Practical No. 06

Title:- Write a Program using Class and Objects.

Relevant Program Outcome:-
L.L.O 6.1: Define Class and Create objects
L.L.O 6.2: Write a program using class & objects.

Relevant Course Outcome:-
Develope c++ program to solve problems using object oriented approach.

Practical Outcome:-
Write / compile / debug / Execute simple c++ program using class & objects.

Minimum Theorotical Background:-

- Object:-
An object is an instance of a class, representing a specific entity with its own data and behavior.
- Memory Allocation:
When an object is created, memory is allocated for its data members.

- State:-
The state of an object is defined by the current values of its data members.

- Behavior:-
Objects interact with the world through their member functions, which define their behavior.

- Access:-
Object members (data & functions) are accessed using the dot (.) operator.

- Lifetime:- An object's lifetime is the period during which it exists in memory, either automatically

- Encapsulation:-
Objects encapsulate data and the functions that operate on the data, ensuring data integrity and security.

- Class:-
A class is a blueprint for creating objects It encapsulates data and functions that operate on the data.

- Components:-
  - Data members: Variables that hold the data of the class.
  - Member functions: Functions that define the behavior of the class.

- Access Specifiers:-
  - Public: Accessible from outside the class
  - Private: Accessible only within the class itself
  - Protected: Accessible within the class & by derived class.

-

- Special Functions:
  - Constructor: Initializes objects of the class, often setting initial values.
  - Destructor: Cleans up when an object is destroyed.

Syntax:

```
// Class
class ClassName {
  public:
  };
```

```
// Object
ClassName objectName;
```

Program 1

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

class Rectangle {
  public:
    int width, height;
    int area() {
        return weidth * height;
    }
};
```

```
int main()
{
   Rectangle rect;
   rect.width = 5;
   rect.height = 10;

   cout<<" Area of the rectangle: "<< rect.area()
       << endl;
   retuen 0;
}
```

→ Output:-

Area of the rectangle: 50

II▶ Program 2.

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

class Animal {
public
   void eat() {
      cout << " Animal eats. " << endl;
      }
};
Class Dog: Public Animal {
 public:
      void bark() {
```

```
    cout << " Dog barks." << endl;
  }
};
int main()
{
   Dog myDog;
   myDog.eat();
   myDog.bark();

   return 0;
}
```

→ Output:
Animal eats
Dog barks.

II▶ Program 3

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

class Print{
 public:
    void display (int x) {
        cout << " Integer: " << x << endl;
    }
    void display (double x) {
        cout << " Double: " << x << endl;
    }
    void display(const string &x) {
```

```
    cout << " String: " << x << endl;
  }
};

int main(){
   Print printer;
   printer.display(10);
   printer.display(3.14);
   printer.display("Hello");

   return 0;
  }
```

→ Output

Integer : 10
Double : 3.14
String : Hello.

Conclusion: We learnt what is object & class how they stores data & perform actions defined by the class.

20/8/24

Practical No. 07

Title:- Write a Program using array of Objects.

Relevant Program Outcome:-
L.L.07.1: Write a program using class and array of objects.

Relevant Course Outcome:-
Develope c++ program to solve problems using object oriented approach.

Practical Outcome:-
Write / compile / debug / Execute simple c++ program using array of objects.

Minimal Theoretical Background:-

- Arrays of object:

An array of objects is a collection of objects of the same class type, stored in contigous memory locations. It allows managing multiple objects of a class using array indexing.

-Creation:-
An array of objects is declared by specifying the class name as the data type, followed by the array name & size.

Syntax:
class Name arrayName [arraysize];

- Access:-
Objects in the array are accessed using an index, with the first object at index 0 & the last at arraysize -1.

Syntax:
arrayName [index]. memberfunction();

- Initialization:-
  - By invoking a constructor during array declaration.
  - By assigning values to data members individually after creation.

Example:-
- ClassName arrayName [arraySize] = {Obj1, Obj2..};
- arrayName [index]. dataMember = value;

Program

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

class Employee
{
   int id;
```

```
    char name[30];
public:
    void inputEmployeeData();
    void printEmployeeData();
};

Void Employee::inputEmployeeData(){
 cout<<" Enter Id: ";
  cin >> id;
  cout<<" Enter Name: ";
  cin >> name;
}
 void Employee::printEmployeeData(){
    cout<<id<<" ";
    cout << name <<"  ";
    cout << endl;
}
 int main(){
    Employee emp;
    emp.inputEmployeeData();
    emp.printEmployeeData();
    return 0;
}
```

→ Output:

```
Enter Id: 243
Enter Name: abcd

243 abcd
```

Program:

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

class Student
{
  int roll;
  char name[30];
  public:
   void enterdata();
   void displaydata();
};
void Student:: enterdata()
{
  cout << " Enter Roll :" ;
  cin >> roll;
  cout << " Enter Name: ";
  cin >> name;
}
void Student:: displaydata()
{
 cout << roll<< " ";
 cout << name<< " ";
 cout << endl;
}
int main()
{
  Student stud [30];
```

```
int n, i;
cout <<" Enter Number of Students- ";
cin >> n;

for (i=0; i< n; i++)
  stud [i]. enterdata();

cout <<" Student Data - " <<endl;
for (i=0; i<n; i++)
  stud[i]. displaydata();
}
```

→ Output:

```
Enter Number of Students- 2
Enter Roll: 101
Enter Name: pqr
Enter Roll: 102
Enter Name: xyz
Student data-
101  pqr
102  xyz
```

Conclusion: Hence we learnt an array of object provides a convinient way to manage & operate on multiple objects of a class using single array variable.

3/5/24 ⑨

Practical No. 08

Title:- Write a program using object as function argument.

Relevant Program Outcome:-
L.L.O 8.1:- Implement the concept of object as function argument.

Relevant Course Outcome:-
Develope c++ program to solve problems using object oriented approach.

Practical Outcome:-
Write / compile / debug / Execute simple c++ program using object as function argument.

Minimal Theoretical Background:

- Function argument:-
Passing an object as a function argument involves providing an object of a class to a function so that the function can operate on or manipulate the object.

Types: Pass by Value
Pass by Reference
Pass by Pointer.

Benifits:

- Encapsulation: Functions can operate on objects while maintaining the object's enscapsulation.
- Modularity: Functions are be written to handle specific operations on objects. improving code organization & reusability.

Program1

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

class student
{
 public:
 int roll, marks;
 void get()
 {
  cout<<" Enter Roll No. And Marks: ";
  cin >> roll >> marks;
 }
 void put()
 {
  Cout <<" Roll No. : " << roll << endl;
  cout<<" Marks. : " << marks << endl; }
void assign (student s1)
{
  roll = s1.roll;
```

```
    marks = s1.marks;
  }
};
void main()
{
  student s1, s2;
  s1.get();
  s2.assign(s1);
  s1.put();
  s2.put();
  getch();
}
```

→ Output:

```
Enter Roll No. And Marks: 100
85
Roll No.: 100
Marks: 85
Roll No.: 100
Marks.: 85
```

II▶ Program 2:

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

class example
{
  public:
```

```
int a;
void add (example e)
{
    a = a + e.a;
}
};
void main()
{
 example e1, e2;
 e1.a = 50;
 e2.a = 100;
 cout << e1.a << endl << e2.a << endl;
 e2.add (e1);
 cout << e1.a << endl << e2.a << endl;
 getch();
}
```

→ Output:-

50
100
50
150

Conclusion: Hence we learnt to execute C++ program using object as a function argument.

3/9/24

Practical No. 09

Title:- Write a Program using Static Members (variables & Function).

Relevant Program Outcome:-
L.L.O 9.1 Use of Static Data Members & member functions

Relevant Course Outcome:-
Develope c++ program to solve problems using object oriented programing.

Practical Outcome:-
Write / compile / Debug / Execute simple c++ program using static Members.

Minimal Theoritical Background:-

- Static Data Members:-
  - static is a keyword used to create a data member as static.
  - It is a predefined keyword.
  - Static Variables are initialized with zero when an object is created.
  - Static variable is stored in memory only once & all objects share commanlly.
  - Static variable not belongs to only particular object

- Characteristics of Data Members:-
  - Only one copy of static variable is created for the entire class.
  - It is visible only within the class.
  - Static variables are normally used to maintain values common to the entire class.

Syntax:-

```
class className{
static datatype variableName;
Public:
.....
};
datatype className :: staticVariableName;
```

- Static Variable defination needs to be defined outside the class.

- Static Member Function:-

It is a function that belong to the class itself rather than any specific instance of the class. It can be called using the class name directly can be only access static member variables or other static member functions within the class.

- Characteristics of Static Member Functions:-
  - A static member functions can access only other static member.

- They may not be virtual.
- They don't have 'this' pointer.
- They can't be static & non-static version of the same function.

Syntax:-

```
static returntype funname (arglist)
{
    // body of function } // fun^ (static) declaration & // defination.

    className :: static_function_name // fun^ (static) // call by using class // name.
```

Program 1

```
#include <iostream>
using namespacestd;

class Device {
  private:
      static int deviceCount;
      int serialNumber;
  public:
      Device (int sn) {
        serialNumber = sn;
        deviceCount++;
      }
  static int getDeviceCount() {
```

Output of Program 1

Device Serial Number: 101
Device Serial Number: 102
Device Serial Number: 103

Total Devices Created: 3

Output of Program 1

Device Serial Number: 101


```
    return deviceCount;
}

void displaySerialNumber() const {
    cout << " Device Serial Number: " << serialNumber
         << endl;
}
};
int Device::deviceCount = 0;
int main() {
    Device d1(101);
    Device d2(102);
    Device d3(103);

    d1.displaySerialNumber();
    d2.displaySerialNumber();
    d3.displaySerialNumber();
    cout << " Total Devices Created: " << Device::getDevice
         Count() << endl;
    return 0;
}
```

11▶ Program-2

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

class Building {
 private:
```

Output of Program 2:

People in the building: 3

People in the building after more entries: 5

```
    static int peopleCount;
public:
    static void enter(){
    peopleCount++;
    }  peopleCount++;
    static int getPeopleCount(){
        return peopleCount;
    }
};
int Building::peopleCount = 0;
int main(){
    Building::enter();
    Building::enter();
    Building::enter();
    cout<<"\n People in the building: "<< Building::get
        PeopleCount() <<endl;
    Building::enter();
    Building::enter();
    cout<<"\n People in the building after more entries:"
    <<Building::getPeopleCount() <<endl;
    return 0
}
```

Conclusion: By Performing this practical we learned that what is static variable & function & how they are commonly used in entire class.

01/10/24

Output of Program 2:

People in the building: 3
People in the building after more entries: 5

Practical No. 10

Title:- Write a Program using Friend Function.

Relevant Program Outcome:-
L.L.O. 10.1: Write a program using friend function.

Relevant Course Outcome:-
Develope c++ program to solve problems using object oriented programming.

Practical Outcome:-
Write / compile / Debug / Execute simple c++ program using friend Function.

Minimal Theoretical Background:-

- Friend Function:-

- Friend is a predefined keyword.
- The private members can't be accessed from outside the class.
- Sometimes there could be sitiuation where we want to share one function between two classes, then friend is used.
- friend function can access private data of class.
- friend function is not in scope of class.
- friend keyword is written before the function declaration.

Characteristics of friend function:
- It is not in the scope of the class.
- It cannot be called by using object. It can be invoked like normal function.
- It can be declared either in public or private.
- Object name & dot operator used for accessing data members inside the friend function.

Syntax:-

```
class className
{ - - - - -
  Public:
  friend returntype funName(arguments);
};
```

Program 1

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

class Box {
   int length;
public
   Box() {
      int l;
      cout<<" Enter the length of box " <<endl;
      cin>>l;
       length = l;
   }
```

```
    friend void doubleLength(Box& b);

    void displayLength() {
        cout<<"Length of the box: "<<length<<endl;
    }
};
void doubleLength(Box& b) {
    b.length *= 2;
}
int main()
{
    Box b;
    b.displayLength();
    doubleLength(b);
    b.displayLength();

    return 0;
}
```

Output:

```
** Enter the length of box: 5
Length of the box: 5
Length of the box: 10
```

Program 2:

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

```
class Counter {
 int count;
public
  Counter() {
      int c;
      cout << " Enter the number of count" << endl
      cin >> c;
      count = c;
    }
  friend void resetCount (Counter& c);
  void displayCounnt () {
      cout << " Current Count: " << count << end;
    }
};
void resetCount ( Counter& c) {
   c.count = 0
   cout << "Count after calling friend fun:" << c.co
      << end; }
int main() {
    Counter counter;
    counter.displayCount();
    resetCount (counter);
    return 0;
  }
```

Output:- Enter the number of count
Current Count: 9
Count after calling friend fun: 0.

Conclusion: We learnt that how can we acc
private data members outside the
using friend function

Practical No. 101

Title:- Write a Program using Constructor & destructor.

Relevant Program Outcome:-
L.L.O 11.1: Use of constructor to initialize objects.
L.L.O 11.2: Write a Program using constructors & destructors

Relevant Course Outcome:-
Develope c++ program to solve problems using object oriented programing.

Practical Outcome:-
Write / Compile / Debug / Execute simple c++ program using static Members.

Minimal Theoretical Background:-

- Constructor:
A constructor is a special method in object-oriented programming that is automatically called when an object of a class is created. The main purpose of a constructor is to initialize the objects properties or allocate resources needed for the object.

- Characteristics of Constructor:
1. Automatic invocation
2. No return type

3. Same name as class
4. Overloading
5. Initializing.

Syntax:-

```
class ClassName{
    public:
// constructor    ClassName(){
                    // code
                  }
};
```

- Destructor:

A destructor is a special method that is automatically called when a object is destroyed or goes out of scope. Its main purpose is to free resources that were acquired by the object, such as memory, file handles, or network connections.

- Charecteristics of Destructor:

1. Automatic invocation
2. No return type or parameters.
3. Only one destructor
4. Same name as class with a tilde (~)
5. Resource cleanup.

Syntax:

```
class ClassName{
    public:
        ~ClassName(){ // cleanup code
        }
};
```

Program 1:

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

class FileHandler {
public:
    FileHandler() {
     cout<<" File handling started..." <<endl;
     cout <<" File opened seccessfully." <<end<<endl;
    }
    ~FileHandler() {
       cout<<" File closed " <<endl;
     }
    void writeData(){
     string userData;
     cout<<" Enter your data (single word)" ;
     cin >> userData;
     cout<<" writing data: " <<userData<<end;
     cout<<" Data writen successfully." <<end;
    }
};
 int main() {
    FileHandler fh;
    fh.writeData();
    return 0;
  }
```

→ Output:

```
file handling started
file opened successfully
Enter your data: Hello!
Writing data
Data written successfully. Hello
File closed.
```

Program 2:

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

class Rectangle{
    int length, breadth;
public:
    Rectangle(){
        int l,b;
    cout<<" Enter length breadth" <<endl;
    cin >>l >>b;
    length = l;
    breadth = b;
    cout<<"Rectangle created" <<endl;
    cout<<"with length: " <<length <<endl;
    cout<<" & breadth: " <<breadth <<endl;
    }
    ~Rectangle(){
    cout<<" Rectangle with" <<end;
    cout<<" length: " <<length <<end;
```

```
    cout << " is destroyed " << end;
}
    int area(){
        return length * breadth;
    }
};

int main(){
    Rectangle rect1;
    cout<< "Area = " << rect1.area() << endl;
    return 0;
}
```

Output:

```
Enter length breadth:  5  6
Rectang Created.
with length:  5
& breadth : 6
Area = 30
Rectangle with
 length: 5
 & breadth: 6
 is destroyed.
```

01/10/24

Conclusion: Hence we performed the practical using constructors & destructors.

Practical No. 12

Title:- Write a Program using Constructor Overloading.

Relevant Program Outcome:-

LLO 12.1: Apply the logic to implement different types of constructor in single program.

Relevant Course Outcome:-

Develope c++ program to solve problems using object oriended programming.

Practical outcome:-

Write / Compile / Debuge / Execute simple c++ program using constructor overloading.

Minimal Theoretical Background:-

Constructor Overloading :-

Constructor Overloading the refers to the ability to define multiple constructors in a class with different parameters. This allows objects to be initialized in various ways depending on the arguments passed when the object is created.

Types of constructor overloading:-

1. Default Constructor: A constructor that takes no parameters and initializes the object with default values.

2. Parameterized Constructor: A constructor that accepts one or more parameters to initialize the object with specific values.
3. Copy Constructor: A constructor that initializes an object using another object of the same class.

- Characteristics of Constructor Overloading:-
1. Multiple Constructors
2. Same names.
3. Overloaded by parameters
4. Flexibility:

Syntax:

```
class ClassName {
public:
    ClassName(){
        // code
    }
    ClassName (int a, int b) {
        //code
    }
    ClassName (const ClassName &obj) {
        //code
    }
};
```

Program 1:

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

class Rectangle {
    int length, width;
  public:
        // constructor1
    Rectangle (int side) {
        length = width = side;
        cout<<" Square created with side length: "
            << side <<endl;
      }
     // constructor 2
    Rectangle (int l, int w) {
        length = l;
        width = w;
        cout <<" Rectangle created with dimensions: "
            << length <<" x" << width <<endl;
     }

   int area() {
       return length * width;
     }
   };
int main() {
   Rectangle square(4);
   Rectangle rectangle( 5, 3);
   cout << " Area of square: " << square.area() <<endl;
```

```
    cout<<" Area of rectangle: " << rectangle.area()
        <<endl;
    return 0;
}
```

→ Output:-

```
Square created with side length: 4
Rectangle created with dimensions: 5x3
Area of square: 16
Area of rectangle: 15.
```

II▶ Program 2:

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

class Point {
    int x,y;
public:
    Point() {
        x=0;
        y=0;
        cout<<" Point created at origin (0,0)."<<endl;
    }
Point (int xCoord, int yCoord) {
        x = xCoord;
        y = yCoord;
        cout<<"Point created at (" <<x<<", "<<y<<")."
            <<endl;
    }
```

```
    void display (){
      cout<<" Point coordinates: ("<<x<<", "<<y<<")."
          <<endl;
      }
};

int main(){
    Point p1;
    p1.display();
    Point p2(10,20);
    p2.display();
    return 0;
}
```

→ Output:

```
Point Created at origin(0,0).
Point Coordinates at: (0,0).
Point Created at: (10,20).
Point Coordinates at: (10,20).
```

Conclusion: Hence we learnt what is constructor Overloading & its use in c++ from this practical.

01/10/24

Practical No. 13

Title:- Write a program to perform following string operation using pre-defined string functions:-

a) String Concatenation
b) String Comparison
c) Find position of an character in a given string
d) String reversing

Relevant Program Outcome:-

L.L.O 13.1:- Understand various predefined string functions.
L.L.O 13.2:- Implement program using predefined string functions.

Relevant Course Outcome:-

Develope ctt program to solve problems using object oriented programming.

Practical Outcome:-

Write / Compile / Debug / Execute simple ctt program using pre-defined string functions.

Minimal Theoritical Background:

- String:-

A string is a sequence of characters used to represent text. Strings are often treated as an object in programing & come with various pre-defined methods to manipulate the text, such as concatanation, comparison

finding a character's position & reversing.

- String Operations:

a) String Concatanation: It is the process of joining 2 or more strings together

b) String Comparison: It is the process of comparing two strings to check if they are the same, or to determine their alphabetical order.

c) Find Position of character:- This operation finds the index of the first occurrence of a specific character in a string.

d) String Reversing:- It is the operation of reversing the order of characters in a string.

Program 1:

```
// String Concatanation
#include <iostream>
#include <string>

int main()
{ string baseDrink, flavor, specialIngredient;

cout << " Enter a flavor (e.g., Vanilla, Caramel): ";
getline( cin, baseDrink);
```

Output of 1st program:-

Enter the Base Drink (e.g., Tea, Coffee): Tea
Enter the Flavor (e.g., Vanilla, Caramal): Vanilla
Enter a Special Ingridient (e.g., Cimmanon, Honey): Honey

Your unique Coffee shop menu Item is: Tea with Vanilla and a Hint of Honey.

```
cout << "Enter a flavor (e.g., Vanilla, Caramel): ";
getline(cin, flavor);

cout << "Enter a special ingredient (e.g., Cinnamon, Honey): ";
getline(cin, specialIngredient);

string menuItem;

menuItem.append(BaseDrink).append(" with ").append
(specialIngredient);

menuItem.append(" and a hint of ").append(specialIn-
gredient);

cout<<"In Your unique coffee shop menu item is:" <<
  menuItem<< endl;
  return 0;
}
```

Program 2:

```
// String Comparison
#include<iostream>
#include<string>
using namespace std;

int main(){
    string caller1, caller2
    cout << "User 1, enter your favourite caller: ";
    getline(cin, caller1);
```

Output of 1st program:-

Enter the Base Drink (e.g., Tea, Coffee): Tea
Enter the flavor (e.g., Vanilla, Caramel): Vanilla
Enter a special ingredient (e.g., Cinnamon, Honey): Honey


```
cout << " Enter a flavor (e.g., Vanilla, Caramel): ";
getline (cin, flavor);

cout << " Enter a special ingredient (eg. Cinnamon,
Honey): ";
getline (cin , specialIngredient);

string menuItem;

menuItem .append (BaseDrink). append(" with") append
(specialIngredient);

menuItem.append (" and a hint of "). append (specialIn-
gredient);

Cout<<"\nYour unique coffee shop menu item is:" <<
 menuItem<< endl;
 retuen 0;
}
```

Program 2:

```
// String Comparison.
# include<iostream>
# include <string>
using namespace std;

int main(){
    string1 color1, color2
    cout <<" User 1, enter your favorite color: ";
    getline(cin, color1);
```

Output of 2nd Program:

User1. Enter Your favourite color: Red
User2. Enter Your favourite color: Red

Wow! Both of you like Red, You are compatible friends!

Output of 2nd Program:


```
cout<<" User 2, enter your favorite color: ";
getline (cin, color2);

if (color1 == color2) {
    cout<<" Wow! Both of you like " <<color1
        <<". You are compatible friends!" <<endl;
} else
{
    cout<<" You both have different tastes, but
            that makes things interesting!" <<endl;
}
return 0;
}
```

Program 3:

```
// Find position of an character in a string.
#include <iostream>
# include<string>
using namespace std;

int main(){
    string password;
    cout <<" Enter a password: ";
    getline (cin, password);

    int position = -1;
    for ( int i=0; i< password.length(); i++) {
            position = i;
            break; }
    }
```

Output of 3rd Program:

Enter a Password: @Admin
Good! The '@' character is found at position: 0.

```
        cout << " Good! The '@' character is at
             position: " << position << endl;
        }
    else {
    cout << " The password must contain the '@' characte
            for added security." << endl;
        }
    return 0;
}
```

Program 4:

```
// string Reversing
#include <iostream>
#include <string>
using namespace std;

int main(){
    string sentence;
    cout << " Enter a sentence: ";
    getline(cin, sentence);

    string result;
    int position = 1;
    string word;
    for (int i = 0; i <= sentence.length(); i++)
    {
        if (sentence[i] == ' ' || sentence[i] == '\0')
        { if (position % 2 != 0) {
                reverse (word.begin(), word.end());
            }
```

Output of 4th Program:

Enter a Sentence: Welcome to the World of Computing

Sentence after Reversing odd-position words:
emoclew to ent World to Computing.

Output of 4th Program:

Enter a Sentence: Welcome to the World of Computing

Sentence after Reversing odd-position words:
enocleW to eht World fo Computing


```
        result += word;
        if (sentence[i] == ' ')
        {   result += ' ';
        }
        word = "";
        position++;
      }
      else {
        word += sentence[i];
      }
    }
    cout << "Senctence after reversing odd-position words:"
         << result << endl;
    return 0;
}
```

Conclusion:- Hence, we learned to preform string operatoins using pre-defined string functoins.

15/10/24

Practical No. 14

Title:- Write a program to perform following string operations without using pre-defined string functions.
i) String Concatenation
ii) String Comparison
iii) Find position of an character in a given string.
iv) String reversing.

Relevant Program Outcome:-
L.L.O 14.1. Understand various predefined string functions.
L.L.O. 14.2. Implement program without using predefined string functions.

Relevant Course Outcome:-
Develop c++ Program to solve problems using object oriented Programming.

Practical Outcome:-
Write / Compile / Debug / Execute simple c++ program without using pre-defined string functions.

Minimal Theoretical Background:-

- Strings can also be manipulated without predefined functions.

1. String Concatenation
To concatenate two strings string1 & string2, you can create a new string & copy each character from both strings.

* Algorithm:
1. Initialize an empty string e.g result
2. Loop through each character of string1 & add it to result.
3. Loop through each character of string2 & add it to result.
4. Return Result.

2. String Comparison
To compare two strings, you can check if each character of both strings is equal until you find a mismatch or reach the end of one of the strings.

* Algorithm:
1. Loop through the characters of both strings simultaneously
2. If characters are different, return false.
3. If one string ends but other doesn't, return false.
4. If all characters match, return true.

3. Find Position of a Character in a given string:
To find the position of a character in a string, you can loop through the string & check each character.

* Algorithm:
1. Loop through each character of the string.
2. If you find the character, return its index.
3. If the loop completes & the character is not found, return -1.

4. String Reversing.
To reverse a string, you can create a new string & add charctees from the original string in reverse order.

* Algorithm:
1. Initialize an empty string e.g reversed.
2. Loop through the original string from the last character to the first.
3. Append each character to the reversed.

Program 1
```
// String Concatenation
#include <iostream>
using namespace std;

int main(){
    string baseDrink, flavour, specialIngridient;
    cout <<" Enter the base Drink (e.g., Tea, Coffee):";
    getline(cin, baseDrink);
    cout<<" Enter the flavour (e.g., Vanilla, Caramel):";
    getline(cin, flavour);
    cout<<" Enter a Special Ingridient (e.g., Cinnamon,
          Honey): ";
    getline(cin, specialIngridient);
```

```
string menuItem = baseDrink + " with " + flavor +
                  " & a Hint of " + specialIngridient;
cout<<"\n Your unique coffee shop Menu is: "<<menuItem
    <<endl;
return 0;
}
```

Output:

```
Enter the Base Drink (e.g., Tea, toffee): Tea
Enter the flavour (e.g. Vanilla, Caremal): Vanilla
Enter special Ingridient (e.g., Cinnamon, Honey): Honey

Your unique coffe shop Menu is: Tea with Vanilla &
                                a Hint of Honey.
```

Program 2

```
// String Comparison.
#include <iostream>
using namespace std;

int main(){
    string color1, color2;
    cout<<" User 1, Enter your favourite color: ";
    getline(cin, color1);
    cout<<" User2, Enter your favourite color: ";
    getline(cin, color2);
    if(color1==color2){
        cout<<"Wow! Both of you like "<<color1
            <<"! You are compatible Friends!"<<endl;
    }
```

```
else {
    cout << " Your both have different choices,
            But that make things Interesting! " .
        << endl;
    }
return 0;
}
```

→ Output:

User1, enter your favourite color: Red
User2, enter your favourite color: Red
Wow! Both of you like Red. You are compatible Friends.

Program 3

```
// find position of string.
#include <iostream>
using namespace std;

int main()
{
  string password;
  cout << " Enter a password: ";
  getline( cin, password );
  int position = -1;
  for( int i = 0; i < password.length(); i++) {
     if (password[i] = '@')
     { position = i;
       break;
     }
  }
```

```
        position"<<position <<endl;
    }
    else {
        cout <<" The password must contain the '@'
                character for added security." <<endl;
    }
    return 0;
}
```

→ Output:

```
Enter a password: gpp@pune
Good! The '@' character is found at position: 3
```

Program 4:

```
// reverse string:
#include <iostream>
using namespace std;

int main() {
    string sentence;
    cout << " Enter a sentence: ";
    getline (cin, sentence);
    string result;
    int position = 1;
    string word;
    sentence += ' ';
    for (int i = 0; i < sentence.length(); i++)
{   if (sentence [i] == ' ')
    { if (position % 2 != 0)
```

```
if (position != -1) {
    cout << " Good! The '@' character is found at
            position" << position << endl;
    }
else {
    cout << " The password must contain the '@'
            character for added security." << endl;
    }
return 0;
}
```

→ Output:

```
Enter a password: gpp@pune
Good! The '@' character is found at position: 3
```

II) Program 4:

```
// reverse string:
#include <iostream>
using namespace std;

int main() {
    string sentence;
    cout << " Enter a sentence: ";
    getline (cin, sentence);
    string result;
    int position = 1;
    string word;
    sentence += ' ';
    for (int i = 0; i < sentence.length(); i++)
{   if (sentence [i] == ' ')
    { if (position % 2 != 0)
```

```
{ for (int j = word.length() -1; j>=0; j--)
  {
    result += word[j];
  }
}
else
{ result += word;
}
result += ' ';
word.clear();
position++;
}
else
{ word += sentence[i];
}
}
cout << " Sentence after Reversing odd-position words: ";
return 0;
}
```

Output:

Enter a sentence: Welcome to the world of Coding.
Sentence after Reversing odd-position words: emoclew to eht world fo Coding.

Conclusion: Olence I come to learn how can we perform string operatoins without using string functions.

15/10/24

Practical No. 15

Title:- Write a Program using single Inheritence.

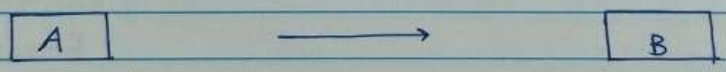
Relevant Program Outcome:-
L.L.O 15.1:- Understand the concept of Inheritence.
L.L.O 15.2:- Impliment single Inheritence.

Relevant Course Outcome:-
Develop c++ program to solve problems using object oriented programing.

Practical Outcome:-
Write / compile / Debug / Execute simple c++ program using single Inheritence.

Minimal Theoretical Background:-

- Inheritence:
  - Inheritence is a one of the major important propeeties of OOP.
  - To create new class by using the properties to old class is known as inheritence.
  - Newly created class is known as derived class.
  - Old class is known as Base or super class.
  - Inheritence also shows reusability.



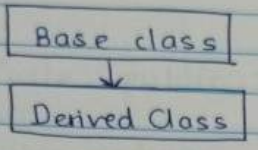
old / Base / super / parent class new / derived / child class.

Inheritence.

- Single inheritance:

- A derived class from only one base class is called as single inheritance.

- Single inheritance is one of the types of inheritance



- In above figure, we can see derived class B is created from only one base class A.

Syntax:

```
class derived_class : visibility_mode base_class
{
    // members of derived class.
};
```

Program 1:

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

class shape {
  protected:
      string name;
```

```
public:
  Shape (string n) : name(n) {
  }
  void displayshape()
  {
    cout <<" Shape: " << name << endl;
  }
};

class Circle: public Shape{
  private:
    double radius;
  public:
    Circle (string n, double r) : Shape (n), radius(r)
    {
    }
    void calculateArea() {
      double area = 3.14 * radius * radius;
      displayShape();
      cout << "Radius" << radius << ", Area:" << area
          << endl;
    }
};
int main()
{
  string name;
  double radius;

  cout <<" Enter Shape Name (e.g., Circle): ";
  getline (cin, name);
```

```
    cout << " Enter Radius: ";
    cin >> radius;

    Circle c (name, radius);
     c. calculateArea();
    return 0;
}
```

Output:

```
Enter Shape Name (e.g., Circle): Circle
Enter Radius: 3
Shape: Circle
Radius: 3, Area: 28.26
```

Program 2:

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

class Device {
   public:
      void turnOn()
     {  cout << "Device is Now ON. " << endl;
      }
      void turnOff()
     {
       cout << " Device is now OFF." << endl;
     }
};
```

```
class Smartphone : public Device {
  public:
    void makeCall (const string& contact)
    {
      cout <<" Calling" <<contact <<"...." <<endl;
      cout<<" Call completed."<<endl;
    }

    void sendText (const string& contact, const string&
                   message)
    {
      cout<<"Sending text to " << contact <<": \""<<
          message <<"\"" <<endl;
    }
};

int main()
{
    Smartphone myPhone;
    string contactName, messageContent;

    myPhone.turnOn();
    cout <<" Enter Contact name to call: ";
    getline (cin, contactName);
    myPhone .makeCall (contactName);
    cout <<" Enter message content: ";
    getline (cin, messageContent);
    myPhone.sendText (contactName, messageContent);
    myPhone. turnOff();
    return 0;
}
```

Output:

```
Device is now ON.
Enter contact name to call: Jack
Calling Jack.....
Call completed.
Enter contact name to send a message: Jill
Enter message content: Hello Jack! Jill Here..
Sending Text to Jill: "Hello Jack! Jill Here.."
Device is now OFF.
```

Conclusion: Hence I come to learn how to use single inheritance and its uses in this program.

24/10/24

Practical No. 16

Title:- Write a Program using multilevel Inheritance.

Relevant Program Outcome:-
L.L.O. 16.1: Understand the concept of Inheritance.
L.L.O. 16.2: Implement multilevel inheritance

Relevant Course Outcome:-
Develop C++ program to solve problems using object oriented programing.

Practical Outcome:-
Write / Compile / Debug / Execute simple C++ program using multilevel Inheritancee.

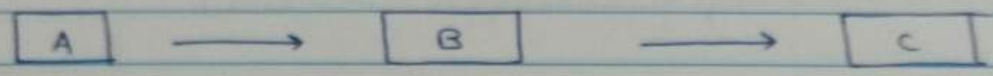
Minimal Theoretical Background:-

- Multilevel Inheritance:
Multilevel inheritance is a type of inheritance where a class is derived from another derived class, forming a chain of inheritance across multiple levels. It enables a class to inherit properties and behaviours from both its immediate parent & higher-level ancestors.

- Characteristics:
- Each derived class inherits all the accessible members (public & protected) from its immediate

overide base class methods.
- A derived class can access the members of its base class based on the access specifies (public protected, private).

* To create new class from another derived class is known as multilevel inheritance.



Base Class derived Class derive derived Class

Multilevel Inheritance.

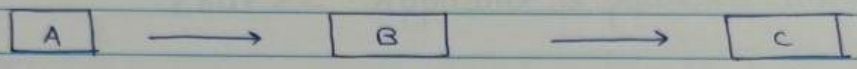
- Above figure shows class C is derived from another derived class B.

Syntax:

```
class A    // Base class A
{ ====
};
class B : public A   // B derived from A
{ =====
};
class C : public B   // C derived from B
{ =====
};
```

base class & from all the base classes above it.
- The derived classes can add new properties or overside base class methods.
- A derived class can access the members of its base class based on the access specifies (public, protected, private).

* To create new class from another derived class is known as multilevel inheritance.



Base Class derived Class derive derived Class

Multilevel Inheritance.

- Above figure shows class C is derived from another derived class B.

Syntax:

```
class A    // Base class A
{ ----
};
class B : public A   // B derived from A
{ -----
};
class C : public B   // C derived from B
{ ----
};
```

Program 1

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

class Appliance {
public:
    void powerOn() {
        cout << "Appliance is powered on." << endl;
    }
};
class WashingMachine : public Appliance {
public:
    void wash(const string& program) {
        cout << "Washing machine is washing clothes
            using the " << program << " program." << endl;
    }
};
class SmartWashingMachine : public WashingMachine {
public:
    void connetToWifi() {
        cout << "Smart washing machine is connected to
            wi-fi." << endl;
    }
};
int main() {
    SmartWashingMachine mySmartWasher;
    string washProgram;
    mySmartWasher.powerOn();
```

```
cout << "Select a wash program (Normal, Quick, Heavy,
        Delicate, Rinse): ";
getline (cin, washProgram);

if (washProgram == "Normal" || washProgram == "Quick"
  || washProgram == "Heavy" || washProgram == "Delicate"
  || washProgram == "Rinse") {
  mySmartWasher.wash(washProgram());
  mySmartWasher.connectToWifi();
  cout << "Your washing machine is ready to go!" << endl;
}else {
  cout << "Invalid wash program. Please enter a valid
        program: Normal, Quick, Delicate, or Rinse."
      << endl; }
  return 0;
}
```

Output:

```
Applience is powered on.
Select a wash program (Normal, Quick, Heavy, Delicate,
 Rinse):
Washing machine is washing clothes using the Normal program.
Smart washing machine is connected to Wi-Fi
Your washing machine is ready to go!
```

Program 2:

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

class Animal{};
```

```
public:
    void generalInfo()
    {
        cout<<" Pets bring joy to people's lives."<<endl;
    }
};

class Dog : public Animal
{ public: void info()
    { cout<<" Dog: Loyal. Lifespan 10-15 years.\n";
    }
};

class Cat : public Animal
{ public : void info()
    { cout<<" Cat : Affectionate, lifespan 12-18 years,
                loves to cuddle. \n";
    }
};
class Fish : public Animal
{ public : void info()
    { cout<<"Fish : Quiet, lifespan 5-10 years. \n";
    }
};
class Bird : public Animal
{ public : void info()
    { cout<<" Bird: Agile & vibrant, lifespan 10-30 years,
                can fly. \n";
    }
};
```

```
class PetStore : public Dog, public Cat, public Fish,
                 public Bird , public Rabbit
{ public:
      void displayPetInfo (int petChoice) {
        switch (petChoice) {
            case 1 : Dog :: Info() ; break;
            case 2 : Cat :: info(); break;
            case 3 : Fish :: info(); break;
            case 4 : Bird :: info(); break;
            case 5 : Rabbit :: info(); break;
            default : cout <<"Invalid choice." <<endl;
        }
   }
};
int main(){
    PetStore store;
    int petChoice;
    cout <<" Welcome to the Pet Store!\n";
    cout<<" 1. Dog\n 2. Cat\n . Fish\n 4. Bird\n 5. Rabbit
          \n pet number: ";
    cin >> petChoice;
    store.displayPetInfo (petChoice);
    return 0;
}
```

Output:

Welcome to the Pet Store!

1. Dog
2. Cat
3. Fish

4. Bird
5. Rabbit
Enter pet number : 1
Dog: Loyal . lifespan 10-15 years.

Conclusion: Olence we learnt multilevel inheritance in c++.

24/10/24

Practical No.17

Title: Write a Program using Multiple Inheritance.

Relevant Program Outcome:-
L.L.O 17.1 :- Understand the concept of Inheritance.
L.L.O 17.2 :- Implement multiple inheritance.

Relevant Course Outcome:-

Develop c++ program to solve problems using object oriented programming.

Practical Outcome:-

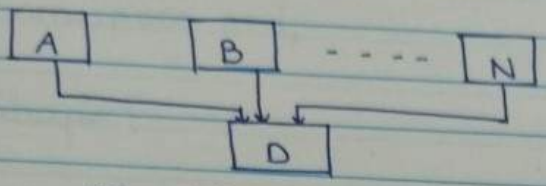
Write / Compile / Debug / Execute simple c++ program using multiple Inheritance.

Minimal Theoretical Background:-

- Multiple Inheritance:

Multiple inheritance in c++ allows a class to inherit from more than one base class. This means that a derived class can inherit properties & behaviours from base classes. However, it can lead to ambiguity if two base classes have methods with the same name. This ambiguity is usually resolved by specifying which base class's method should be used or by overridding the method in the derived class.

* To create new class from more than one old class is known as multiple inheritance.



Multiple Inheritance.

- Above fig. shows class D is derived from base class A, B .... etc.

Syntax:

```
class BaseClass1 {
    //Member of BaseClass1
};
class BaseClass2 {
    //Members of BaseClass2
};
class DerivedClass : public BaseClass1,
public BaseClass2 {
        //Members of DerivedClass
};
```

- DerivedClass inherits from both BaseClass1 & BaseClass2.
- Members from both base classes are available in DerivedClass.

Program 1:

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

class Shape{
 public:
     virtual void calculateArea() = 0;
 };
class Dimensions{
protected:
     double side;
public:
     void getInput(){
        cout<<"Enter side length of the square: ";
        cin >> side;
     }
};
class square : public Shape, public Dimensions{
 public:
     void calculateArea() overcide {
        double area = side * side;
        cout<<" Area of Square: "<<area<<endl;
     }
};
int main(){
     Square square;
     square.getInput();
     square.calculateArea();
     return 0;
}
```

Output:
Enter side length of the square: 4
Area of Square: 16

Program 2:

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

class Flyer {
public:
    void fly() {
        cout << " Flying in the sky. " << endl;
    }
};
class Swimmer {
public:
    void swim() {
        cout << "Swimming in the water. " << endl;
    }
};
class Duck: public Flyer, public Swimmer {
public:
    void quack() {
        cout << " Duck say: Quack! " << endl;
    }
};
int main() {
    Duck myDuck;
    int choice;
    cout << " What do you want the duck to do? " << endl;
```

```
cout<<" 1. Fly " <<endl;
cout <<" 2. Swim" <<endl;
cout <<" 3. Quack"<<endl;
cout<<" Enter your choice (1-3): ";
cin>> choice;

switch (choice){
    case1:
        myDuck();
        break;
    case2
        myDuck.swim();
        break;
    case3:
        myDuck.quack();
        break;
    default:
        cout<<"Invalid choice. Please enter a number
            between 1 & 3."<<endl;
    }
return 0;
}
```

→ Output:

What do you want the duck to do?

1. Fly
2. Swim
3. Quack

Enter your choice (1-3): 1

Flying in the sky.

Practical No. 18

Title:- Write a Program using Virtual Base Class.

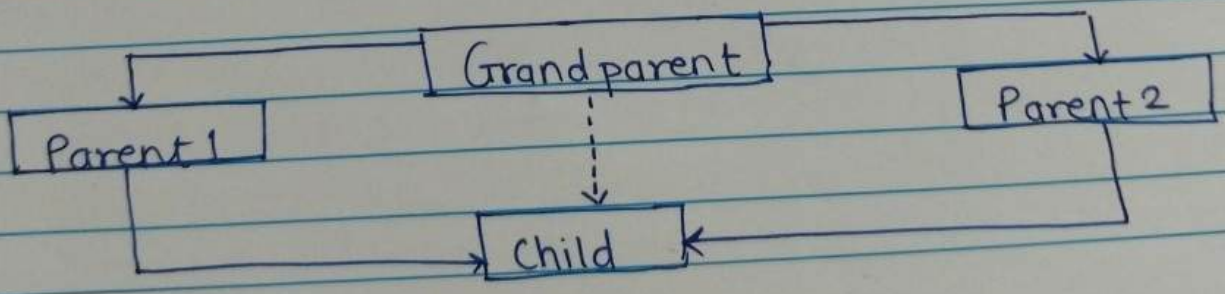
Relevant Program Outcome:-
L.L.O. 18.1:- Understand the concept of diamond problem.
L.L.O. 18.2:- Implement hybrid inheritance.

Relevant Course Outcome:-
Develop c++ program to solve problems using object oriented programming.

Practical Outcome:-
Write / Compile / Debug / Execute simple c++ program using Virtual Base Class.

Minimal Theoretical Background:-

- Virtual Base class:
- Virtual is a predefined keyword
- sometimes in situations, all three types of inheritance multilevel, multiple & hierarchical are involved.
- This is shown in below figure.



- In above figure child has two base classes parent1 & parent2 which is derived from common base class Grandparent.
- Child class inherits the properties of grandparent class from parent1 & parent2.
- It means members of grandparent class present twice in child class.
- Compiler gets confuse which method will call.
- To avoid this problem, we can use virtual base class concept

Syntax:

```
class A  // grandparent
{ ::::
};
class B1 : virtual public A  // parent 1
{ ::::
};
class B2 : public virtual A  // parent 2
{ ::::
};
class C : public B1, public B2  // child
{
  // only one copy of A will be inherited.
};
```

Program 1:

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

class PowerDevice {
  public:
    PowerDevice() {
        cout << " Power Device created" << endl;
    }
    void checkPower() {
        cout << " Power supply ok" << endl;
    }
};
class Phone: virtual public PowerDevice {
  public:
    Phone() {
        cout << " Phone created " << endl;
    }
};
class Laptop: virtual public PowerDevice {
  public:
    Laptop() {
        cout << " Laptop created" << endl;
    }
};
class HybridDevice : public Phone, public Laptop {
```

```
public:
    HybridDevice(){
        cout<<"HybridDevice created"<<endl;
    }
};
int main(){
    HybridDevice hybrid;
    hybrid.checkPower();
    return 0;
}
```

Program 2:

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

class Identify() {
    cout << " Identity created \n";
}
void showIdentity (const string & name) {
    cout <<" Name: " << name << endl;
}
};
class passport : virtual public Identity {
public :
    Passport(){
        cout<<" Passport created \n";
    }
};
class DriverLicense : virtual public Identity{
public:
```

```
    DriverLisense(){
        cout<<" Driver Liscense created\n";
    }
};
class citizen : public Possport , public Driverliscense{
public:
    Citizen (const string& name){
    cout << "Entei created\n";
    showIdentity(name);
    }
};
int main(){
    string name;
    cout<< " entce your name:" << endl;
    cin >> name;
    Citizen citezen(name);
    return 0;
}
```

Conclusion: Olence, we learnt what is virtual base class & how to use it.

8/11/24

Practical No. 19

Title:- Write a Program for Operator Overloading (unary & binary operator)

Relevant Program Outcome:-

L.L.O 19.1: Understand the concept of Polymorphism

L.L.O. 19.2: Write a programs to implement the concept of operator overloading.

Relevant Course Outcome:-

Develope c++ program to solve problems using object oriented programming.

Practical Outcome:-

Write / Compile / Debug / Execute simple c++ program using ~~static~~ ~~mer~~ operator overloading.

Minimal Theoretical Background:-

- Operator Overloading:-

To assign a new meaning to the existing operator is known as operator overloading.

- Operator Overloading assign special meaning to an operator.
- Operator Overloading is a compile time polymorphism

- Overloading Unary operator:
  - When operator takes one operand for doing operation, then it is called unary operator.
  - Let us consider unary minus operator
  - Unary operator takes one Operand.
  - This operator changes the sign of an operand.

examples: +, -, ++, --, !, ~

Syntax:

```
return_type operator <operator> ();
```

- Overloading Binary Operator:-
  - It refers to the ability to redefine the funtionality of binary operators for user defined types.
  - This allows object of thes types to use operators in a natural way, as you would with built-in types.
  - The left operand is the calling object, & the right operand is passed as an argument.

examples: +, -, *, /

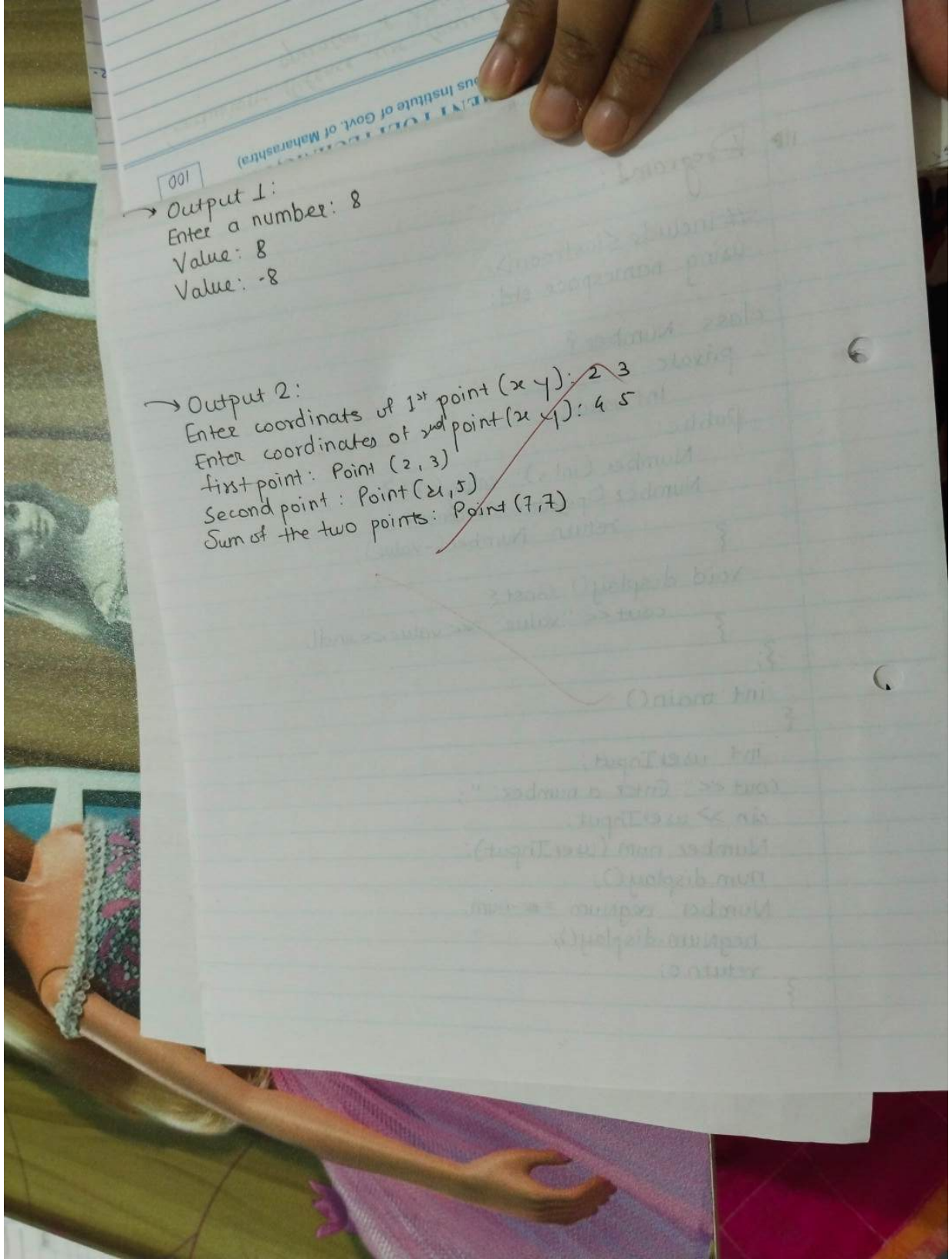
Syntax:

```
return_type operator OP (const Type &rhs);
```

Program 1:

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

class Number {
  private:
      int value;
  public:
      Number (int v): value(v) { }
      Number Operator -() const {
            return Number(-value);
      }
      void display() Const {
            cout << "value: " << value << endl;
      }
};
int main()
{
  int userInput;
  cout << " Enter a number: ";
  cin >> userInput;
  Number num (userInput);
  num.display();
  Number negNum = -num;
  negNum.display();
  return 0;
}
```


001
Output 1:
Enter a number: 8
Value: 8
Value: -8
Output 2:
Enter coordinats of 1st point (x y): 2 3
Enter coordinates of 2nd point (x y): 4 5
first point: Point (2, 3)
Second point: Point (x1, 5)
Sum of the two points: Point (7, 7)

Output 1:

```
Enter a number: 8
Value : 8
Value : -8
```


Program 2:

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

class Point {
    private:
      int x,y;
public:
    Point (int xVal, int yVal): x(xVal), y(yVal){}
    Point operator+ (const Point& other) const {
          return Point(x + other.x, y + other.y);
    }
void display () const {
      cout<<" Point(" <<x <<", "<<y<<")"<<endl;
    }
};
int main(){
    int x1, y1, x2, y2;
    cout<<" Enter coordinates of 1st point (x,y): ";
    cin >>x1>>y1;
    cout<<" Enter coordinates of second point (x,y): ";
    cin>> x2>>y2;
    Point p1(x1, y1), p2(x2, y2);
    Cout<<" Sum of the two points: ";
     p3.display;
    return 0;
}
```

Practical No. 20

Title:- Write a Program for Operator Overloading using friend function.

Relevant Program Outcome:-
L.L.O 20.1: Understand the concept of polymorphism.
L.L.O 20.2: Write a programs to implement the concept of operator overloading using friend function.

Relevant Course Outcome:-
Develop C++ program to solve problems using object oriented programming.

Practical Outcome:-
Write / Compile / Debug / Execute simple C++ program using constructor overloading using friend function.

Minimal Theoretical Background:

Operator Overloading:-
Operator Overloading is feature in C++ that allows operators to have user-defined meanings for operands of user-defined types (like object of classes).

- Friend Function:
  - A friend function is a non-member function that can access the private and protected members of a class.
  - In operator overloading, friend functions are used when we need to overload an operator where the left-hand operand is not an object of the class (or when access to private members is required).

- Types of Operator Overloading:

1. Unary Operator Overloading:
   - Unary operators like, -, ++, or --, operate on a single operand.
   - The friend function for unary operators will only take one argument (the object on which it operates).

2. Binary Operator Overloading:
   - Binary operators like +, -, *, / work on two operands.
   - The friend function for binary operators takes two arguments: one for the left-hand operand & one for the right-hand operand.

Program 11

11▶ Program 2:

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

class Point {
  private:
    int x, y;
  public:
    Point(int xVal, int yVal) : x(xVal), y(yVal){}

    friend Point operator + (const Point& p1, const
     Point& p2);
    void display() const {
      cout << "Point (" << x << ", " << y << ")" << endl;
    }
};
Point operator + (const Point& p1, const Point& p2) {
  return Point(p1.x + p2.x, p1.y + p2.y); }
int main()
  int x1, y1, x2, y2;
  cout << "Enter coordindes of point (x,y)";<< endl;
  cin >> x1 >> y1;
  cout << "Enter coordinates of 2nd point(x y):";
  cin >> x2 >> y2;
  Point p1 (x1, y1), p2(x2, y2);
  cout << "First point: "; p1.display();
  cout << "Second point: "; p2.display();
  Point p3 = p1 + p2;
  cout << "Sum of the two points: "; p3.display();
  return 0;
}
```

Practical No. 21

Title:- Write a Program using 'this' Pointer.

Relevant Program Outcome:-
P.L.O 21.1 : Understand the concept of pointer.
P.L.O 21.2 : Implement this pointer.

Relevant Course Outcome:-
Develop C++ program to solve problems using object oriented programming.

Practical Outcome:-
Write / Compile / Debug / Execute simple C++ program using 'this' pointer.

Minimal Theoretical Background:-

'This' Pointer:
In C++, the this pointer is an implicit parameter to all non-static members functions.
It is a pointer that holds the address of current object (the object that invokes the member functions).
this pointer is useful when you need to refer to the calling object inside a member function, particularly variables & function parameters.

Practical No. 21

Title:- Write a Program using 'this' Pointer.

Relevant Program Outcome:-
L.L.O 21.1 : Understand the concept of pointer.
L.L.O 21.2 : Implement this pointer.

Relevant Course Outcome:-
Develop C++ program to solve problems using object oriented programming.

Practical Outcome:-
Write / Compile / Debug / Execute simple C++ program using 'this' pointer.

Minimal Theoretical Background:-

- 'This' Pointer:
  - In C++, the this pointer is an implicit parameter to all non-static members functions.
  - It is a pointer that holds the address of current object (the object that invokes the member functions).
  - this pointer is useful when you need to refer to the calling object inside a member function, particularly variables & function parameters.

- Use of this Pointer:

1. To differentiate between instance variables and parameters
   - When the parameter name of the member variable name are the same, this pointer helps in distinguishing them.

2. Returning the current object:
   - In functions that return the object itself (like in chained function calls), this pointer can be used to return the current object.

3. Accessing data members & member functions:
   - It helps access the data members & member functions of the current object from within a member function.

Program 1:

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

class Point {
    private:
        int x, y;
    public:
        Point(int xVal, int yVal) : x(xVal), y(yVal) {}
        Point * setX(int xVal) {
            this -> x = xVal;
```

```
    return this;
}
void display() const {
    cout << "Point(" << x << "," << y << ")" << endl;
}
};
int main() {
    int xVal, yVal;
    cout << "Enter x coordinate: ";
    cin >> xVal;
    cout << "Enter y coordinate: ";
    cin >> yVal;
    Point p(0,0);
    p.setX(xVal)->setY(yVal);
    p.display();
    return 0;
}
```

→ Output:

```
Enter x coordinate: 4
Enter y coordinate: 5
Point (4, 5)
```

Program 2:

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

class Number{
    private:
        int value;
    public:
```

```
Number (int v): value(v) {}
bool isSameAs(const Number& other) const {
    return this -> value == other.value;
}
void display() const {
    cout << "Value: " << value << endl;
}
};
int main() {
    int v1, v2, v3
    cout << "Enter value for 1st no. ";
    cin >> v1;
    cout << "Enter value for 2nd no. ";
    cin >> v2;
    cout << "Enter value for 3rd no. ";
    cin >> v3;
    Number n1(v1), n2(v2), n3(v3);
    n1.display();
    n2.display();
    n3.display();
    if (n1.isSameAs(n2)) {
        cout << "n1 & n2 are the sam" << endl;
    } else {
        cout << "n1 & n2 are different." << endl;
    }
    if (n1.isSameAs(n3)) {
        cout << "n1 & n3 are the same" << endl;
    } else {
        cout << "n1 & n3 are different" << endl;
    } return 0; }
```

Conclusion: Learner are performed the practical using this platform.

(An Autonomous Institute of Govt. of Maharashtra)

Practical No. 22

Title:- Write a Program using Virtual Function.

Relevant Program Outcome:-
L.L.O 22.1: Understand the concept of function Overloading
L.L.O 22.2: Implement this pointer.

Relevant Course Outcome:-
Develope c++ program to solve problems using object oriented programming.

Practical Outcome:-
Write / Compile / Debug / Execute simple c++ program using Virtual function.

Minimal Theoretical Background:

- Virtual Function:-
A virtual function is a member function in a base class that is declared using the virtual keyword & is intended to be overridden in derived classes, enabling dynamic (runtime) polymorphism.

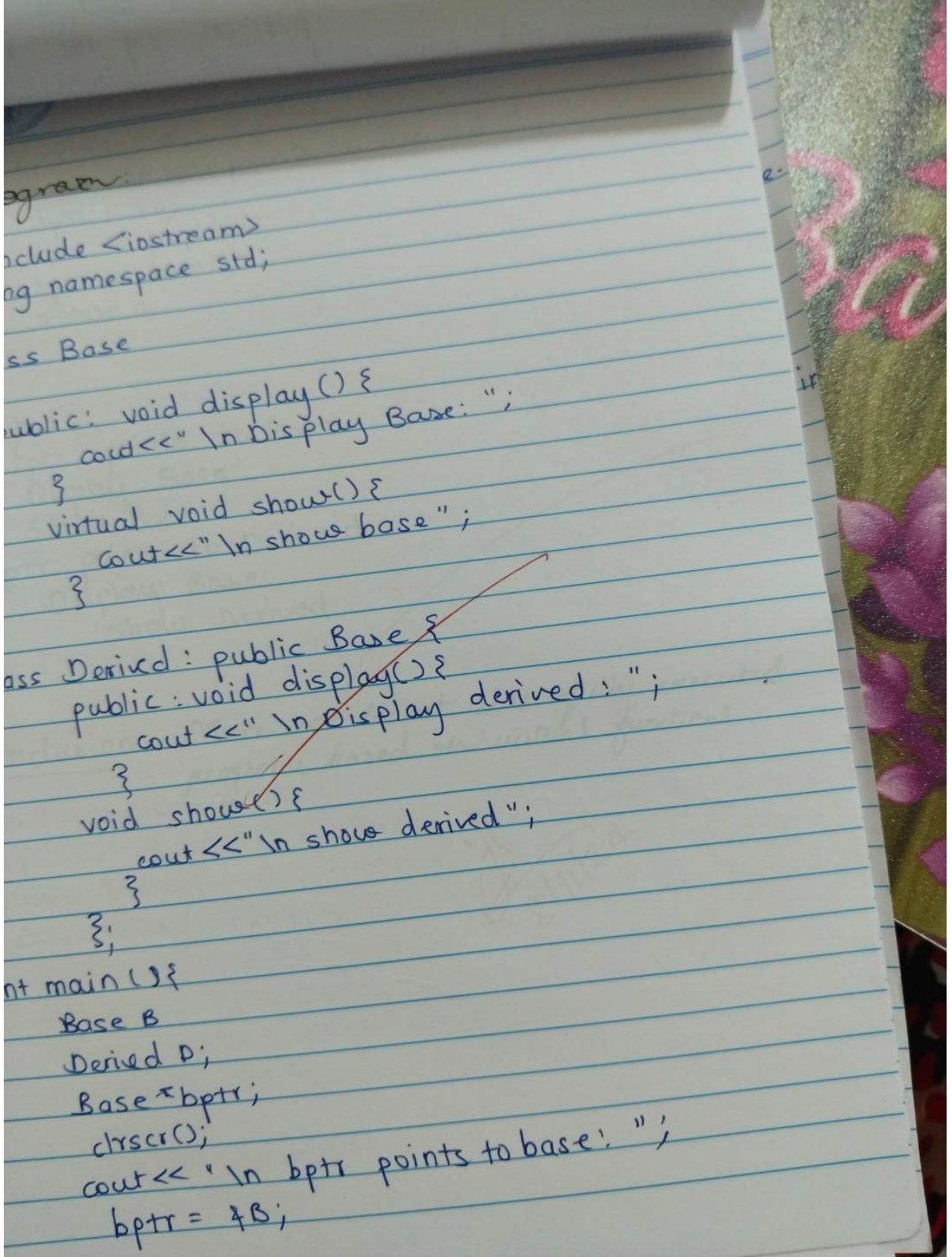
- Rules for virtual functions:

1. The virtual function cannot be static members.
2. The virtual functions are accessed by using object pointer.

3. It can be friend of another class
4. Virtual constructor cannot be created, but virtual destructor can be created.
5. A base pointer can point to any type of derived object, but derived pointer cannot points to the base class.
6. If a virtual function is defined in the base class, then it is not needed to redefined in the derived class.
7. In a base class virtual function must be defined, even although it may not be used.
8. The declaration of base class virtual function & all the derived class functions must be the same.

Syntax:-

```
class Base {
public:
virtual void functionName()
  {
  }
};
// derived class overrides the virtual fun^n.
class Derived : public Base {
public:
  void functionName() override {
  }
};
```



```
ogram:
nclude <iostream>
ng namespace std;

ss Base

ublic: void display() {
    cout<<" \n Display Base: ";
  }
  virtual void show(){
    cout<<" \n show base ";
    }

ass Derived : public Base {
  public : void display(){
    cout <<" \n Display derived : ";
    }
  void show(){
    cout <<" \n show derived ";
    }
  };
nt main(){
  Base B
  Derived D;
  Base *bptr;
  clrscr();
  cout << " \n bptr points to base: ";
  bptr = &B;
```

Program

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

class Base
{
  public: void display() {
     cout<<"\n Display Base: ";
    }
    virtual void show() {
      cout<<"\n show base";
    }
};
class Derived : public Base {
    public : void display() {
       cout <<"\n Display derived : ";
      }
    void show() {
       cout <<"\n show derived";
      }
    };
int main() {
    Base B
    Derived D;
    Base *bptr;
    clrscr();
    cout << '\n bptr points to base: ";
     bptr = &B;
```

```
bptr->show();
cout<<"\n bptr points to derived ";
bptr = &D;
bptr->display();
bptr->show();
return 0;
```

utput:

bptr points to Base.
Display Base:
Show Base
bptr points to Derived
Display Base:
Show Derived

onclusion: Hence we learnt successfully executed programs based on virtual functions.

8/11/24

```
bptr->show();
cout<<"\n bptr points to derived ";
bptr = &D;
bptr->display();
bptr->show();
return 0;
}
```

→ Output:

bptr points to Base.
Display Base:
Show Base
bptr points to Derived
Display Base:
show Derived

Conclusion: Hence we learnt successfully executed programs based on virtual functions.

8/11/24

Practical No. 23

Title: Write a program to implement type conversion concept.

Relevant Program Outcome:-
L.L.O. 23.1: Understand conversion of basic to class type, class type to basic type, class type to class type.

Relevant Course Outcome:-
Develop c++ program to solve problems using object oriented programming.

Practical Outcome:-
Write / Compile / Debug / Execute simple c++ Program using type conversion concept.

Minimal Theoretical Background:

Types Conversion: Type conversion in c++ is the process of changing a variable from on type to another.

This can happen using two ways
- Implicitly - Automatically done by the compiler
- Explicitly - Manually done by the user.

Types of Type Conversion:

1) Base to Class Type
2) Class to Basic Type
3) Class to class Type.

1) Basic to Class type

Syntax:- className(data_type variable)
{ //code }

2) Class to Basic Type

Syntax:- operator typename(){ //code }

3) Class to class type

Syntax:- class dest {
public : destination(source obj) {
//conversion code
};
class source { //members };

Program 1.

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

class Distance {
    int meters;
public: Distance() { }
    Distance (int m) {
        meters = m; }
    void display() {
        cout << meters << "meters" << endl;
    } };
int main() {
    Distance d1, d2;
    int length = 10;
    d1 = length
    d2 = Distance(length * 2);
    d1.display();
    d2.display();
    return 0; }
```

Program 2:

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

class Distance {
    int meters
```

```
public: Distance (int m) {
    meters=m;
}
Operator int() {
    return meters;
}};
int main() {
    distance d1 =20;
    int meters = d1;
    cout<<" meters : "<<meters;
    return 0;
}
```

Program 8:

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

class Distance {
    public: int meters;
    Distance(){
    meters = 5; }};
class length {
    public :int centimeters;
    length (int m){
        centimeters = cm
    }
    length (Distance d) { centimeters = d.meters * 100;
    }
    void display () {
```

```
    cout << centimeters;
};
int main(){
  Distance d1;
  Length l1 = d1;
  l1.display();
  return 0;
}
```

Conclusion: We understood the concept of type conversion & executed program.

8/11/24

Practical No. 24

Title:- Write a Program For file Processing.

Relevant Program Outcome:-
LLO 24.1: Understan the concept of file Processing.

Relevant Course Outcome:
solve problems using object oriented programming. Develop c++ program to

Practical Outcome:-
simple c++ Program using type conversion concept. Write / compile / Debug / Execute

Minimal Theoretical Background:-

File processing in c++ refers to the handling of files to perform operation such as creating, reading, writing which & closing files. c++ provides the fstream, library which contains the necessary classes & four functions to work with files.

The file processing operations include
1) file input (Reading): Reading data from a file.
2) File output (writing): Writing data to a file
3) file manipulating: Deleting, updation or appending to the file.

- Classes in fstream library:-
1) ifstream (input file stream)
2) ofstream (output file stream)
3) fstream (can be used for both reading & writing)

- Operations (common):-
1) Opening a file
2) ofstream outfile("file.txt")
3) ifstream infile("file.txt")
4) Reading & writing data
5) Crossing a file.
6) outfile.close();
7) infile.close();
8) writing: outfile<<" "
9) reading: infile>>" "

11▶ Program:

```
#include <iostream>
#include <fstream>

int main(){
    string filename = "example.txt";
    string line;
    ofstream outfile(filename);
    if(outfile.is_open())
    {
outfile<<"This is a file processesing exampe in
    c++ \n";
outfile<<" Waiting...";
outfile<<"Easy!!!";
outfile<<"";
outfile.close()
    cout<<"Data written to "<<filename;}
else {
    cout<<" unable to open the file for
writting";
}
    ifstream infile(filename);
if(infile.is_open()){
    cout<<"\n reading data from "<<filename;
while (getline (infile,line){
    cout<<"\n reading";
    }
    getch();}
```