

**Department of Computer Science & Engineering**

**LAB MANUAL**

**CS 1365 Object Oriented Concepts & Programming  
using C++ Lab**

**(B. Tech III Semester)**



**SMU** SIKKIM  
MANIPAL  
UNIVERSITY

Established under Govt. of Sikkim, Act 9 of 1995, recognised under 2(f) of the UGC Act, 1956

**Sikkim Manipal Institute of Technology  
Majitar, Rangpo, East Sikkim**



## List of Programs

SN	Program Description	Page No.	Remarks	Signature
<b>LAB 1: Structure concepts</b>		<b>Date:</b>		
1.1	Write a program to create a structure named "cricket" with following details:- a. Player name    b. Team name    c. Batting average Using structure cricket declare an array with 50 elements & read information about all the players and display team-wise list containing names of players with their batting average score.	8		
1.2	Define a structure named "census" with following three members: i. A character array "city" to store the names. ii. A long integer to store population of the city. iii. A float number to store literacy level. Implement the following:- i. Read the details for five cities randomly using an array variable. ii. Sort the city list alphabetically and display all the details.	11		
1.3	Define a structure named "Date" with three integer members "day", "month" and "year". Develop an interactive modular program to perform the following tasks: i. To read the data into structure members using a function. ii. Create a function named validate() to validate the date Display the date in the format "April 29 2014" using a function.	14		
<b>LAB 2: Implementing classes and objects</b>		<b>Date:</b>		
2.1	Create a class named circle with data member Radius, pi (constant member) and area. Create following member functions: i. to accept the radius from the user ii. to calculate the area of the circle iii. to display the details	21		

2.2	<p>Given that an EMPLOYEE class contains following data members: Employee_Number, Employee_Name, Basic, DA, IT, Net_Sal. Member Functions: to read the data, to calculate Net_Sal and to print data members. Write a C++ program to read the data of an employee and compute Net_Sal of the employee (DA= 52% of Basic and Income Tax(IT)=30% of the gross salary).While writing consider following:</p> <ol style="list-style-type: none"> <li>Initialize members using method.</li> <li>Create more than two objects.</li> </ol>	23		
2.3	<p>Write an OOP in C++ to prepare a student Record using class and object with the following data members and member functions:</p> <p>Data Members: Name, Regn Number, and Marks in three subjects</p> <p>Member functions: Read, display and average marks</p>	25		
<b>LAB 3: Array of objects</b>			<b>Date:</b>	
3.1	<p>Define a STUDENT class with USN, Name and marks in 3 tests of a subject, declare an array of 10 student objects find the average of two better marks for each student. Print USN, Name and average marks of all the students.</p>	30		
3.2	<p>Write an OOP in C++ to create a class Employee with data members as Name, Employee code and Gross salary. Include member functions to read input and display output of individual objects. Create an array of 10 employees. Display the details of the employees with highest and lowest gross salary.</p>	33		
3.3	<p>Write an OOP in C++ to create a class called BOOK with the following data members and member function:</p> <p>Data Members: Title, Author, Price,ISBN No.</p> <p>Member functions: Read, Display, Search_book(using ISBN No).</p> <p>Create an array of 5 BOOK type objects and demonstrate all member functions.</p>	36		

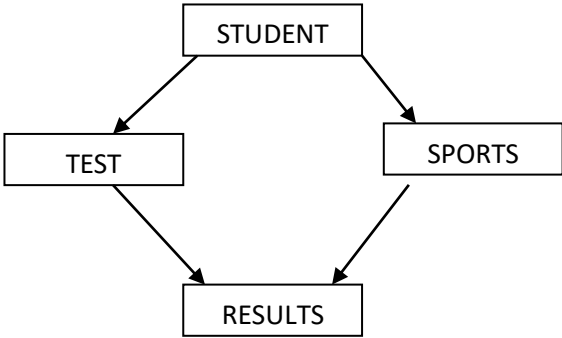
LAB 4: Function overloading, passing objects as arguments and returning objects		Date:		
4.1	Create a class named C_POWER with a function <b>power()</b> to raise a number <b>m</b> to the power <b>n</b> . The function takes a <b>double</b> value for m and <b>int</b> value for n, returns the result correctly. Use a default value of 2 for n to make the function to calculate squares when this argument is omitted. Write another function that takes an <b>int</b> value of <b>m</b> . Both functions should have same name. Write a <b>main</b> function that gets values of m and n from user.	42		
4.2	Write a C++ program to create a class called COMPLEX and implement the following by overloading functions ADD() that return a COMPLEX number. ADD(a, s2)- where a is an integer (real part) and s2 is a complex number. ADD(s1, s2)- where s1 and s2 are complex numbers.	44		
4.3	Write a program in C++ to create a class named TIME that stores two time values in hours (float) and minutes (integer). Perform the following operations by using a function add():- Add T1+T2 where , T1 and T2 are time objects. Add T1+x where T1 is a TIME object and x is any integer. Return time objects from the member function to main() and display.	47		
LAB 5: Programs on constructors and destructors		Date:		
5.1	Write a C++ program to create a class called STRING and implement the following operations. STRING s1="SMIT" STRING s2="MAJITAR" STRING s3=s1+s2 (Use copy constructor).	57		
5.2	Write a program in C++ to create a class named student, create a constructor and destructor for the class and demonstrate sequence of execution of constructors and destructors.	60		
LAB 6: Static data and static member function		Date:		
6.1	Write a program in C++ to define a class named Bank.  Include the following data members:- a. Name of account holder      b. Account number	65		

	c. type of account d. balance amount in account e. number of transactions in the bank(static data member)  Create the following member functions:- i. To read inputs ii. To deposit an amount iii. To withdraw amount after checking current balance Display number of transactions (use static function).			
6.2	Write an OOP in C++ to demonstrate static members of a class. Include static data members to count the number of objects created for the class	69		
6.3	Write an OOP in C++ to demonstrate static members function of a class to pre-initialize private static data members. Also show restrictions that apply on static member functions.	71		
<b>LAB 7: Friend function.</b>		<b>Date:</b>		
7.1	Create two classes DM and DB which stores the value in distances. DM stores distance in meters and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB. Use a friend function to carry out the addition operation. The object that stores the results may be a DM object or DB object, depending on the units in which the results are required. The display should be in the format of feet and inches or meters and centimeters depending on the object on display. Use the concept of friend function.	75		
7.2	Write an OOP in C++ to create a Class Twovalues with two integer type data members and member function to read values in those data members. Create another Class Min_Max and declare it as friend of Class Twovalues. Min_Max should have member functions to find the minimum and maximum of two values.	78		

7.3	Write an OOP in C++ to demonstrate the advantage of overloading '+' operator using friend function.	81		
<b>LAB 8: Implementing linked list in C++</b>		<b>Date:</b>		
8.1	Write a C++ program to create a class called LIST (linked list) with member functions to :- i. Insert an element at any position. ii. To display the list.	83		
8.2	Write a C++ program to create a class called LIST (linked list) with member functions to :- i. Delete an element from any position. ii. To display the list.	87		
8.3	Write an OOP in C++ to create a class called STACK with an array of integer type data member and member functions to PUSH/POP elements from the stack and display the elements on the stack.	90		
<b>LAB 9: Operator overloading</b>		<b>Date:</b>		
9.1	Write a program in C++ to overload unary minus operator to change the sign of a number. Perform the addition on the entered values from text boxes.	96		
9.2	Write a C++ program to create a class called COMPLEX and implement the following by overloading + operator:- i. a + s2; where a is an integer (real part) and s2 is a complex number. ii. s1 + s2; where s1 and s2 are complex numbers. iii. Return object and display the result.	99		
9.3	Write a C++ program to create a class called STACK using array of integers. Implement the following operations by overloading the operators + and --. i) s1 = s1 + element; where s1 is a object of the class STACK and element is an integer to be pushed on top of the stack. ii) s1 = s1 --; where s1 is a object of the class STACK, -- operator pops the element. <b>Note: Handle the STACK empty and STACK full conditions.</b>	102		

9.4	Write a C++ program to create a class called MATRIX using a two- dimensional array of integers. Implement the following operations by overloading the operator== which checks the compatibility of two matrices to be added and subtracted. Perform the addition and subtraction by overloading the operators + and - respectively :- <pre>             if(m1==m2)             {                 m3=m1-m2;                 m4=m1+m2;             }             else display error </pre>	105		
9.5	Write a C++ program to create a class called DATE. Accept two valid dates in the form dd/mm/yy. Implement the following perations by overloading the operators + and - . i. no_of_days=d1-d2; where d1 and d2 are DATE objects. d1>=d2 and no_of_days is an integer. ii. d2=d1-no_of_days; where d1 is a DATE object and no_of_days is an integer.	108		
<b>LAB 10: Template class and template functions.</b>		<b>Date:</b>		
10.1	Write a program in C++ to create a template function and perform bubble sort.	113		
10.2	Write a program, in C++ to create a template class STACK and perform push and pop operations.	116		
10.3	Write an OOP in C++ to create a template class ARRAY and implement run time boundary checking by overloading [] operator.	119		
<b>LAB 11: Inheritance</b>				
11.1	Write a C++ program to create a class called STUDENT with data members USN, Name and Age. Using inheritance, create the classes UGSTUDENT and PGSTUDENT having fields as Semester, Fees and Stipend. Enter the data for at least 5 students. Find the semester wise average age for all UG and PG students separately.	126		
11.2	Write an OOP in C++ to demonstrate accessing derived class objects' members using base class	129		



	pointers. Also show the sequence of execution of constructors and destructors when inheritance is involved.			
<b>LAB 12: Run-Time Polymorphism</b>		<b>Date:</b>		
12.1	WAP in C++ to create a class called <b>Figure</b> . Use this class to store two double type values that could be used to compute the area of various shapes (i.e. length, breadth for rectangle and base, height for triangle). Derive two specific classes called <b>Triangle</b> and <b>Rectangle</b> inherited from super class <b>Figure</b> . Create a virtual function called <b>area( )</b> to compute and display the area of the basic shapes. Redefine this function in derived classes to suit its requirements.	137		
12.2	<p>Write a C++ to implement the following using the concept of inheritance.-</p>  <pre> graph TD     STUDENT --&gt; TEST     STUDENT --&gt; SPORTS     TEST --&gt; RESULTS     SPORTS --&gt; RESULTS </pre> <ol style="list-style-type: none"> <li>Create data member roll_no, reg_no and member function get_number() to take input and put_number() to display in the base class STUDENT.</li> <li>The class Test contains a data member part1 and part2. Redefine the above two functions.</li> <li>The class SCORE contains a member score. Redefine get_number and put_number ().</li> <li>Redefine display function in RESULT class as well to calculate total, where  TOTAL=part1+part2+score.</li> </ol> <p><b>Note: Implement the concept of run-time polymorphism, and include constructors in all the classes.</b></p>	140		

12.3	Write an OOP in C++ to create a base class convert with two data members val1 and val2 and two member functions getinit() and getconvert() which return the initial values and converted values. It includes a pure virtual function compute() which must be defined by by two derived classes lit_to_gal and far_to_cel. Each of these class have their own definition of compoute()	143		
	<b>LAB 13: Files in C++</b>	<b>Date:</b>		
13.1	Write a program in C++ to create a file named "costs" and implement the following by creating menu driven program:- i. The file should contain two columns "item_name" and "cost". ii. Insert data under these two columns assuming the file is empty initially. iii. Allow user to add more data as and when he wants. iv. Retrieve the item details if item name is given as an input. v. Display the entire content of the file.	147		
13.2	Write a program in C++ to copy the contents of one file into another in reverse order.	150		
13.3	Write a program in C++ to change a particular character in a file using command line.	152		

### Course Objective:

At least 12 experiments covering the entire syllabus of the corresponding theory paper to be carried out using the theory studied /programming skill of the subject concerned to get insight into the practical applications of the theoretical studies.

**Course outcomes:** On successful completion of this course, students will be able to

1. Write an object-oriented program using C++.
2. Solve a problem using concept of OOP.
3. Use the idea of data reusability.
4. To understand how C++ improves C with object-oriented features.

### Evaluation Plan:

- a) Internal Assessment Marks: 60
- b) Continuous evaluation component  
Each experiment and Lab file :10 marks

*Note: The assessment will depend on punctuality, program execution, maintaining the lab record and viva voce on each lab*

- c) Total marks of minimum 12 experiments scaled down to 60.
- d) End semester assessment of 3 hour duration: 40 Marks

### NOTE: Overall Lab Assessment Scheme

Continuous assessment of lab-classes	60 marks
Department level lab examination	40 marks
Total	100 marks

### INSTRUCTIONS TO THE STUDENTS

#### Pre-Lab Session Instructions

1. Students should carry the Lab Manual and the required stationery to every lab session.
2. Be in time and follow the institution dress code.
3. Must Sign in the log register provided (during 1<sup>st</sup> lab only).
4. Make sure to occupy the allotted seat and answer the attendance.
5. Adhere to the rules and maintain the decorum.

#### In-Lab Session Instructions

- Follow the instructions on the allotted exercises.
- Show the program and results to the instructors on completion of experiments.
- On receiving approval from the instructor, copy the program and results in the Lab record.
- Prescribed textbooks and class notes can be kept ready for reference if required.

#### General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:

- Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
- Programs should perform input validation (Data type, range error, etc.) and five appropriate error messages and suggest corrective actions.
- Comments should be used to give the statement of the problem and every member function should indicate the purpose of the member function, inputs and outputs.
- Statements within the program should be properly indented.
- Use meaningful names for variables, classes, interfaces, packages and methods.
- Make use of constant and static members wherever needed.
- Plagiarism (copy from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
  - Solved exercise
  - Lab exercises- to be completed during lab hours
  - Additional Exercises – to be completed outside the lab or in the lab to enhance the skill.
- In case a student misses a lab class, he/she must ensure that the experiment is completed during the repetition class with the permission of the HOD/faculty concerned but credit will be given only to one day's experiment(s).

**Note: Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and/or combinations of the questions.**

**The students should not**

- Bring mobile phones or any other electronic gadgets (pen drive/tablets etc.) to the lab.
- Go out of the lab without permission.
- Enter the lab in casual dress.
- Delete or change any system settings/configuration.



## LAB 1: Structure concepts

### C++ Structure:

Structure is commonly referred to as user-defined data type. Structure is similar to an array but the only difference is that array is collection of similar data type on the other hand structure is collection of different data type. A structure can contain any data type including array and another structure as well. Each variable declared inside structure is called member of structure.

### Structure Declaration:

Declaration of structure must start with the keyword **struct** followed by the structure name and structure's member variables are declared within braces.

### Syntax for declaring structure:

---

```
struct structure-name
{
    datatype var1;
    datatype var2;
    -----
    -----
    datatype varN;
};
```

---

### Example for declaring structure:

---

```
struct Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};
```

---

### Accessing the structure members:

We have to create an object of structure to access its members. Object is a variable of type structure. Structure members are accessed using the dot operator(.) between structure's object and structure's member name.

**Syntax for creating object:**

structure-name obj;

**Example for creating object & accessing structure members:**

```
#include<iostream.h>
struct Employee
{   int Id;
    char Name[25];
    int Age;
    long Salary;
};
void main()
{
    Employee E;           //Statement 1
    cout << "\nEnter Employee Id : ";
    cin >> E.Id;
    cout << "\nEnter Employee Name : ";
    cin >> E.Name;
    cout << "\nEnter Employee Age : ";
    cin >> E.Age;
    cout << "\nEnter Employee Salary : ";
    cin >> E.Salary;
    cout << "\n\nEmployee Id : " << E.Id;
    cout << "\nEmployee Name : " << E.Name;
    cout << "\nEmployee Age : " << E.Age;
    cout << "\nEmployee Salary : " << E.Salary;
}
```

**Output :**

```
Enter Employee Id : 1
Enter Employee Name : Kumar
Enter Employee Age : 29
Enter Employee Salary : 45000
```

```
Employee Id : 1
Employee Name : Kumar
Employee Age : 29
Employee Salary : 45000
```

Statement 1 is creating an object E of Employee type.

**Initialization of structure:**

Like normal variable structures can be initialized at the time of declaration. Initialization of structure is almost similar to initializing array. The structure object is followed by equal sign and the list of values enclosed in braces and each value is separated with comma.

**Example for declaring & initializing structure at same time:**

---

```
#include<iostream.h>
struct Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};
void main()
{
    Employee E = {2,"Suresh",35,35000};
    cout << "\n\nEmployee Id : " << E.Id;
    cout << "\nEmployee Name : " << E.Name;
    cout << "\nEmployee Age : " << E.Age;
    cout << "\nEmployee Salary : " << E.Salary;
}
```

**Output :**

```
Employee Id : 1
Employee Name : Kumar
Employee Age : 29
Employee Salary : 45000
```

---

**Array of Structure:**

Structure is collection of different data type. An object of structure represents a single record in memory, if we want more than one record of structure type, we have to create an array of structure or object. As we know, an array is a collection of similar type, therefore an array can be of structure type.



### Syntax for declaring structure array:

---

```
struct structure-name
{
    datatype var1;
    datatype var2;
    -----
    -----
    datatype varN;
};
structure-name obj [ size ]; // structure array
```

---

### Array within Structure:

As we know, structure is collection of different data type. Like normal data type, It can also store an array as well.

### Syntax for array within structure:

---

```
struct structure-name
{
    datatype var1;           // normal variable
    datatype array [size];   // array variable
    -----
    -----
    datatype varN;
};

structure-name obj;
```

---

### Structure and function in C++:

Using function we can pass structure as function argument and we can also return structure from function.

### Passing structure as function argument:

Structure can be passed to function through its object therefore passing structure to function or passing structure object to function is same thing because structure object represents the structure. Like normal variable, structure variable (structure object) can be pass by value or by references / addresses.

**Passing structure by value:** In this approach, the structure object is passed as function argument to the definition of function, here object is representing the members of structure with their values.

**Example for passing structure object by value:**

---

```
#include<iostream.h>
struct Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};
void Display(struct Employee);
void main()
{
    Employee Emp = {1,"Kumar",29,45000};
    Display(Emp);
}
void Display(struct Employee E)
{
    cout << "\n\nEmployee Id : " << E.Id;
    cout << "\nEmployee Name : " << E.Name;
    cout << "\nEmployee Age : " << E.Age;
    cout << "\nEmployee Salary : " << E.Salary;
}
```

**Output :**

```
Employee Id : 1
Employee Name : Kumar
Employee Age : 29
Employee Salary : 45000
```

**Passing structure by reference:**

In this approach, the reference/address structure object is passed as function argument to the definition of function.

### Example for passing structure object by reference:

---

```
#include<iostream.h>
struct Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};
void Display(struct Employee*);
void main()
{
    Employee Emp = {1,"Kumar",29,45000};
    Display(&Emp);
}
void Display(struct Employee *E)
{
    cout << "\n\nEmployee Id : " << E->Id;
    cout << "\nEmployee Name : " << E->Name;
    cout << "\nEmployee Age : " << E->Age;
    cout << "\nEmployee Salary : " << E->Salary;
}
```

#### Output :

```
Employee Id : 1
Employee Name : Kumar
Employee Age : 29
Employee Salary : 45000
```

### Function Returning Structure:

Structure is user-defined data type, like built-in data types structure can be return from function.

### Example for function returning structure :

---

```
#include<iostream.h>
struct Employee
{
    int Id;    char Name[25];    int Age;    long Salary;
};

Employee Input();    //Statement 1
```

```
void main()
{   Employee Emp;
    Emp = Input();
    cout << "\n\nEmployee Id : " << E.Id;
    cout << "\nEmployee Name : " << E.Name;
    cout << "\nEmployee Age : " << E.Age;
    cout << "\nEmployee Salary : " << E.Salary;
}
Employee Input()
{   Employee E;
    cout << "\nEnter Employee Id : ";
    cin >> E.Id;
    cout << "\nEnter Employee Name : ";
    cin >> E.Name;
    cout << "\nEnter Employee Age : ";
    cin >> E.Age;
    cout << "\nEnter Employee Salary : ";
    cin >> E.Salary;
    return E;          //Statement 2
}
```

**Output :**

```
Enter Employee Id : 10
Enter Employee Name : Ajay
Enter Employee Age : 25
Enter Employee Salary : 15000
```

```
Employee Id : 10
Employee Name : Ajay
Employee Age : 25
Employee Salary : 15000
```

In the above example, statement 1 is declaring Input() with return type Employee. As we know structure is user-defined data type and structure name acts as our new user-defined data type, therefore we use structure name as function return type.

Input() have local variable E of Employee type. After getting values from user statement 2 returns E to the calling function and display the values.

**Lab assignments:**

- 1.1 Write a program to create a structure named “cricket” with following details:-  
Player name                      b. Team name      c. Batting average  
Using structure cricket declare an array with 50 elements & read information about all the players and display team-wise list containing names of players with their batting average score.

**Algorithm:**

Step1 : Declare a structure CRICKET

Step1.1 : create its members player\_name, team\_name, batting\_average

Step 2: Declare an array of structure for 50 elements.

Step 3: Read data into the structure.

Step 4: Create a function display()

Step 4.1: Display batting average score of players along with their names in  
team-wise manner.

Step 5: End





1.2 Define a structure named “census” with following three members:

- i. A character array “city” to store the names.
- ii. A long integer to store population of the city.
- iii. A float number to store literacy level.

Implement the following:-

- i. Read the details for five cities randomly using an array variable.
- ii. Sort the city list alphabetically and display all the details.







- 1.3 Define a structure named "Date" with three integer members "day" , "month" and "year". Develop an interactive modular program to perform the following tasks:
- i. To read the data into structure members using a function.
  - ii. Create a function named validate() to validate the date .
  - iii. Display the date in the format "April 29 2014" using a function.



## **LAB 2: Introduction to classes and objects**

### **Classes in C++:**

Class is an encapsulation of data and coding. Classes are an expanded version of structures. Structure can only contain multiple variables. Classes can contain multiple variables, even more, classes can also contain functions as class member. Variables declared in class are called Data Members (Attributes). Functions declared or defined in class are called Member Functions.

### **Class declaration:**

Declaration of class must start with the keyword **class** followed by the class name and class members are declared within braces.

### **Syntax of declaring class:**

---

```
class class-name
{
    datatype var1;
    datatype var2;
    -----
    datatype varN;

    method1();
    method2();
    -----
    methodN();
};
```

### **Example of declaring class:**

---

```
class Employee
{
    int Id;   char Name[25];
    int Age;
    long Salary;
    public:
    void GetData();
    void PutData();
};
```

---

### Instantiation of objects OR Accessing the class members:

Instantiation of object means, create an object of class to access its members. Object is a variable of class type. Class members are accessed using the dot operator(.) between class's object and class's member name.

### Syntax for creating an object of class:

---

```
class-name obj;
```

### Example for creating an object of class:

---

```
Employee Emp;
```

### Defining Class Member functions:

We can define member function inside or outside class.

### Syntax of defining member function outside class:

---

```
return_type class_name :: function_name(argument list)
{
    -----
    body of function
    -----
}
```

---

**return\_type** : Type of value function will return.

**class\_name::** A program may contain more than one class and these classes may have similar member functions. Class\_name:: tells the compiler which class the function belongs to and the scope of the member function is restricted to the class\_name.

**function\_name**: Can be any valid C++ identifier.

**argument list** : Represents the type and number of value function will take, values are sent by the calling statement.

### Example of defining member function outside class

---

```
#include<iostream.h>
#include<conio.h>
class Employee
{
    int Id;    char Name[25];    int Age;    long Salary;
```

```
public:
void GetData();
void PutData();
};
void Employee :: GetData()      //Statement 1 : Defining GetData()
{
    cout<<"\n\tEnter Employee Id : ";
    cin>>Id;
    cout<<"\n\tEnter Employee Name : ";
    cin>>Name;
    cout<<"\n\tEnter Employee Age : ";
    cin>>Age;
    cout<<"\n\tEnter Employee Salary : ";
    cin>>Salary;
}
void Employee :: PutData()      //Statement 2 : Defining PutData()
{
    cout<<"\n\nEmployee Id : "<<Id;
    cout<<"\nEmployee Name : "<<Name;
    cout<<"\nEmployee Age : "<<Age;
    cout<<"\nEmployee Salary : "<<Salary;
}
void main()
{
    Employee E;      //Statement 3 : Creating Object
    E.GetData();      //Statement 4 : Calling GetData()
    E.PutData();      //Statement 5 : Calling PutData()
}
```

**Output :**

```
Enter Employee Id : 1
Enter Employee Name : Kumar
Enter Employee Age : 29
Enter Employee Salary : 45000
```

```
Employee Id : 1
Employee Name : Kumar
Employee Age : 29
Employee Salary : 45000
```

---

In the above program, we have declared two member functions GetData() and PutData() within class and statement 1 and 2 is defining these methods

---

outside class. Statement 3 is creating an object E of Employee class. Statement 4 and 5 is invoking/calling member function through the object E of Employee class.

---

**Syntax of defining member function inside class:**

---

```
return_type function_name(argument list)
{
    -----
    body of function
    -----
}
```

---

Definition of member function inside class is similar to defining normal function. There is no need to tell compiler about the class the function belongs to because the definition of member function is already in the class.

---

**Example of defining member function inside class:**

---

```
#include<iostream.h>
#include<conio.h>
class Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
    public:
    void GetData()      //Statement 1 : Defining GetData()
    {
        cout<<"\n\tEnter Employee Id : ";
        cin>>Id;
        cout<<"\n\tEnter Employee Name : ";
        cin>>Name;
        cout<<"\n\tEnter Employee Age : ";
        cin>>Age;
        cout<<"\n\tEnter Employee Salary : ";
        cin>>Salary;
    }
    void PutData()      //Statement 2 : Defining PutData()
    {
        cout<<"\n\nEmployee Id : "<<Id;
```



```
        cout<<"\nEmployee Name : "<<Name;
        cout<<"\nEmployee Age : "<<Age;
        cout<<"\nEmployee Salary : "<<Salary;
    }
};
void main()
{
    Employee E;        //Statement 3 : Creating Object
    E.GetData();        //Statement 4 : Calling GetData()
    E.PutData();        //Statement 5 : Calling PutData()
}
```

**Output :**

```
Enter Employee Id : 1
Enter Employee Name : Kumar
Enter Employee Age : 29
Enter Employee Salary : 45000
```

```
Employee Id : 1
Employee Name : Kumar
Employee Age : 29
Employee Salary : 45000
```

---

In the above program, we have defined two member functions GetData() and PutData() within class. Statement 3 is creating an object E of Employee class. Statement 4 and 5 is invoking/calling member function through the object E of Employee class.

**Lab assignments:**

- 2.1 Create a class named circle with data member Radius ,pi (constant member) and area. Create following member functions:
- i. to accept the radius from the user
  - ii. to calculate the area of the circle
  - iii. to display the details.

**Algorithm:**

- Step 1: Declare a class CIRCLE  
Step 1.2 Declare its data members radius, pi and area.
- Step 2: Create its member function accept(), to read values for radius from the user.
- Step 3: Create another member function area() to calculate area of the circle.
- Step 4: Create third member function display() to print the value of area.
- Step 5: Define main()  
Step 5.1 Create the object of class CIRCLE.  
Step 5.2 Call the function accept().  
Step 5.3 Call the function area().  
Step 5.4 Call the function display()
- Step 6: End main().



- 2.2 Given that an EMPLOYEE class contains following data members: Employee\_Number, Employee\_Name, Basic, DA, IT, Net\_Sal. Member Functions: to read the data, to calculate Net\_Sal and to print data members. Write a C++ program to read the data of an employee and compute Net\_Sal of the employee (DA= 52% of Basic and Income Tax(IT)=30% of the gross salary).



- 2.3 Write an OOP in C++ to prepare a student Record using class and object with the following data members and member functions:

Data Members: Name, Regn Number, and Marks in three subjects

Member functions: Read, display and average marks







### **LAB 3: Array of object**

#### **Array of object:**

An object of class represents a single record in memory, if we want more than one record of class type, we have to create an array of class or object. As we know, an array is a collection of similar type, therefore an array can be a collection of class type.

#### **Syntax for Array of object:**

---

```
class class-name
{
    datatype var1;
    datatype var2;
    -----
    datatype varN;
    method1();
    method2();
    -----
    methodN();
};
class-name obj[ size ];
```

#### **Example for Array of object:**

---

```
#include<iostream.h>
#include<conio.h>
class Employee
{
    int Id;      char Name[25];      int Age;      long Salary;

    public:
    void GetData()      //Statement 1 : Defining GetData()
    {
        cout<<"\n\tEnter Employee Id : ";
        cin>>Id;
        cout<<"\n\tEnter Employee Name : ";
        cin>>Name;

        cout<<"\n\tEnter Employee Age : ";
```

```

        cin>>Age;
        cout<<"\n\tEnter Employee Salary : ";
        cin>>Salary;
    }
    void PutData()        //Statement 2 : Defining PutData()
    {
        cout<<"\n"<<Id<<"\t"<<Name<<"\t"<<Age<<"\t"<<Salary;
    }
};
void main()
{
    int i;
    Employee E[3];        //Statement 3 : Creating Array of 3 Employees
    for(i=0;i<3;i++)
    {
        cout<<"\nEnter details of "<<i+1<<" Employee";
        E[i].GetData();
    }
    cout<<"\nDetails of Employees";
    for(i=0;i<3;i++)
        E[i].PutData();
}

```

**Output :**

```

Enter details of 1 Employee
    Enter Employee Id : 101
    Enter Employee Name : Suresh
    Enter Employee Age : 29
    Enter Employee Salary : 45000

Enter details of 2 Employee
    Enter Employee Id : 102
    Enter Employee Name : Mukesh
    Enter Employee Age : 31
    Enter Employee Salary : 51000

Enter details of 3 Employee
    Enter Employee Id : 103
    Enter Employee Name : Ramesh
    Enter Employee Age : 28
    Enter Employee Salary : 47000

```

Details of Employees

101	Suresh	29	45000
102	Mukesh	31	51000
103	Ramesh	28	47000

---

In the above example, we are getting and displaying the data of 3 employee using array of object. Statement 3 is creating an array of Employee Emp to store the records of 3 employees.

**Lab assignments:**

- 3.1 Define a STUDENT class with USN, Name and marks in 3 tests of a subject, declare an array of 10 student objects find the average of two better marks for each student. Print USN, Name and average marks of all the students.

**Algorithm:**

Step 1: Create a class STUDENT

Step 1.1 Define its data members USN, name, sub1[3]

Step 1.2 Define its member function accept() to take input.

Step 1.3 Define another member function avg() , compare marks of three tests and find average of two best marks.

Step 1.4 Define third member function display() to display USN, Name and average marks of all the students.

Step 2: Define main()

Step 2.1 Create an array of objects for ten students.

Step 2.2 Call accept()

Step 2.3 Call avg()

Step 2.4 Display()

Step 3: End main()





- 3.2 Write an OOP in C++ to create a class Employee with data members as Name, Employee code and Gross salary. Include member functions to read input and display output of individual objects. Create an array of 10 employees. Display the details of the employees with highest and lowest gross salary.







- 3.3 Write an OOP in C++ to create a class called BOOK with the following data members and member function:  
Data Members: Title, Author, Price, ISBN No.  
Member functions: Read, Display, Search\_book(using ISBN No)  
Create an array of 5 BOOK type objects and demonstrate all member functions.



## LAB 4: Function overloading, passing objects as arguments and returning objects

### Function overloading:

More than one function with same name, with different signature in a class or in a same scope is called function overloading. Signature of function includes :

- Number of arguments
- Type of arguments
- Sequence of arguments

When you call an overloaded function, the compiler determines the most appropriate definition to use by comparing the signature of calling statement with the signature specified in the definitions.

### Example of function overloading:

---

```
#include<iostream.h>
#include<conio.h>
class CalculateArea
{
    public:
    void Area(int r)          //Overloaded Function 1
    {
        cout<<"\n\tArea of Circle is : "<<3.14*r*r;
    }
    void Area(int l,int b)      //Overloaded Function 2
    {
        cout<<"\n\tArea of Rectangle is : "<<l*b;
    }
    void Area(float l,int b)    //Overloaded Function 3
    {
        cout<<"\n\tArea of Rectangle is : "<<l*b;
    }
    void Area(int l,float b)    //Overloaded Function 4
    {
        cout<<"\n\tArea of Rectangle is : "<<l*b;
    }
};
```

```
void main()
{
    CalculateArea C;
    C.Area(5);      //Statement 1
    C.Area(5,3);    //Statement 2
    C.Area(7,2.1f); //Statement 3
    C.Area(4.7f,2); //Statement 4
}
```

**Output :**

```
Area of Circle is : 78.5
Area of Rectangle is : 15
Area of Rectangle is : 14.7
Area of Rectangle is : 29.4
```

---

In the above example, we have four member functions named Area. Statement 1 will invoke the function 1 because the signature of function 1 is similar to the statement 1. Similarly Statement 3 will invoke function 4 because statement 3 is passing two arguments, 1st is of integer type and 2nd is of float type. Function 4 is the only function who is receiving integer and float respectively.

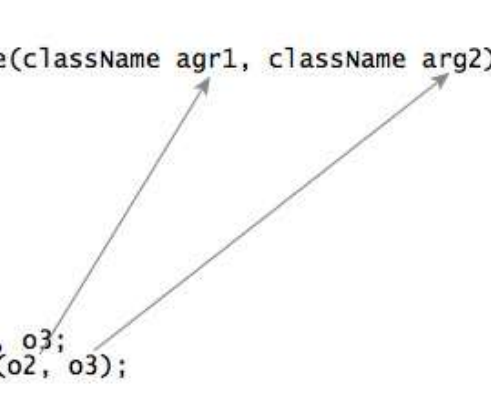
**C++ returning object and passing object as function:**

Object is a variable of class type. An object is used to access the class members. Like normal variable, object can be pass as function argument.

**Syntax to pass objects to a function:**

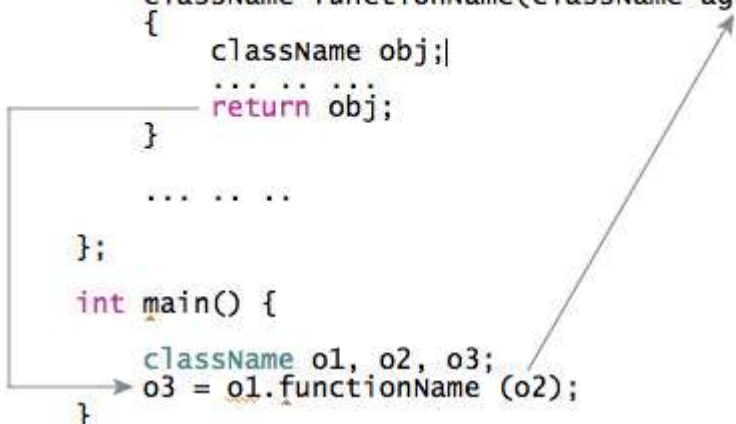
```
class className {
    ... ..
public:
    void functionName(className agr1, className arg2)
    {
        ... ..
    }
    ... ..
};

int main() {
    className o1, o2, o3;
    o1.functionName (o2, o3);
}
```

A diagram illustrating object passing. Two arrows originate from the arguments 'o2' and 'o3' in the function call 'o1.functionName (o2, o3);' within the 'main()' function. These arrows point to the parameters 'agr1' and 'arg2' respectively in the 'functionName' definition of the 'className' class.

Syntax to return an object from the function:

```
class className {  
    ... ..  
public:  
    className functionName(className agr1)  
    {  
        className obj;  
        ... ..  
        return obj;  
    }  
    ... ..  
};  
int main() {  
    className o1, o2, o3;  
    o3 = o1.functionName(o2);  
}
```



Example of passing object & returning object:

---

```
#include<iostream.h>  
#include<conio.h>  
class Rectangle  
{  
    int L,B;  
    Rectangle(int l,int b)  
    {  
        L = l;  
        B = b;  
    }  
    Rectangle()  
    {  
        L = 0;  
        B = 0;  
    }  
    Rectangle Sum(Rectangle Rec)    //function 1  
    {  
        Rectangle temp;  
        temp.L = L + Rec.L;    //Statement 1  
        temp.B = B + Rec.B;    //Statement 2  
        return temp;  
    }  
}
```

```
}  
void Display()  
{  
    cout<<"\nLength : "<<L;  
    cout<<"\nBreadth : "<<B;  
}  
};  
void main()  
{  
    Rectangle R1(5,7),R2(2,3),R3;  
    cout<<"\n\nValues of Rectangle 1 : ";  
    R1.Display();  
    cout<<"\n\nValues of Rectangle 2 : ";  
    R2.Display();  
    R3 = R1.Sum(R2);    //Statement 3  
    cout<<"\n\nValues of Rectangle 3 : ";  
    R3.Display();  
}
```

**Output :**

Values of Rectangle 1 :  
Length : 5  
Breadth : 7

Values of Rectangle 2 :  
Length : 2  
Breadth : 3

Values of Rectangle 3 :  
Length : 7  
Breadth : 10

---

In statement 3, object **R1** is calling member function Sum and passing object **R2** as argument. Function 1 is receiving values of **R2** in **Rec**. In statement 1 **L** is length of calling object **R1** and **Rec**. **L** is length of object **R2**.

Both statement 1 and 2 is adding the length and breadth of Rectangle 1 and 2 respectively and assigning the result to object **temp**.

Statement 3 is receiving the values of object temp in object **R3**.

**Lab assignments:**

- 4.1 Create a class named C\_POWER with a function **power()** to raise a number **m** to the power **n**. The function takes a **double** value for **m** and **int** value for **n**, returns the result correctly. Use a default value of 2 for **n** to make the function to calculate squares when this argument is omitted. Write another function that takes an **int** value of **m**. Both functions should have same name. Write a **main** function that gets values of **m** and **n** from user.

**Algorithm:**

- Step 1: Create a class named C\_POWER
- Step 1.1 Define its member function power() with m and n as arguments. Declare m as double and n as integer.
  - Step 1.2 Assign a default value 2 to n
  - Step 1.3 Define another function power() with m and n arguments and calculate the power.  
Declare m and n as integer.
  - Step 1.4 Declare a member function accept() to take values for n and m and calculate the power.
- Step 2: Define main()
- Step 2.1 Create object of the class.
  - Step 2.2 Call accept().
  - Step 2.3 Call power() appropriately.
- Step 3: End main().





- 4.2 Write a C++ program to create a class called COMPLEX and implement the following by overloading functions ADD() that return a COMPLEX number.  
ADD(a, s2)- where a is an integer (real part) and s2 is a complex number.  
ADD(s1, s2)- where s1 and s2 are complex numbers.





- 4.3 Write a program in C++ to create a class named TIME that stores two time values in hours (float) and minutes (integer). Perform the following operations by using a function add():-
- Add T1+T2 where , T1 and T2 are time objects.
  - Add T1+x where T1 is a TIME object and x is any integer.
  - Return time objects from the member function to main() and display.





## LAB 5: Constructors and destructors

### C++ Constructor:

We can't initialize class data members directly as given below:

```
class Student
{
    int Roll=1;           // Error : Cannot initialize a class member here
    char Name[]="Kumar"; // Error : Cannot initialize a class member here
    float Marks=78.53;    // Error : Cannot initialize a class member here
};
```

Constructor is a special function used to initialize class data members or we can say constructor is used to initialize the object of class.

### Characteristics constructor:

- Constructor name class name must be same.
- Constructor doesn't return value.
- Constructor is invoked automatically, when the object of class is created.

### Types of Constructor:

- Default Constructor
- Parameterized Constructor
- Copy Constructor

**Default Constructor:** Construct without parameter is called default constructor.

### Example of C++ default constructor:

```
#include<iostream.h>
#include<string.h>
class Student
{
    int Roll;
    char Name[25];
    float Marks;

    public:
```

```
Student()    //Default Constructor
{
    Roll = 1;
    strcpy(Name,"Kumar");
    Marks = 78.42;
}
void Display()
{
    cout<<"\n\tRoll : "<<Roll;
    cout<<"\n\tName : "<<Name;
    cout<<"\n\tMarks : "<<Marks;
}
};
void main()
{
    Student S;    //Creating Object
    S.Display();  //Displaying Student Details
}
```

**Output :**

```
Roll : 1
Name : Kumar
Marks : 78.42
```

---

**Parameterized Constructor:** Construct with parameter is called parameterize constructor.

**Example of C++ parameterize constructor:**

---

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Student
{
    int Roll;          char Name[25];          float Marks;

    public:
    Student(int r,char nm[],float m) //Parameterize Constructor
    {
        Roll = r;          strcpy(Name,nm);
        Marks = m;
    }
}
```



```
void Display()
{
    cout<<"\n\tRoll : "<<Roll;
    cout<<"\n\tName : "<<Name;
    cout<<"\n\tMarks : "<<Marks;
}

};

void main()
{
    Student S(2,"Sumit",89.63);
    //Creating Object and passing values to Constructor
    S.Display(); //Displaying Student Details
}
```

**Output :**

```
Roll : 2
Name : Sumit
Marks : 89.63
```

In parameterize constructor, we have to pass values to the constructor through object.

---

**Copy constructor:**

Initialization of an object through another object is called **copy constructor**. In other words, copying the values of one object into another object is called **copy constructor**.

**Example of C++ copy constructor:**

---

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Student
{
    int Roll; char Name[25]; float Marks;
public:
    Student(int r,char nm[],float m) //Constructor 1 : Parameterize Constructor
    {
        Roll = r;
        strcpy(Name,nm);
        Marks = m;
    }
}
```

```
    }  
    Student(Student &S) //Constructor 2 : Copy Constructor  
    {  
        Roll = S.Roll;  
        strcpy(Name,S.Name);  
        Marks = S.Marks;  
    }  
    void Display()  
    {  
        cout<<"\n\tRoll : "<<Roll;  
        cout<<"\n\tName : "<<Name;  
        cout<<"\n\tMarks : "<<Marks;  
    }  
};  
void main()  
{  
    Student S1(2,"Sumit",89.63);  
    Student S2(S1); //Statement 1  
    cout<<"\n\tValues in object S1";  
    S1.Display();  
    cout<<"\n\tValues in object S2";  
    S2.Display();  
}
```

**Output :**

```
Values in object S1  
Roll : 2  
Name : Sumit  
Marks : 89.63
```

```
Values in object S2  
Roll : 2  
Name : Sumit  
Marks : 89.63
```

---

In the above example, Statement 1 is creating an object S2 and passing another object S1 as parameter to the constructor 2.

Constructor 2 will take the reference of object S1 passed by the statement 1 and copy all the values of object S1 to data members associated to the object S2.

**Destructor :**

Constructor allocates the memory for an object. Destructor deallocate the memory occupied by an object. Like constructor, destructor name and class name must be same, preceded by a tilde(~) sign. Destructor take no argument and have no return value.

Constructor is invoked automatically when the object created. Destructor is invoked when the object goes out of scope. In other words, Destructor is invoked, when compiler comes out form the function where an object is created.

**Example of C++ destructor:**

---

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Student
{
    int Roll;          char Name[25];          float Marks;

    public:
    Student() //Default Constructor
    {
        Roll = 4;
        strcpy(Name,"Sumit");
        Marks = 84.56;
    }
    void Display()
    {
        cout<<"\n\tRoll : "<<Roll;
        cout<<"\n\tName : "<<Name;
        cout<<"\n\tMarks : "<<Marks;
    }
    ~Student() //Destructor
    {
        cout<<"\n\tEnd of program.";
    }
};
void main()
{
    Student S;      S.Display();      }
```

**Output :**

Roll : 4  
Name : Sumit  
Marks : 89.56  
End of program.

---

**Constructor overloading:**

More than one constructor with different signature in a class is called function overloading. Signature of constructor includes :

- Number of arguments
- Type of arguments
- Sequence of arguments

When we create an object, the compiler determines the most appropriate constructor to use by comparing the signature of the statement which is creating object with the signature of specific constructor definition.

**Example of constructor overloading:**

---

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Student
{
    int Roll;    char Name[20];    float Marks;
public:
    Student(int r,char nm[],float mks)    // Constructor 1
    {
        Roll = r;
        strcpy(Name , nm);
        Marks = mks;
    }
    Student(int r,float mks,char nm[])    // Constructor 2
    {
        Roll = r;
        strcpy(Name , nm);
        Marks = mks;
    }
    Student(char nm[],int r,float mks)    // Constructor 3
    {
        Roll = r;
```

```
        strcpy(Name , nm);
        Marks = mks;
    }
    void Display()
    {
        cout<<"\n\t"<<Roll<<"\t"<<Name<<"\t"<<Marks;
    }
};
void main()
{
    Student S1(101,"Kumar",78.53);           //Statement 1
    Student S2("Sumit",102,89.27);           //Statement 2
    Student S3(103,67.38,"Kunal");           //Statement 3
    S1.Display();
    S2.Display();
    S3.Display();
}
```

**Output :**

```
101   Kumar   78.53
102   Sumit   89.27
103   Kunal   67.38
```

In the above example, we have three constructors with different sequence of arguments. Statement 1 will invoke the constructor 1 because the signature of constructor 1 is similar to constructor 1. Similarly Statement 2 will invoke constructor 3 because the signature of statement 2 is similar to constructor 3. Statement 2 is passing three arguments, 1st is of string type (character array) ,2nd is of int type and the 3<sup>rd</sup> is float type. And constructor 2 is the only constructor which is receiving string as 1st value, int as 2nd value and float as third value.

**Lab assignments:**

- 5.1 Write a C++ program to create a class called STRING and implement the following operations.

```
STRING s1="SMIT"  
STRING s2="MAJITAR"  
STRING s3=s1+s2 (Use copy constructor).
```

**Algorithm:**

Step 1: Declare class STRING

Step 1.1 Declare its data members name and length.

Step 1.2 Declare default constructor and single parameterized constructors to initialize length and name.

Step 1.3 Declare its member function display() to print the concatenated result.

Step 1.4 Declare a member function join() to concatenate two strings

Step 2: Define main()

Step 2.1 Create object of the class.

Step 2.2 Initialize data members.

Step 2.3 Call join()

Step 2.4 Call display()

Step 3 : End main()







- 5.2 Write a program in C++ to create a class named student, create a constructor and destructor for the class and demonstrate sequence of execution of constructors and destructors.



- 5.3 Write an OOP in C++ to demonstrate overloading of constructors for a class that allows creating initialized and uninitialized array of objects.



## LAB 6: Static data and static member function

### Static class members:

Data members and member functions of a class in C++, may be qualified as static. We can have static data members and static member function in a class.

### Static data member:

It is generally used to store value common to the whole class. The static data member differs from an ordinary data member in the following ways :

- i. Only a single copy of the static data member is used by all the objects.
- ii. It can be used within the class but its lifetime is the whole program.

For making a data member static, we require :

- i. Declare it within the class.
- ii. Define it outside the class.

### For example:

```
class student
{
    static int count; //declaration within class
    -----
    -----
};
```

The static data member is defined outside the class as :

```
int student :: count;    //definition outside class
int student :: count = 0; // definition outside class with initial value
```

**The definition outside the class is a must.**

### Static member function:

A static member function can access only the static members of a class. We can do so by putting the keyword static before the name of the function while declaring it.

### For example:

```
class student
{
    static int count;
    -----
    public :
```

```

-----
static void showcount (void) //static member function
{
    cout<<"count="<<count<<"\n";
}
};

int student ::count=0;

```

Here we have put the keyword static before the name of the function showcount ().

In C++, a static member function differs from the other member functions in the following ways:

- i. Only static members (functions or variables) of the same class can be accessed by a static member function.
- ii. It is called by using the name of the class rather than an object as given below:

Name\_of\_the\_class :: function\_name

**For example:**

student::showcount();

### Lab assignments:

6.1 Write a program in C++ to define a class named Bank. Include the following data members:-

- |   |                              |
|---|------------------------------|
| a. Name of account holder                                 | b. Account number            |
| c. type of account  | d. balance amount in account |
| e. number of transactions in the bank(static data member) |                              |

Create the following member functions:-

- i. To read inputs
- ii. To deposit an amount
- iii. To withdraw amount after checking current balance
- iv. Display number of transactions (use static function).

**Algorithm:**

Step1: Create class Bank

1.1 Data members

Name

Acc\_number

type

balance

no\_of\_tran

1.2 create functions

Input()

deposit()

withdraw()

Display()

Step4: main()

4.1 create object of class Bank

4.2 call input()

4.3 call deposit() and withdraw().

4.4 Display output.







- 6.2 Write an OOP in C++ to demonstrate static members of a class. Include static data members to count the number of objects created for the class



- 6.3 Write an OOP in C++ to demonstrate static members function of a class to pre-initialize private static data members. Also show restrictions that apply on static member functions.



## LAB 7: Friend function

### Friend function:

If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.

The compiler knows a given function is a friend function by the use of the keyword **friend**.

### Declaration of friend function in C++:

```
class class_name
{
    ... ..
    friend return_type function_name(argument/s);
    ... ..
}
```

Now, you can define the friend function as a normal function to access the data of the class. No friend keyword is used in the definition.

```
class className
{
    ... ..
    friend return_type functionName(argument/s);
    ... ..
}
return_type functionName(argument/s)
{
    ... ..
    // Private and protected data of className can be accessed from
    // this function because it is a friend function of className.
    ... ..
}
```

### Example of friend function:

```
#include <iostream>
using namespace std;
class Distance
{
    private:
```

```
        int meter;
        public:
        Distance()
        {
            meter=0
        }
        friend int addFive(Distance); //friend function

};
// friend function definition
int addFive(Distance d)
{
    //accessing private data from non-member function
    d.meter += 5;
    return d.meter;
}
int main()
{
    Distance D;
    cout<<"Distance: "<< addFive(D);
    return 0;
}
Output:
    Distance: 5
```

### Lab Assignments:

- 7.1 Create two classes DM and DB which stores the value in distances. DM stores distance in meters and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB. Use a friend function to carry out the addition operation. The object that stores the results may be a DM object or DB object, depending on the units in which the results are required. The display should be in the format of feet and inches or meters and centimeters depending on the object on display. Use the concept of friend function.

### Algorithm:

Step1: Create class DM

- 1.1 Data members
  - meters
  - centimeters
- 1.2 create functions
  - Input()
  - friend Sum()
  - friend Add ()
  - Display()

Step2: Create class DB

- 1.1 Data members
  - feet
  - inches
- 1.2 create functions
  - Input()
  - friend Sum()
  - friend Add ()
  - Display()

Step3: Define the entire friend function friend Sum (), friend Add ().

Step4: main()

- 4.1 create object of each class
- 4.2 call input()
- 4.3 call sum() and add()
- 4.4 Display output.







- 7.2 Write an OOP in C++ to create a Class Twovalues with two integer type data members and member function to read values in those data members. Create another Class Min\_Max and declare it as friend of Class Twovalues. Min\_Max should have member functions to find the minimum and maximum of two values.





- 7.3 Write an OOP in C++ to demonstrate the advantage of overloading '+' operator using friend function.



## LAB 8: Implementing Stacks And Linked Lists In C++

### Lab assignments:

- 8.1 Write a C++ program to create a class called LIST (linked list) with member functions to :-
- Insert an element at any position.
  - To display the list.

### Algorithm:

Step1: Create class LIST

1.1 Data members

info

link

1.2 create functions

insert() //Insert an element at the position specified.

Display() //to show the list

Step2: main()

2.1 create object of class LIST

2.2 call insert()

2.3 Display the list









- 8.2 Write a C++ program to create a class called LIST (linked list) with member functions to :-
- i. Delete an element from any position.
  - ii. To display the list.

**Algorithm:**

Step1: Create class LIST

1.1 Data members

info

link

1.2 create functions

delete() //delete an element from the position specified.

Display() //to show the list

Step2: main()

2.1 create object of class LIST

2.2 call insert()

2.3 Display the list





- 8.3 Write an OOP in C++ to create a class called STACK with an array of integer type data member and member functions to PUSH/POP elements from the stack and display the elements on the stack.







**LAB 9: Operator overloading:**

An operator is a symbol that tells the compiler to perform specific task. Every operator have their own functionality to work with built-in data types. Class is user-defined data type and compiler doesn't understand, how to use operators with user-defined data types. To use operators with user-defined data types, they need to be overload according to a programmer's requirement.

Operator overloading is a way of providing new implementation of existing operators to work with user-defined data types.

An operator can be overloaded by defining a function to it. The function for operator is declared by using the **operator** keyword followed by the operator.

There are **two types** of operator overloading in C++

- Binary Operator Overloading
- Unary Operator Overloading

**Binary operator overloading:**

Binary operator is an operator that takes two operand (variable). Binary operator overloading is similar to unary operator overloading except that a binary operator overloading requires an additional parameter.

**Binary operators:**

- Arithmetic operators (+, -, \*, /, %)
- Arithmetic assignment operators (+=, -=, \*=, /=, %=)
- Relational operators (>, <, >=, <=, !=, ==)

**Example of binary operator overloading:**

---

```
#include<iostream.h>
#include<conio.h>
class Rectangle
{
    int L,B;
    public:
    Rectangle()    //Default Constructor
    {
        L = 0;    B = 0;
    }
}
```

```

        Rectangle(int x,int y)    //Parameterize Constructor
        {
            L = x;                  B = y;
        }

Rectangle operator+(Rectangle Rec) //Binary operator overloading func.
{
    Rectangle R;
    R.L = L + Rec.L;
    R.B = B + Rec.B;
    return R;
}
void Display()
{
    cout<<"\n\tLength : "<<L;
    cout<<"\n\tBreadth : "<<B;
}
};
void main()
{
    Rectangle R1(2,5),R2(3,4),R3;    //Creating Objects
    cout<<"\n\tRectangle 1 : ";
    R1.Display();
    cout<<"\n\n\tRectangle 2 : ";
    R2.Display();
    R3 = R1 + R2;    Statement 1
    cout<<"\n\n\tRectangle 3 : ";
    R3.Display();
}

```

**Output :**

```

Rectangle 1 :
L : 2
B : 5
Rectangle 2 :
L : 3
B : 4
Rectangle 3 :
L : 5
B : 9

```

In statement 1, Left object R1 will invoke operator+() function and right object R2 is passing as argument.

Another way of calling binary operator overloading function is to call like a normal member function as follows,

---

```
R3 = R1.operator+ ( R2 );
```

---

### **Unary operator overloading:**

Unary operator is an operator that takes single operand(variable). Both increment(++) and decrement(--) operators are unary operators.

### **Example of unary operator overloading:**

---

```
#include<iostream.h>
#include<conio.h>
class Rectangle
{
    int L,B;
public:
    Rectangle()    //Default Constructor
    {
        L = 0;    B = 0;
    }

    void operator++() //Unary operator overloading func.
    {
        L+=2;    B+=2;
    }
    void Display()
    {
        cout<<"\n\tLength : "<<L;
        cout<<"\n\tBreadth : "<<B;
    }
};

void main()
{
    Rectangle R; //Creating Object
    cout<<"\n\tLength Breadth before increment";
    R.Display();
}
```

```
        R++;  
        cout<<"\n\n\tLength Breadth after increment";  
        R.Display();  
    }
```

**Output :**

```
Length Breadth after increment  
L : 0  
B : 0  
Length Breadth after increment  
L : 2  
B : 2
```

---

**Lab assignments:**

- 9.1 Write a program in C++ to overload unary minus operator to change the sign of a number.

**Algorithm:**

Step1: Create class Space

1.1 data member x, y, z

1.2 member function

    getdata() // for taking input

    display() // for display result

    operator-() // operator function used for changing symbol of data members

Step2: main()

2.1 Create object

2.2 call getdata(), operator-(), display().





- 9.2 Write a C++ program to create a class called COMPLEX and implement the following by overloading + operator:-
- i.  $a + s2$ ; where  $a$  is an integer (real part) and  $s2$  is a complex number.
  - ii.  $s1 + s2$ ; where  $s1$  and  $s2$  are complex numbers.
  - iii. Return object and display the result.







- 9.3 Write a C++ program to create a class called STACK using array of integers. Implement the following operations by overloading the operators + and --.
- i) `s1=s1+element`; where `s1` is a object of the class STACK and `element` is an integer to be pushed on top of the stack.
  - ii) `s1=s1--`; where `s1` is a object of the class STACK, -- operator pops the element.

**Note: Handle the STACK empty and STACK full conditions.**





- 9.4 Write a C++ program to create a class called MATRIX using a two-dimensional array of integers. Implement the following operations by overloading the operator== which checks the compatibility of two matrices to be added and subtracted. Perform the addition and subtraction by overloading the operators + and - respectively :-

```
if(m1==m2)
{
    m3=m1-m2;
    m4=m1+m2;
}
else display error
```







- 9.5 Write a C++ program to create a class called DATE. Accept two valid dates in the form dd/mm/yy. Implement the following operations by overloading the operators + and -.
- i.  $\text{no\_of\_days} = d1 - d2$ ; where  $d1$  and  $d2$  are DATE objects.  $d1 \geq d2$  and  $\text{no\_of\_days}$  is an integer.
  - ii.  $d2 = d1 - \text{no\_of\_days}$ ; where  $d1$  is a DATE object and  $\text{no\_of\_days}$  is an integer.





## LAB 10: Template class and template functions

### C++ template :

C++ template is used in situation where we need to write the same function for different data types. For example, if we need a function to add two variables. The variable can be integer, float or double. For this purpose we have to write one function for each data type. To avoid writing the same function for different data types we use template.

There are two types of templates in C++ :

- i. Function template
- ii. Class template

### Function template:

C++ Function templates are those functions which can handle different data types without separate code for each of them.

```
#include<iostream.h>
#include<conio.h>

template <class T>
T Sum(T n1, T n2)           // Template function
{
    T rs;
    rs = n1 + n2;
    return rs;
}

void main()
{
    int A=10,B=20,C;
    long I=11,J=22,K;

    C = Sum(A,B);           // Calling template function
    cout<<"\nThe sum of integer values : "<<C;

    K = Sum(I,J);           // Calling template function
    cout<<"\nThe sum of long values : "<<K;

}
```

**Output :**

The sum of integer values : 30

The sum of long values : 33

To make a function templates, we must write the following statement before function definition.

```
template <class T>
```

Here T is the type name, which is dynamically determined by the compiler according to the parameter passed to function definition.

**Class template:**

Like function templates, we can also use templates with class to make member function common for different data types.

```
#include<iostream.h>
#include<conio.h>
template <class T>
class Addition                                // Template class
{
    public:
    T Add(T, T);
};
template <class T>
T Addition<T>::Add(T n1, T n2)                // Template member function
{
    T rs;
    rs = n1 + n2;
    return rs;
}
void main()
{
    Addition <int>obj1;
    Addition <long>obj2;
    int A=10,B=20,C;
    long I=11,J=22,K;
    C = obj1.Add(A,B);    // Calling template member function
    cout<<"\nThe sum of integer values : "<<C;
    K = obj2.Add(I,J);    // Calling template member function
    cout<<"\nThe sum of long values : "<<K;
}
```

**Output :**

The sum of integer values : 30

The sum of long values : 33

**Lab Assignments:**

- 10.1 Write a program in C++ to create a template function and perform bubble sort.

**Algorithm:**

step1: Declare Template T

step2: Create Class BSORT

2.1 Data Type

T Array

len

2.2 Member Function

BSORT() //constructor

getArray()

Bubble\_Sort()//perform sorting

print\_Array()

step3:main()

3.1 create object

3.2 call getArray()

3.3call Bubble\_Sort()

3.4 call print\_Array()







- 10.2 Write a program, in C++ to create a template class STACK and perform push and pop operations.





- 10.3 Write an OOP in C++ to create a template class ARRAY and implement run time boundary checking by overloading [] operator.



**LAB 11: Inheritance****Inheritance:**

Inheritance means access the properties and features of one class into another class. The class who is going to provide its features to another class will be called base class and the class who is using the properties and features of another class will be called derived class.

**Example of inheritance:**

---

```
#include<iostream.h>
#include<conio.h>
class Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;

    public:
    void GetData()
    {
        cout<<"\n\tEnter Employee Id : ";
        cin>>Id;

        cout<<"\n\tEnter Employee Name : ";
        cin>>Name;

        cout<<"\n\tEnter Employee Age : ";
        cin>>Age;

        cout<<"\n\tEnter Employee Salary : ";
        cin>>Salary;
    }

    void PutData()
    {
        cout<<"\n\nEmployee Id : "<<Id;
        cout<<"\nEmployee Name : "<<Name;
        cout<<"\nEmployee Age : "<<Age;
        cout<<"\nEmployee Salary : "<<Salary;
```

```

    }

};
class Company : public Employee //Statement 1
{
    int RegNo;
    char CName[25];
    public:
    void ReadData()
    {
        cout<<"\n\nEnter Registration No. : ";
        cin>>RegNo;

        cout<<"\n\nEnter Company Name : ";
        cin>>CName;

    }
    void WriteData()
    {
        cout<<"\n\nRegistration No. : "<<RegNo;
        cout<<"\n\nCompany Name : "<<CName;
    }

};
void main()
{
    Company C;        //Statement 2 : Creating Object of Derived Class
    C.GetData();       //Statement 3 : Calling Base Class Method()
    C.ReadData();
    C.PutData();       //Statement 5 : Calling Base Class Method()
    C.WriteData();
}

```

**Output :**

```

Enter Employee Id : 1
Enter Employee Name : Kumar
Enter Employee Age : 29
Enter Employee Salary : 45000

Enter Registration No. : 715
Enter Company Name : TutorialDost

```

Employee Id : 1  
Employee Name : Kumar  
Employee Age : 29  
Employee Salary : 45000  
Registration No. : 715  
Company Name : TutorialDost

Consider the statement 1, we are publically inheriting an Employee class into Company class using colon(:).  
Now, The object of Company class can access the member function **GetData()** and **PutData()** of Employee class.

---

**Derived class can inherit Base class as private, protected or public.**

#### **Types of Inheritance:**

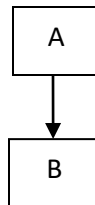
C++ supports **six types** of inheritance as follows:

- Single Inheritance
- Multilevel Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance
- Multipath Inheritance

#### **Single Inheritance:**

A derived class with only one base class is called **single inheritance**.

---

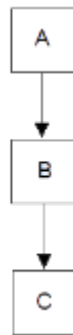


#### **Multilevel Inheritance:**

A derived class with one base class and that base class is a derived class of another is called **multilevel inheritance**.

---



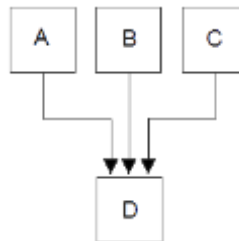


---

### Multiple Inheritance:

A derived class with multiple base classes is called **multiple inheritance**.

---

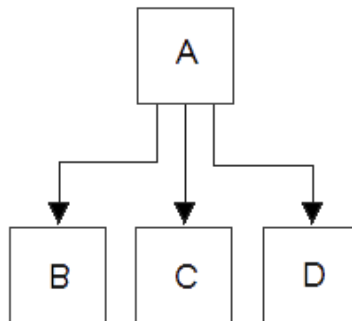


---

### Hierarchical Inheritance:

Multiple derived classes with same base class is called **hierarchical inheritance**.

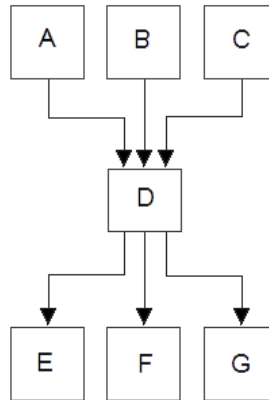
---



### Hybrid Inheritance:

Combination of multiple and hierarchical inheritance is called **hybrid inheritance**.

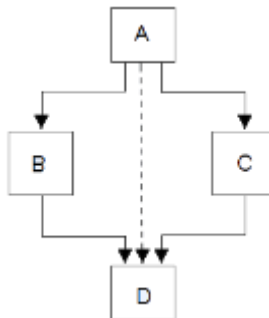
---



### Multipath Inheritance:

A derived class with two base classes and these two base classes have one common base class is called **multipath inheritance**.

---



**Lab Assignments:**

- 11.1 Write a C++ program to create a class called STUDENT with data members USN, Name and Age. Using inheritance, create the classes UGSTUDENT and PGSTUDENT having fields as Semester, Fees and Stipend. Enter the data for at least 5 students. Find the semester wise average age for all UG and PG students separately.

**Algorithm:**

Step1: create a class STUDENT

1.1 data members

Reg, Name, Age

Step2: create a class UG

2.1 data members

Semester, Fees, Stipend

2.2 member function

Get\_data()

Put\_data()

Step3: create a class PG

3.1 data members

Semester

Fees

Stipend

3.2 member function

Get\_data()

Put\_data()

Step4: main()

4.1 create array of objects 5 for UG and PG class.

4.2 calculate average of sum semester wise

4.3 Display the output















- 11.2 Write an OOP in C++ to demonstrate accessing derived class objects' members using base class pointers. Also show the sequence of execution of constructors and destructors when inheritance is involved.

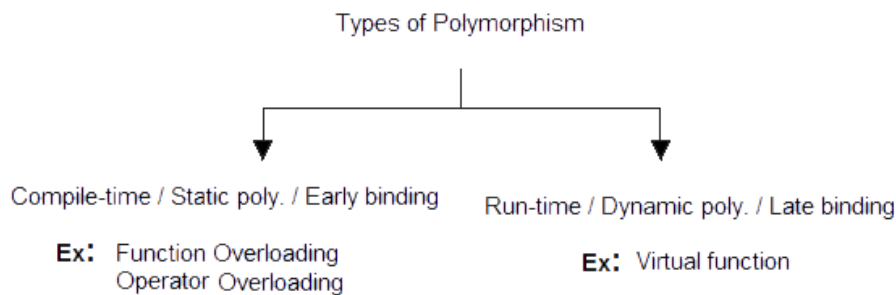


## LAB 12: Polymorphism

### Polymorphism:

Polymorphism means more than one function with same name, with different working. Polymorphism can be static or dynamic. In static polymorphism memory will be allocated at compile-time. In dynamic polymorphism memory will be allocated at run-time. Both function overloading and operator overloading are an examples of static polymorphism. Virtual function is an example of dynamic polymorphism.

- Static polymorphism is also known as early binding and compile-time polymorphism.
  - Dynamic polymorphism is also known as late binding and run-time polymorphism.
- 



---

**Function overloading :** Function overloading is an example of static polymorphism. More than one function with same name, with different signature in a class or in a same scope is called function overloading.

**Operator overloading :** Another example of static polymorphism is Operator overloading. Operator overloading is a way of providing new implementation of existing operators to work with user-defined data types.

**Virtual function :** Virtual function is an example of dynamic polymorphism. Virtual function is used in situation, when we need to invoke derived class function using base class pointer. Giving new implementation of derived class method into base class and the calling of this new implemented function with base class's object is done by making base class function as virtual function. This is how we can achieve "Runtime Polymorphism".

**Example of virtual function:**

```
#include<iostream.h>
#include<conio.h>
class BaseClass
{
    public:
    virtual void Display()
    {
        cout<<"\n\tThis is Display() method of Base Class";
    }
    void Show()
    {
        cout<<"\n\tThis is Show() method of Base Class";
    }
};
class DerivedClass : public BaseClass
{
    public:
    void Display()
    {
        cout<<"\n\tThis is Display() method of Derived Class";
    }

    void Show()
    {
        cout<<"\n\tThis is Show() method of Derived Class";
    }
};
void main()
{
    DerivedClass D;
    BaseClass *B;        //Creating Base Class Pointer
    B = new BaseClass;
    B->Display();         //This will invoke Display() method of Base Class
    B->Show();            //This will invoke Show() method of Base Class
    B=&D;
    B->Display();         //This will invoke Display() method of Derived Class
                        //bcoz Display() method is virtual in Base Class
    B->Show();            //This will invoke Show() method of Base Class
                        //bcoz Show() method is not virtual in Base Class
}
```

**Output :**

This is Display() method of Base Class  
This is Show() method of Base Class  
This is Display() method of Derived Class  
This is Show() method of Base Class

**Lab assignments:**

- 12.1 WAP in C++ to create a class called **Figure**. Use this class to store two double type values that could be used to compute the area of various shapes (i.e. length, breadth for rectangle and base, height for triangle). Derive two specific classes called **Triangle** and **Rectangle** inherited from super class **Figure**. Create a virtual function called **area( )** to compute and display the area of the basic shapes. Redefine this function in derived classes to suit its requirements.

**Algorithm:**

Step1: create a class Figure

1.1 Data Members

ar // variable for storing area

1.2 Member Function

virtual void area()

Step2: create a class Triangle inherits Figure class

2.1 Data Members

base, height

2.2 Member Function

Get\_data() //for taking input from user

area()

Display() //for displaying output

Step3: create a class Rectangle inherits Figure class

3.1 Data Members

length, breadth

3.2 Member Function

Get\_data() //for taking input from user

area() // for calculating area

Display() //for displaying output

Step4: main()

4.1 create a pointer for base class //for runtime polymorphism

4.2 Assign object of other classes to the base class pointer

4.3 By using the pointer calculate area.

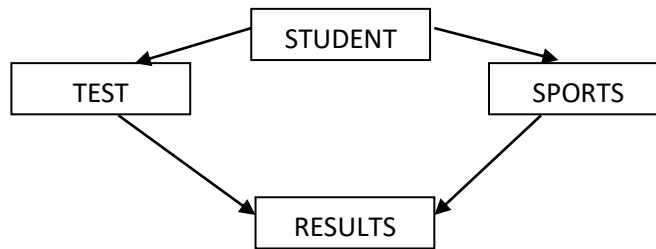
4.4 Display the results







12.2 Write a C++ to implement the following using the concept of inheritance.-



- i. Create data member roll\_no, reg\_no and member function get\_number() to take input and put\_number() to display in the base class STUDENT
- ii. The class Test contains a data member part1 and part2. Redefine the above two functions.
- iii. The class SCORE contains a member score. Redefine get\_number and put\_number ().
- iv. Redefine display function in RESULT class as well to calculate total, where  $TOTAL = part1 + part2 + score$ .

**Note: Implement the concept of run-time polymorphism, and include constructors in all the classes.**





- 12.3 Write an OOP in C++ to create a base class convert with two data members val1 and val2 and two member functions getinit() and getconvert() which return the initial values and converted values. It includes a pure virtual function compute() which must be defined by two derived classes lit\_to\_gal and far\_to\_cel. Each of these class have their own definition of compute()



## **LAB 13: Files in C++**

### **C++ Files:**

At the time of execution, every program comes in main memory to execute. Main memory is volatile and the data would be lost once the program is terminated. If we need the same data again, we have to store the data in a file on the disk. A file is sequential stream of bytes ending with an end-of-file marker.

Types of file supported by C++:

- Text Files
- Binary Files

### **Difference between text file and binary file:**

- Text file is human readable because everything is stored in terms of text. In binary file everything is written in terms of 0 and 1, therefore binary file is not human readable.
- A newline(\n) character is converted into the carriage return-linefeed combination before being written to the disk. In binary file, these conversions will not take place.
- In text file, a special character, whose ASCII value is 26, is inserted after the last character in the file to mark the end of file. There is no such special character present in the binary mode files to mark the end of file.
- In text file, the text and characters are stored one character per byte. For example, the integer value 23718 will occupy 2 bytes in memory but it will occupy 5 bytes in text file. In binary file, the integer value 23718 will occupy 2 bytes in memory as well as in file.

### **The fstream.h Header File:**

C++'s standard library called **fstream**, defines the following classes to support file handling.

- **ofstream class** : Provides methods for writing data into file. Such as, open(), put(), write(), seekp(), tellp(), close(), etc.
- **ifstream class** : Provides methods for reading data from file. Such as, open(), get(), read(), seekg(), tellg(), close(), etc.
- **fstream class** : Provides methods for both writing and reading data from file. The fstream class includes all the methods of ifstream and ofstream class.

**Opening a file using open() member function:**

The open() function takes file-name argument. The purpose of opening the file i.e, whether for reading or writing, depends on the object associated with open() function.

**Example of opening file:**

---

```
ofstream fout;
fout.open("filename"); // Open file for writing
ifstream fin;
fin.open("filename"); // Open file for reading
fstream f;
f.open("filename",mode);
// Using fstream, we can do both read and write, therefore,
// mode specifies the purpose for which the file is opened.
```

**File opening modes:**

Mode	Purpose
ios::in	Open a text file for reading.
ios::out	Open a text file for writing. If it doesn't exist, it will be created.
ios::ate	Open a file and move the pointer at the end-of-file.
ios::app	Open a text file for appending. Data will be added at the end of the existing file. If file doesn't exist, it will be created.
ios::binary	Open file in a binary mode.
ios::nocreate	The file must already exist. If file doesn't exist, it will not create new file.
ios::trunc	If the file already exists, all the data will be lost.

### Reading and writing into file:

We can read data from file and write data to file in four ways.

- Reading or writing characters using `get()` and `put()` member functions.
- Reading or writing formatted I/O using insertion operator ( `<<` ) and extraction operator ( `>>` ).
- Reading or writing object using `read()` and `write()` member functions.

### Lab assignments:

- 13.1 Write a program in C++ to create a file named "costs" and implement the following by creating menu driven program:-
- i. The file should contain two columns "item\_name" and "cost".
  - ii. Insert data under these two columns assuming the file is empty initially.
  - iii. Allow user to add more data as and when he wants.
  - iv. Retrieve the item details if item name is given as an input.
  - v. Display the entire content of the file.

### Algorithm:

- Step 1: Define `main()`.
- Step 2: Create object of `ofstream`.
- Step 3: Create an output file COSTS
- Step 2: Create two columns `item_name` and `cost`.
- Step 3: Insert data into the file using the `ostream` object.
- Step 4: Close the file.
- Step 4: Declare object of `ifstream`.
- Step 5: Reopen COSTS file using `ifstream` object.
- Step 6: Read the content of the file using `getline()`.
- Step 7: Read the contents until EOF.
- Step 8: Display the contents.
- Step 9: Close the file.
- Step 10: End `main()`.







- 13.2 Write a program in C++ to copy the contents of one file into another in reverse order.



- 13.3 Write a program in C++ to change a particular character in a file using command line.





## ADDITIONAL PAGES























