

DYNAMIC MEMORY ALLOCATION USING NEW AND DELETE OPERATOR

AIM:

A Book shop maintains the inventory of books that are being sold at the shop. The list includes details such as title, author, publisher, price and available stock.
Write a program in C++ which will have a class called books with suitable member functions for

i. Add
ii. Update
iii. Search a book
iv. Purchase a book (update the stock and display the total cost)
v. Record number of successful/unsuccessful transactions
(use static data members to keep count of transactions)

Use **new** operator in **constructors** to allocate memory space required.

OBJECTIVE:

Dynamic Memory allocation, new and delete operator ,static member and static member function this pointer, inline code.

THEORY:

Dynamic Memory allocation

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 obje static members have the same properties as global variables but they enjoy class scope. For that reason, and to avoid them to be declared several times, we can only include the prototype (its declaration) in the class declaration but not its definition (its initialization). In order to initialize a static data-member we must include a formal definition outside the class, in the global scope of this class to another. Because it is a unique variable value for all the

objects of the same class, it can be referred to as a member of any object of that class or even directly by the class name (of course this is only valid for static members.

A static member function has following properties

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

Class_name :: fun_name;

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

The following restrictions apply to such static functions:

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

E.g.

// static members in classes

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

Output

```
9
```

Friend functions:

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

Properties of friend function:

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

```
// friend functions
#include <iostream>
using namespace std;

class CRectangle {
  int width, height;
 public:
  void set_values (int, int);
  int area () {return (width * height);}
  friend CRectangle duplicate (CRectangle);
};

void CRectangle::set_values (int a, int b) {
 width = a;
 height = b;
}

CRectangle duplicate (CRectangle rectparam)
{
```

```
 CRectangle rectres;
 rectres.width = rectparam.width*2;
 rectres.height = rectparam.height*2;
 return (rectres);
}

int main () {
 CRectangle rect, rectb;
 rect.set_values (2,3);
 rectb = duplicate (rect);
 cout << rectb.area();
 return 0;
}
```

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

Friend classes

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

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

class CSquare;

class CRectangle {
  int width, height;
 public:
  int area ()
   {return (width * height);}
  void convert (CSquare a);
};
```

```
class CSquare {
 private:
  int side;
 public:
  void set_side (int a)
   {side=a;}
  friend class CRectangle;
};

void CRectangle::convert (CSquare a) {
 width = a.side;
 height = a.side;
}

int main () {
 CSquare sqr;
 CRectangle rect;
 sqr.set_side(4);
 rect.convert(sqr);
 cout << rect.area();
 return 0;
}
```

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

Pointers:

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

Declaration of pointer variable is in the following form :

```
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 a, *ptr;  // declaration
ptr = &a   //initialization
```

Pointers to objects:

Consider the following eg

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

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

Item *it_ptr ;

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

Eg item X;

item *ptr = &X;

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

we can access the member functions and data using pointers as follows

ptr->getdata();

ptr->show();

this pointer:

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

Important notes on this pointer:

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

INPUT:

Name, Date of Birth, Blood group, Height, Weight, Insurance Policy number, Contact address, telephone number, driving license no.

OUTPUT:

Display the result in following format:

	Name	DOB	.....	Driving License No
1				
2				
.				
.				
.				
n				

FAQ:

1. Explain new and delete operator in c++.
2. Differentiate between new and delete operator and malloc and dealloc in c
3. What is static Function?
4. What is friend function? State the adv& dis of using the friend function.
5. What is friend class? Explain with examples.
6. Explain with examples pointers to object.
7. What is this pointer? Explain with examples.
8. State the advantages of this pointer.
9. What are inline functions?
10. How do we declare member of a class static?

PRACTICE ASSIGNMENTS:

1. Develop an object oriented program in C++ to create a database of the Student information system containing the following information: Name, Date of Birth, Class, Marks, Contact address, telephone number, 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.

2. Develop an object oriented program in C++ to create a database of the Employee information system containing the following information: Name, Date of Birth, Empcode, Contact address, telephone number, 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.

Subject Coordinator	**Head of Department (I.T)**
Mr. Abhinay G. Dhamankar	**Dr. B. A. Sonkamble**

Assignment No 3:
OPERTOR OVERLOADING
COMPLEX NUMBERS OPERATOR OVERLOADING

AIM:

Write a C++ program to 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.

OBJECTIVE:

Understand decomposition of a problem into a number of entities called object and then build data and function around these objects.

THEORY:

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

Over loadable operators

```
+ - * / = < > += -= *= /= << >>
<<= >>= == != <= >= ++ -- % & ^ ! |
~ &= ^= |= && || %= []
```

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

INPUT:

Enter real and imaginary values for two complex numbers.

Example :

Complex No 1: Real Part : 5

Imaginary part : 4

Complex No 2: Real Part : 5

Imaginary part : 4

OUTPUT :

Display the result in following format:

Operation Performed:

Complex No: 1

Operation Symbol

Complex No: 2

RESULT

FAQ:

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

PRACTICE ASSIGNMENTS:

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. Overload << to accept two complex numbers

2. 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. Overload >> to display output of the above performed operations on complex numbers

3. Write a class to represent the vector (a series of float value). Include member function to create a vector and to modify. Overload operator * to multiply by a scalar value. Overload << to display scalar value.

4. Write a C++ program to perform following operations on the strings.
   a) > To Check two strings equal or not
   b) + Concatenation by friend functions
   c) - To delete a substring
   d) ~ To Get Reverse of String
   e) ^ To Get Palindrome strings

5. Write a C++ program to perform String operations using Operator Overloading
   a) "=" String Equality Check
   b) "<=" String Copy
   c) "<<" Display String (Using Friend Function)
   d) "*" String concatenation (e.g. S1="PICT", S2="SCTR", S3=S1*S2 i.e. s3="PICTSCTR").

Subject Coordinator
Mr. Abhinay G. Dhamankar

Head of Department (I.T)
Dr. B. A. Sonkamble

Assignment No 4:

INHERITANCE

AIM:

Design a base class with name, date of birth, blood group and another base class consisting of the data members such as height and weight.
Design one more base class consisting of the insurance policy number and contact address. The derived class contains the data members' telephone numbers and driving license number.
Write a menu driven program to carry out the following things:

- **i.** Build a master table
- **ii.** Display Record
- **iii.** Insert a record
- **iv.** Delete record
- **v.** Edit record
- **vi.** Search for a record

OBJECTIVE:

Understand the protected access specifier.

To learn concept of Inheritance

Different types of inheritance.

Single level, Multi level, Multiple, Hybrid.

Inheritance

Inheritance is the process by which objects can acquire the properties of objects of other class. In OOP, inheritance provides reusability, like, adding additional features to an existing class without modifying it. This is achieved by deriving a new class from the existing one. The new class will have combined features of both the classes.

Types of Inheritance

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Hybrid Inheritance

There are three types of class inheritance: public, private and protected

The protected access specifier is similar to private. Its only difference occurs in fact with inheritance. When a class inherits from another one, the members of the derived class can access the protected members inherited from the base class, but not its private members.

Difference between public, private and protected

- A member (either data member or member function) declared in a private section of a class can only be accessed by member functions and friends of that class
- A member (either data member or member function) declared in a protected section of a class can only be accessed by member functions and friends of that class, and by member functions and friends of derived classes
- A member (either data member or member function) declared in a public section of a class can be accessed by anyone

The following table indicates how the attributes are inherited in the three different types of inheritance:

Access specifiers in the base class			
	private	Public	protected
public inheritance	The member is private.	The member is public.	The member is protected.
private inheritance	The member is private.	The member is private.	The member is private.
protected inheritance	The member is private.	The member is protected.	The member is protected.

What a derived class inherits

- Every data member defined in the parent class (although such members¤ may not always be *accessible* in the derived class!)
- Every ordinary member function of the parent class (although such members¤ may not always be *accessible* in the derived class!)
- The same initial data layout as the base class

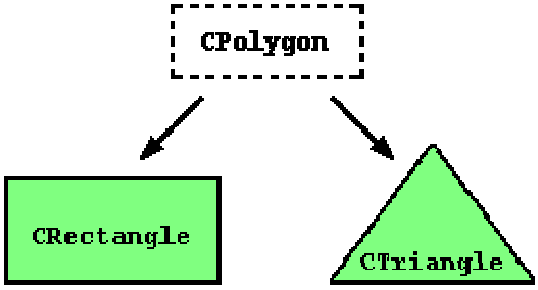
What a derived class doesn't inherit

- The base class's constructors and destructor
- The base class's assignment operator
- The base class's friends

What a derived class can add

- New data members
- New member functions
- New constructors and destructor
- New friends

E.g. Suppose that we want to declare a series of classes that describe polygons like our CRectangle, or like CTriangle. They have certain common properties, such as both can be described by means of only two sides: height and base. This could be represented in the world of classes with a class CPolygon from which we would derive the two other ones: CRectangle and CTriangle.



The class CPolygon would contain members that are common for both types of polygon. In our case: width and height. And CRectangle and CTriangle would be its derived classes, with specific features that are different from one type of polygon to the other.

Classes that are derived from others inherit all the accessible members of the base class. That means that if a base class includes a member A and we derive it to another class with another member called B, the derived class will contain both members A and B.

In order to derive a class from another, we use a colon (:) in the declaration of the derived class using the following format:

Class derived_class_name: public base_class_name
{ /*...*/ };

Where derived_class_name is the name of the derived class and base_class_name is the name of the class on which it is based. The public access specifier may be replaced by any one of the other access specifiers protected and private. This access specifier describes the minimum access level for the members that are inherited from the base class.

```
// derived classes
#include <iostream>
using namespace std;

class CPolygon {
 protected:
   int width, height;
```

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

class CRectangle: public CPolygon {
 public:
  int area ()
   { return (width * height); }
 };

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

int main () {
 CRectangle rect;
 CTriangle trgl;
 rect.set_values (4,5);
 trgl.set_values (4,5);
 cout << rect.area() << endl;
 cout << trgl.area() << endl;
 return 0;
}
```

The objects of the classes CRectangle and CTriangle each contain members inherited from CPolygon. These are: width, height and set_values().

INPUT:

Create three Base classes (B1, B2, and B3) which contains data members

1. name, date of birth, blood group
2. height and weight
3. insurance policy number and contact address

Create Derived class which consist data members telephone numbers and driving license number

OUTPUT :

1. Master table containing all data Member
2. Display
3. Insert a new entry
4. Delete entry
5. Edit
6. Search for a record

FAQ:

1. What are the different forms of inheritance?
2. When do we use protected access specifier?
3. How does containership differ from inheritance?
4. What are Virtual Functions? How to implement virtual functions in "C"
5. Explain protected mode

PRACTICE ASSIGNMENTS:

Design a base class with Employee name, date of birth, blood group and another base class consisting of the data members such as designation and salary. Design one more base class consisting of the insurance policy number and contact address. The derived class contains the data member telephone numbers and driving license number. Write a menu driven program to carry out the following things:

Build a master table
Display
Insert a new entry
Delete entry
Edit
Search for a record

Subject Coordinator **Head of Department (I.T)**

Mr. Abhinay G. Dhamankar **Dr. B. A. Sonkamble**

Assignment No :5
MULTIPLE INHERITANCE

AIM:

Create employee bio-data using following classes

i. Personal record
ii. Professional record
iii. Academic record

Assume appropriate data members and member function to accept required data & print bio-data. Create bio-data using **multiple inheritance** using C++.

OBJECTIVE:

Understand the concept of multiple inheritance

THEORY:

Multiple inheritance

Allows you to create a derived class that inherits properties from more than one base class. Because a derived class inherits members from all its base classes, ambiguities can result. For example, if two base classes have a member with the same name, the derived class cannot implicitly differentiate between the two members.

In the following example, classes A, B, and C are direct base classes for the derived class X:

```
class A { /* ... */ };
class B { /* ... */ };
class C { /* ... */ };
class X : public A, private B, public C { /* ... */ };
```

INPUT:

Create three Base classes

(Personol_details, Professional_details, and Academic_details) which contains data members

1.name, date of birth, blood group
2.no_of_yr _exp, skill_expert
3.degree_percentile

Create Derived class Resume which consist data members from three base class and display resume in derived class.

OUTPUT :

1. Read detail

2 .Display detail

FAQ:

1. What are the different forms of inheritance?
2. When do we use protected access specifiers?
3. Diagram of multiple inheritance
4. What's the difference between public, private, and protected?
5. Pictorial representation of object from your program

PRACTICE ASSIGNMENTS:

Write C++ program using three classes

a. Student's personal information (Name, Address, Phone no, Birthdate etc.)
b. Student academic information (10th, 12th and F.E.)
c. Student's other information (Project done, Seminar, Hobbies, and Sports Record). Use multiple inheritance and print bio-data of particular student.

Subject Coordinator	Head of Department (I.T)
Mr. Abhinay G. Dhamankar	Dr. B. A. Sonkamble

Assignment No :6

INHERITANCE USING VIRTUAL FUNCTION

AIM:

Create a base class shape with two double type values and member functions to input the data and compute_area() for calculating area of figure.

Derive two classes' triangle and rectangle. Make **compute_area()** as a **virtual function** and redefine this function in the derived class to suit their requirements.

Write a program that accepts dimensions of triangle/rectangle and display calculated area.

OBJECTIVE:

Study the concept of Polymorphism

Understand the function overriding

Understand the concept of virtual function.

THEORY:

Virtual Function:

A **virtual** function is a function in a base class that is declared using the keyword **virtual**. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function.What we do want is the selection of the function to be called at any given point in the program to be based on the kind of object for which it is called. This sort of operation is referred to as **dynamic linkage**, or **late binding**.

Pure Virtual Functions:

It's possible that you'd want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class

C++ Virtual Function - Properties:

1. A member function of a class

2. Declared with *virtual* keyword
3. Usually has a different functionality in the derived class
4. A function call is resolved at run-time

The difference between a non-virtual c++ member function and a virtual member function is, the non-virtual member functions are resolved at compile time. This mechanism is called ***static binding***. Whereas the c++ virtual member functions are resolved during run-time. This mechanism is known as ***dynamic binding***.

C++ Virtual Function - Reasons:

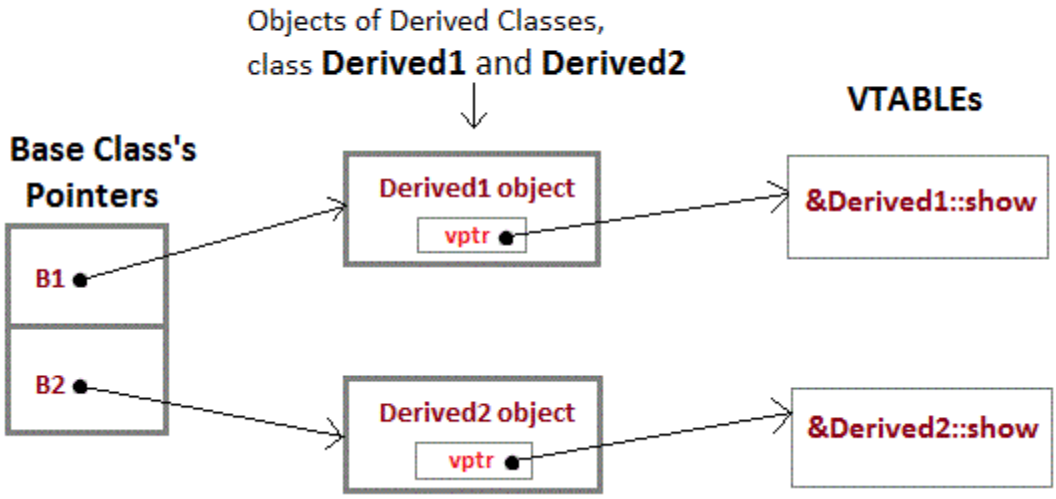
The most prominent reason why a C++ virtual function will be used is to have a different functionality in the derived class.

Rules for virtual functions:

When virtual functions are created for implementing late binding, we should observe some basic rules that satisfy the compiler requirements.

1. The virtual functions must be member of the same class
2. They cannot be static members
3. They are accessed by using object pointers
4. A virtual function can be a friend of another class.

Mechanism of Late Binding :



vptr, is the vpointer, which points to the Virtual Function for that object.

VTABLE, is the table containing address of Virtual Functions of each class.

To accomplich late binding, Compiler creates VTABLEs, for each class with virtual function. The address of virtual functions is inserted into these tables. Whenever an object of such class is created the compiler secretly inserts a pointer called vpointer, pointing to VTABLE for that object. Hence when function is called, compiler is able to resovle the call by binding the correct function using the vpointer.

Important Points to Remember

1. Only the Base class Method's declaration needs the Virtual Keyword, not the definition.
2. If a function is declared as virtual in the base class, it will be virtual in all its derived classes.
3. The address of the virtual Function is placed in the VTABLE and the copiler uses VPTR(vpointer) to point to the Virtual Function.

INPUT:

Create three Base classes

(Personol_details, Professional_details, and Academic_details) which contains data members

1.name, date of birth, blood group

2.no_of_yr _exp, skill_expert

3.degree_percentile

Create Derived class Resume which consist data members from three base class and display resume in derived class.

OUTPUT :

1. Read detail
2 .Display detail

FAQ:

1. What is diamond problem?
2. What is a virtual function?
3. When do we make a virtual function pure?
4. What is virtual base class?
5. What is an abstract class?

PRACTICE ASSIGNMENTS:

1. Write a program in C++ that makes the use of abstract base class. Base Class name is “Draw object” and various other derived classes (circle, rectangle, triangle, line, point). To construct a picture of truck use virtual function.

Subject Coordinator **Head of Department (I.T)**

Mr. Abhinay G. Dhamankar **Dr. B. A. Sonkamble**

Assignment No :7
EXCEPTION HANDLING

AIM:

Write a program in C++ which includes the code for following operations :

i. A function to read two double type numbers from keyboard

ii. A function to calculate the division of these two numbers

iii. A **try block** to detect and throw an exception if the condition "divide-by-zero"occurs

iv. Appropriate **catch block** to handle the **exceptions** thrown

OBJECTIVE:

Exception handling.

THEORY:

One benefit of C++ over C is its exception handling system. An exception is a situation in which a program has an unexpected circumstance that the section of code containing the problem is not explicitly designed to handle. In C++, exception handling is useful because it makes it easy to separate the error handling code from the code written to handle the chores of the program. Doing so makes reading and writing the code easier.

Furthermore, exception handling in C++ propagates the exceptions up the stack; therefore, if there are several functions called, but only one function that needs to reliably deal with errors, the method C++ uses to handle exceptions means that it can easily handle those exceptions without any code in the intermediate functions. One consequence is that functions don't need to return error codes, freeing their return values for program logic.

When errors occur, the function generating the error can 'throw' an exception. For example, take a sample function that does division:

```
const int DivideByZero = 10;
//....
double divide(double x, double y)
{
  if(y==0)
```

```
  {
    throw DivideByZero;
  }
  return x/y;
}
```

The function will throw DivideByZero as an exception that can then be caught by an exception-handling catch statement that catches exceptions of type int. The necessary construction for catching exceptions is a try catch system. If you wish to have your program check for exceptions, you must enclose the code that may have exceptions thrown in a try block. For example:

```
try
{
  divide(10, 0);
}
catch(int i)
{
  if(i==DivideByZero)
  {
    cerr<<"Divide by zero error";
  }
}
```

The catch statement catches exceptions that are of the proper type. You can, for example, throw objects of a class to differentiate between several different exceptions. As well, once a catch statement is executed, the program continues to run from the end of the catch.

It is often more useful for you to create a class that stores information on exceptions as they occur. For example, it would be more useful if you had a class to handle exceptions.

```
class DivideByZero
{
  public:
    double divisor;
    DivideByZero(double x);
};
DivideByZero::DivideByZero(double x) : divisor(x)
{}
int divide(int x, int y)
{
```

```
  if(y==0)
  {
    throw DivideByZero(x);
  }
}
try
{
  divide (12, 0);
}
catch (DivideByZero divZero)
{
  cerr<<"Attempted to divide "<<divZero.divisor<<" by zero";
}
```

If you wish to catch more than one possible exception, you can specify separate catch blocks for each type of exception. It's also possible to have a general exception handler that will respond to any thrown exception. To use it, simply use catch (...) for the catch statement and print a general warning of some kind.

The handy thing to remember about exception handling is that the errors can be handled outside of the regular code. This means that it is easier to structure the program code, and it makes dealing with errors more centralized. Finally, because the exception is passed back up the stack of calling functions, you can handle errors at any place you choose.

In C, you might see some error handling code to free memory and close files repeated five or or six times, once for each possible error. A solution some programmer's preferred was to use a goto statement that jumped all the way to the cleanup code. Now, you can just surround your code with a try-catch block and handle any cleanup following the catch (possibly with an additional copy of your cleanup routine inside the catch block if you intend to throw the exception again to alert the calling function of an error).

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch,** and **throw**.

- **throw:** A program throws an exception when a problem shows up. This is done using a **throw** keyword.
- **catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
- **try:** A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks

INPUT:

Accept two double integers

Data members: two double integer
Member functions:
Calculate division of these number.
Handle the "divide-by-zero exception using catch and thrown statement.
Get data, Display data.

OUTPUT :

If exception is caught display error message.

FAQ:

1. How exception is handled in c++?
2. When should a program throw exception?
3. When do we use multiple catch handlers?

PRACTICE ASSIGNMENTS:

1. Perform Division by zero exceptional handling?
2. Perform Array Index out of range exceptional handling?

Subject Coordinator	**Head of Department (I.T)**
Mr. Abhinay G. Dhamankar	**Dr. B. A. Sonkamble**

Assignment No 8
GENERIC PROGRMMING : TEMPLATES

AIM:

Write a program in C++ using function/class template to read two matrices of different data types such as integers and floating point values and perform simple arithmetic operations on these matrices separately and display it.

OBJECTIVE:

Understanding of Template and Template function.

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 are 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(T a, T b)
{
```

Return a+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 typename. 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.

INPUT:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

OUTPUT :

$$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

FAQ:

1. Distinguish between overloaded functions and function templates?
2. What is the difference between macro and template?
3. What is the use of class template?
4. What is the difference between class template & function template?
5. What do you mean by a class template instantiation?
6. Can we use static data members in class template?
7. How to define a member function outside of its template?
8. How to define and declare the function template?
9. List the advantages & disadvantages of function template.
10. Can we overload the function template?

PRACTICE ASSIGNMENTS:

1. Write C++ programs to create a template class QUEUE with add and delete member functions. Using it, implement a queue of integers and doubles.

2. Write a program to create a class template LINKED LIST with insert & delete operations. Using it, implement singly linked list of integer & floating point values.

Subject Coordinator	**Head of Department (I.T)**
Mr. Abhinay G. Dhamankar	**Dr. B. A. Sonkamble**

Assignment No :9

FILE OPERATIONS C++

AIM:

Design a base class consisting of the data members such as name of the student, roll number and subject. The derived class consists of the data member's subject code, internal assessment and university examination marks. Construct a virtual base class for the item name of the student and roll number. The program should have the facilities. i) Build a master table ii) List a table iii) Insert a new entry iv) Delete old entry v) Edit an entry vi) Search for a record

OBJECTIVE:

To understand how various operations can be performed on files.
Virtual Function

THEORY:

Classes for file stream operations.
File modes.
Function: seekg(), seekp(), tellg(), tellp(), put(), get(), write(), read().
C++ virtual function is a member function of a class, whose functionality can be over-ridden in its derived classes. The whole function body can be replaced with a new set of implementation in the derived class. An iessensial property of the polymorphism is the ability to refer to objects w/o any regard to their classes. This necessitates the use of a single pointer variable to refer to the objects of different clases. Here we use pointers to the base class to refer to all the derived objects. But the base pointer , even when it is made to contain the address of a derived class , always executes the function in the base class. The compiler simply ignores the contents of the pointer and chooses the member function that matches the type of the pointer. This problem is solved by virtual function.

Input/output with files

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

- **ofstream:** Stream class to write on files
- **ifstream:** Stream class to read from files
- **fstream:** Stream class to both read and write from/to files.

These classes are derived directly or indirectly from the classes istream, and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therfore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout, with the only difference that we have to associate these streams with physical files

Opening files using open()

The function open () can be used to open multiple files that use same stream object .

```
File_stream_class stream _object;
Stream_object .open ("filename", mode);
```

Eg fstream inoutfile;

Inoutfile.open("book.dat");

Where filename is a null-terminated character sequence of type const char * (the same type that string literals have) representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set to any value, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file. This flag can only be used in streams open for output-only operations.
ios::trunc	If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one.

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open():

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

For ifstream and ofstream classes, ios::in and ios::out are automatically and respectivelly assumed, even if a mode that does not include them is passed as second argument to the open() member function.

The default value is only applied if the function is called without specifying any value for the mode parameter. If the function is called with any value in that parameter the default mode is overridden, not combined.

File streams opened in binary mode perform input and output operations independently of any format considerations. Non-binary files are known as *text files*, and some translations may occur due to formatting of some special characters (like newline and carriage return characters).

Since the first task that is performed on a file stream object is generally to open a file, these three classes include a constructor that automatically calls the open() member function and has the exact same parameters as this member. Therefor, we could also have declared the previous myfile object and conducted the same opening operation in our previous example by writing:

```
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

Combining object construction and stream opening in a single statement. Both forms to open a file are valid and equivalent.
To check if a file stream was successful opening a file, you can do it by calling to member is_open() with no arguments. This member function returns a bool value of true in the case that indeed the stream object is associated with an open file, or false otherwise:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

Closing a file

When we are finished with our input and output operations on a file we shall close it so that its resources become available again. In order to do that we have to call the stream's member function close(). This member function takes no parameters, and what it does is to flush the associated buffers and close the file:

```
myfile.close();
```

Once this member function is called, the stream object can be used to open another file, and the file is available again to be opened by other processes.

Checking state flags

eof() Returns true if a file open for reading has reached the end.

Functions for manipulation of file pointers :

All the actions on the file pointers take place automatically by default. The file stream classes support following functions to manage file pointers manually.

tellg() and tellp()

These two member functions have no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the get stream pointer (in the case of tellg) or the put stream pointer (in the case of tellp).

seekg() and seekp()

These functions allow us to change the position of the get and put stream pointers. Both functions are overloaded with two different prototypes. The first prototype is:

```
seekg ( position );
seekp ( position );
```

Using this prototype the stream pointer is changed to the absolute position position (counting from the beginning of the file). The type for this parameter is the same as the one returned by functions tellg and tellp: the member type pos_type, which is an integer value.

INPUT:

Enter the students Details- Name, Age etc

OUTPUT :

1. Master table containing all data Member
2. Display
3. Insert a new entry
4. Delete entry
5. Edit
6. Search for a record

FAQ:

1. What are two different ways to open the file?
2. Explain seekg(), seekp(), tellg(), tellp();
3. Differentiate between 'ate' mode and 'app' mode.
4. Explain the technique of static and dynamic binding.

PRACTICE ASSIGNMENTS:

1. Write a C++ program to generate Fibonacci series of n numbers using run time binding. The series initials are 0 and 1.
2. Write a C++ program to maintain the employee record using file handling.

Subject Coordinator	**Head of Department (I.T)**
Mr. Abhinay G. Dhamankar	**Dr. B. A. Sonkamble**

Assignment No :10

SEQUNTIAL FILE OPERATIONS IN C++

AIM:

Write a program in C++ to implement **sequential file** for students' database and perform following operations on it

i. Create
ii. Display
iii. Add
iv. Delete
v. Modify

OBJECTIVE:

To understand how various operations can be performed on sequential files.

THEORY:

The idea of storing the information has laid down the concept of file handling.

A file is a bunch of bytes stored on some storage devices like hard disk, floppy disk etc

Most of the application programs process large volume of data which is permanently stored infiles.

File Handling Introduction-

The idea of storing the information has laid down the concept of file handling.

A file is a bunch of bytes stored on some storage devices like hard disk, floppy disk etc

Most of the application programs process large volume of data which is permanently stored infiles.

We can write program that can read data from file(s) and write data to file(s). Data transfer is generally done in two ways:

i. Reading data from Input device (memory) and writing into file(s).

ii. Reading data from file(s) and writing into or display on Output device(memory)

Need of file Handling

i)Fast response: There are applications where the required info. Is needed very fast. Insuch situation if the data is stored in a file(s) it can be utilized immediately.

ii)Ease of Use : There are different ways of utilizing the data by different users. Data can beshared by users if it is stored in the form of files

iii)Constraint on performance: in real time systems, there are constraints on the performance of the system.

iv)Saving of repetitive typing of data: – if the same data is needed again and again for processing we need not to type it repeatedly, if stored in files

Types of Data files – TEXT FILE and BINARY FILE

Text Files : It contains alphanumeric data input using a text editor program. ASCII file format isused, so it can be displayed on screen or manipulated easily (Less Data Security) . Datatranslation is required, resulting in slower speed.

Binary File : It allows us to write alphanumeric data to the disk file in less number of files as compared to text files. The data is stored in Binary (Machine) language, not readable without program code, data is secure as no easy manipulation is there. Speed is faster as data needs no

Steps to use (or read) a sequential access file:

Required header: **<fstream.h>**

It allows classes ofstream and ifstream to become available.

1. Declare a stream variable name:

   ifstream fin; // each file has its own stream buffer

2. Open the file

   fin.open("myfile.dat", ios::in);

   fin is the stream variable name previously declared
   "myfile.dat" is the name of the file
   ios::in is the steam operation mode

3. Read data from the file.

```
//to read one number and
//one string variable
int number;
apstring item;
fin>>number>>item;
```

```
//to read one number and
//one string variable
int number;
apstring item, dummy;
getline(fin, item);
fin>>item;
```

This format is OK only if the string variable is a **single** word.	getline(fin,dummy); This format is needed if the string variable contains whitespace.

4. Close the file

fin.close( );

Note: Always close your files as soon as you are finished accessing information.

INPUT:

Enter the students Details- Student id, Name, Age, etc

OUTPUT :

Master table containing all data Member

1. Insert a new entry
2. Display
3. Delete entry
4. Edit
5. Search for a record

FAQ:

6. What are two different ways to open the file?
7. Explain seekg(), seekp(), tellg(), tellp();
8. Differentiate between 'ate' mode and 'app' mode.

PRACTICE ASSIGNMENTS:

1. Write a C++ program to maintain the employee record using file handling.

Subject Coordinator
Mr. Abhinay G. Dhamankar

Head of Department (I.T)
Dr. B. A. Sonkamble