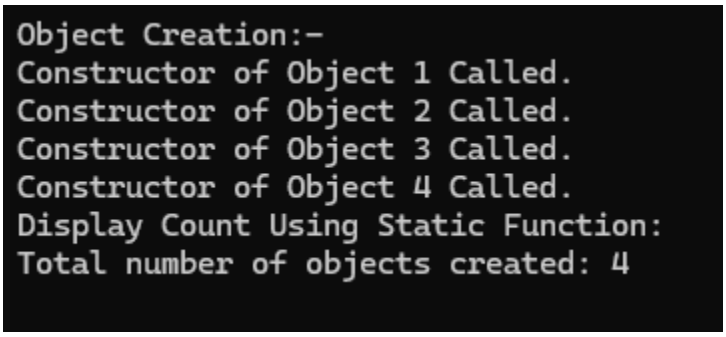
1. **Write a program to define a class that uses static data members and static member functions to count and display the number of objects created.**

Source Code:

```
#include<iostream>
using namespace std;
class Counter {
private:
static int count;
public:
Counter() {
count++;
cout << “Constructor of Object “ << count <<“ Called.”<< endl;
}
static void displayCount() {
cout << “Total number of objects created: “ << count << endl;
}
};
int Counter::count = 0;
int main(){
cout << “Object Creation:-” << endl;
Counter obj1;
Counter obj2;
Counter obj3;
Counter obj4;
cout << “Display Count Using Static Function:” << endl;
Counter::displayCount();
return 0;
}
```

Output:



2. Write a program to create a class that uses a copy constructor to copy data from one object to another.

Source Code:

```
#include <iostream>

using namespace std;

class Student {

private:

int roll;

string name;

public:

Student(int r, string n) {

roll = r;

name = n;

cout << “Parameterized constructor called.” << endl;

}

Student(const Student &s) {

roll = s.roll;

name = s.name;

cout << “Copy constructor called.” << endl;

}
```

```
void display() {
cout << “Name: “ << name << “, Roll Number: “ << roll << endl;
}
};
int main() {
int r;
string n;
cout << “Enter student name: “;
getline(cin, n);
cout << “Enter roll number: “;
cin >> r;
cout << “\n--- Creating Original Object ---” << endl;
Student s1(r, n);
cout << “\n--- Creating Copy Using Copy Constructor ---” << endl;
Student s2 = s1;
cout << “\n--- Displaying Objects ---” << endl;
cout << “Original Object: “;
s1.display();
cout << “Copied Object: “;
s2.display();
return 0;
}
```

Output:

```
Enter student name: Suraj Nepal
Enter roll number: 48

--- Creating Original Object ---
Parameterized constructor called.

--- Creating Copy Using Copy Constructor ---
Copy constructor called.

--- Displaying Objects ---
Original Object: Name: Suraj Nepal, Roll Number: 48
Copied Object: Name: Suraj Nepal, Roll Number: 48
```

3. Write a program to dynamically allocate and deallocate memory for a single object and an array of objects using the new and delete operators.

Source Code:

```
#include <iostream>

using namespace std;

class Student {

private:

string name;

int roll;

public:

Student() {

name = "";

roll = 0;

cout << "Student object created using default constructor." << endl;

}

Student(string n, int r) {

name = n;
```

```
roll = r;
cout << "Student object created using parameterized constructor." << endl;
}
void input() {
cout << "Enter name: ";
getline(cin, name);
cout << "Enter roll number: ";
cin >> roll;
}
void display() {
cout << "Name: " << name << ", Roll Number: " << roll << endl;
}
~Student() {
cout << "Destructor called for student: " << name << endl;
}
};
int main() {
cout << "--- Dynamic Allocation for Single Object ---" << endl;
Student *s1 = new Student;
s1->input();
s1->display();
delete s1;
int n;
cout << "\nEnter number of students: ";
cin >> n;
Student* sArray = new Student[n];
for (int i = 0; i < n; i++) {
```

```
cout << "\nEnter details for student " << i + 1 << ":" << endl;
sArray[i].input();
}
cout << "\n--- Student Details ---" << endl;
for (int i = 0; i < n; i++) {
cout << "Student " << i + 1 << ": ";
sArray[i].display();
}
delete[] sArray;
return 0;
}
```

Output:

```
---Called through constructor---
Enter Name: Suraj Nepal
Enter Roll Number: 48
Name: Suraj Nepal, Roll Number: 48

Enter number of students: 1
---Called through constructor---

Enter details for student 1,
Enter Name: Suraj Nepal
Enter Roll Number: 48

--- Student Details ---
Student: 1: Name: Suraj Nepal, Roll Number: 48
```

4. Write a program that demonstrates default, parameterized, and copy constructors. Use the this pointer to calculate the midpoint between two points.

Source Code:

```
#include <iostream>
using namespace std;
class Point {
   float x, y;
public:
   Point() {
      x = 0;
      y = 0;
      cout << "Default constructor called.\n";
   }
   Point(float a, float b) {
      x = a;
      y = b;
      cout << "Parameterized constructor called for (" << x << ", " << y << ")\n";
   }
   Point(const Point &p) {
      x = p.x;
      y = p.y;
      cout << "Copy constructor called for (" << x << ", " << y << ")\n";
   }
   Point midpoint(const Point &p) const {
      float midX = (this->x + p.x) / 2;
      float midY = (this->y + p.y) / 2;
      return Point(midX, midY);
```

```
    }

    void input() {
        cout << "Enter x and y coordinates: ";
        cin >> x >> y;
    }

    void display() const {
        cout << "(" << x << ", " << y << ")\n";
    }
};

int main() {
    cout << "--- Enter Coordinates for Point 1 ---" << endl;
    Point p1;
    p1.input();

    cout << "--- Enter Coordinates for Point 2 ---" << endl;
    Point p2;
    p2.input();

    cout << "--- Copying Point 1 to Point 3 ---" << endl;
    Point p3 = p1;

    cout << "\n--- Midpoint of Point 1 and Point 2 ---" << endl;
    Point mid = p1.midpoint(p2);
```

```
    cout << "\nPoint 1: ";
    p1.display();
    cout << "Point 2: ";
    p2.display();
    cout << "Copied Point (p3): ";
    p3.display();
    cout << "Midpoint: ";
    mid.display();
    return 0;
}
```

Output:

```
--- Enter Coordinates for Point 1 ---
Default constructor called.
Enter x and y coordinates: 4 5
--- Enter Coordinates for Point 2 ---
Default constructor called.
Enter x and y coordinates: 7 8
--- Copying Point 1 to Point 3 ---
Copy constructor called for (4, 5)

--- Midpoint of Point 1 and Point 2 ---
Parameterized constructor called for (5.5, 6.5)

Point 1: (4, 5)
Point 2: (7, 8)
Copied Point (p3): (4, 5)
Midpoint: (5.5, 6.5)
```

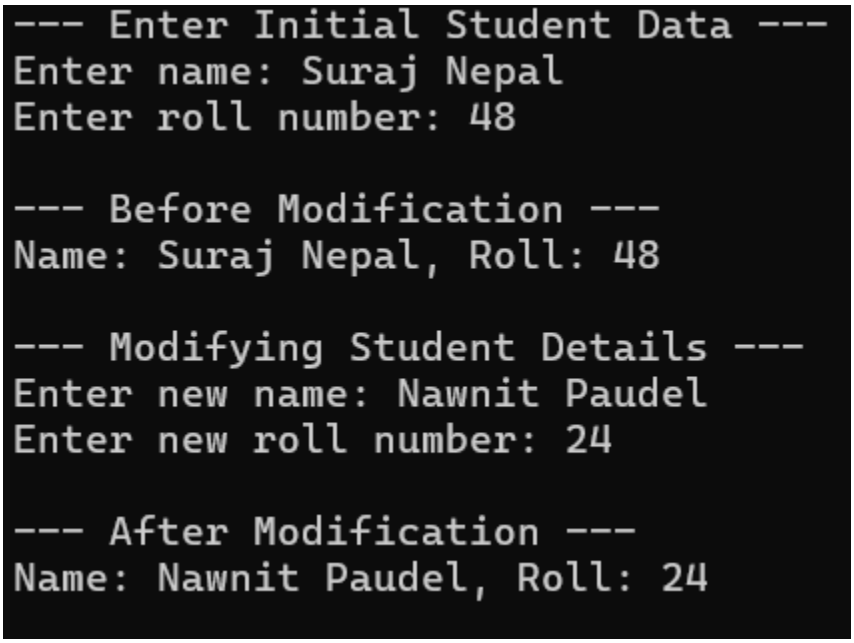
5. Write a program to define a function that takes an object as a reference parameter and modifies its data members.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;
class Student {
private:
    string name;
    int roll;

public:
    Student() {
        name = "";
        roll = 0;
    }

    void input() {
        cout << "Enter name: ";
        getline(cin, name);
        cout << "Enter roll number: ";
        cin >> roll;
        cin.ignore();
    }

    void display() const {
        cout << "Name: " << name << ", Roll: " << roll << endl;
```

```
    }
    friend void modify(Student &s);
};

void modify(Student &s) {
    cout << "\n--- Modifying Student Details ---" << endl;
    cout << "Enter new name: ";
    getline(cin, s.name);
    cout << "Enter new roll number: ";
    cin >> s.roll;
}

int main() {
    Student s;
    cout << "--- Enter Initial Student Data ---" << endl;
    s.input();
    cout << "\n--- Before Modification ---" << endl;
    s.display();
    modify(s);
    cout << "\n--- After Modification ---" << endl;
    s.display();
    return 0;
}
```

Output:



6. Write a program that defines a class inside a namespace and uses a reference to modify object attributes.

Source Code:

```
#include<iostream>
using namespace std;
namespace School{
    class Student{
        int roll;
        string name;
        public:
            Student(){
                name=" ";
                roll=0;
            }
            void getdata(){
                cout<<"Enter the Name of the Student: ";
```

```
        getline(cin,name);
                cout<<"Enter the Roll Number of the Student:";
                cin>>roll;
        }
        void Display(){
                cout<<"----Student Details----"<<endl;
                cout<<"Name:"<<name<<"  "<<"Roll Number:"<<roll<<endl;
        }
        friend void modify(School::Student &s);
};
        void modify(School::Student &s){
                cout<<"Enter the Name of the New Student: ";
                cin.ignore();
                getline(cin,s.name);
                cout<<"Enter the Roll Number of the Student:";
                cin>>s.roll;
        }
}


        int main(){
                School::Student s;
                s.getdata();
        cout<<"---Before Modifications---"<<endl;
        s.Display();
        modify(s);
        cout<<"---After Modifications---"<<endl;
```

```
        s.Display();
        return 0;
    }
```

Output:

```
Enter the Name of the Student: Suraj Nepal
Enter the Roll Number of the Student:48
---Before Modifications---
----Student Details----
Name:Suraj Nepal  Roll Number:48
Enter the Name of the New Student: Nawnit Paudel
Enter the Roll Number of the Student:24
---After Modifications---
----Student Details----
Name:Nawnit Paudel  Roll Number:24
```

7. Write a program that defines a constant member function to access constant object data, and use const_cast to modify it safely.

Source Code:

```
#include <iostream>
#include <string>

using namespace std;

class Student {
private:
    string studentName;
    int studentRoll;

public:
    Student(string name = " ", int roll = 0) {
```

```
        studentName = name;
        studentRoll = roll;
    }

    void display() const {
        cout << "Name: " << studentName << ", Roll: " << studentRoll << endl;
    }

    void modify() const {
        cout << "\n--- Attempting to Modify Constant Object ---" << endl;

        Student* modifiable = const_cast<Student*>(this);

        cout << "Enter new name: ";
        getline(cin, modifiable->studentName);
        cout << "Enter new roll number: ";
        cin >> modifiable->studentRoll;
        cin.ignore();
    }
};

int main() {
    const Student student("Suraj", 48);

    cout << "--- Constant Object (Before Modification) ---" << endl;
    student.display();
    student.modify();
```

```
    cout << "\n--- Constant Object (After Modification) ---" << endl;
    student.display();

    return 0;
}
```

Output:

```
--- Constant Object (Before Modification) ---
Name: Suraj, Roll: 48

--- Attempting to Modify Constant Object ---
Enter new name: Nawnit
Enter new roll number: 24

--- Constant Object (After Modification) ---
Name: Nawnit, Roll: 24
```

8. Write a program to demonstrate a friend function that accesses private data from two different classes and adds their values.

Source Code:

```
#include <iostream>

using namespace std;

class ClassB;

class ClassA {

private:
    int valueA;

public:
    ClassA(int a = 0) {
```

```
        valueA = a;
    }

    friend int addValues(ClassA, ClassB);
};

class ClassB {
private:
    int valueB;

public:
    ClassB(int b = 0) {
        valueB = b;
    }
    friend int addValues(ClassA, ClassB);
};

int addValues(ClassA a, ClassB b) {
    return a.valueA + b.valueB;
}

int main() {
    int x, y;

    cout << "Enter value for ClassA: ";
    cin >> x;
    cout << "Enter value for ClassB: ";
```

```
    cin >> y;

    ClassA objA(x);
    ClassB objB(y);

    int result = addValues(objA, objB);
    cout << "\nSum of values (ClassA + ClassB): " << result << endl;

    return 0;
}
```

Output:

```
Enter value for ClassA: 35
Enter value for ClassB: 48

Sum of values (ClassA + ClassB): 83
```

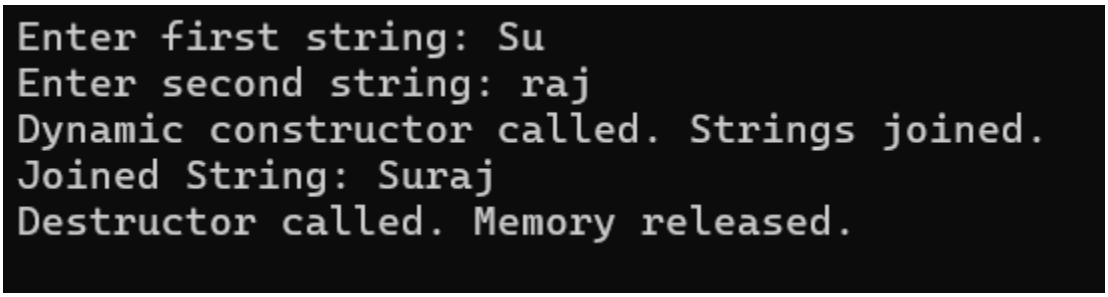
9. Write a program to define a class String that uses a dynamic constructor to allocate memory and join two strings entered by the user.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;
class MyString {
private:
    string str;
public:
    MyString(string* s1, string* s2) {
```

```
        str = *s1 + *s2;
        cout << "Dynamic constructor called. Strings joined." << endl;
    }

    void display() const {
        cout << "Joined String: " << str << endl;
    }

    ~MyString() {
        cout << "Destructor called. Memory released." << endl;
    }
};

int main() {
    string input1, input2;

    cout << "Enter first string: ";
    cin >> input1;

    cout << "Enter second string: ";
    cin >> input2;

    MyString s(&input1, &input2);
    s.display();

    return 0;
}
```

Output:



10.Write a program to create an array of five Employee objects, each with name and salary. Display the employee with the highest salary.

Source Code:

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

class Employee {
private:
    string name;
    float salary;

public:
    void input() {
        cout << "Enter employee name: ";
        getline(cin, name);
        cout << "Enter salary: ";
        cin >> salary;
    }

    void display() const {
```

```
        cout << "Name: " << name << ", Salary: Rs. " << salary << endl;
    }

    float getSalary() const {
        return salary;
    }
};

int main() {
    Employee emp[5];
    int highestIndex = 0;

    for (int i = 0; i < 5; i++) {
        cout << "\n--- Enter Details for Employee " << i + 1 << " ---" << endl;
        emp[i].input();
        cin.ignore();
        if (emp[i].getSalary() > emp[highestIndex].getSalary()) {
            highestIndex = i;
        }
    }

    cout << "\n--- Employee with Highest Salary ---" << endl;
    emp[highestIndex].display();

    return 0;
}
```

Output:

```
--- Enter Details for Employee 1 ---
Enter employee name: Suraj Nepal
Enter salary: 85000

--- Enter Details for Employee 2 ---
Enter employee name: Nawnit Paudel
Enter salary: 80000

--- Enter Details for Employee 3 ---
Enter employee name: Sarthak Bhattarai
Enter salary: 82000

--- Enter Details for Employee 4 ---
Enter employee name: Siddhant Giri
Enter salary: 84000

--- Enter Details for Employee 5 ---
Enter employee name: Ashutosh Thapa
Enter salary: 90000

--- Employee with Highest Salary ---
Name: Ashutosh Thapa, Salary: Rs. 90000
```

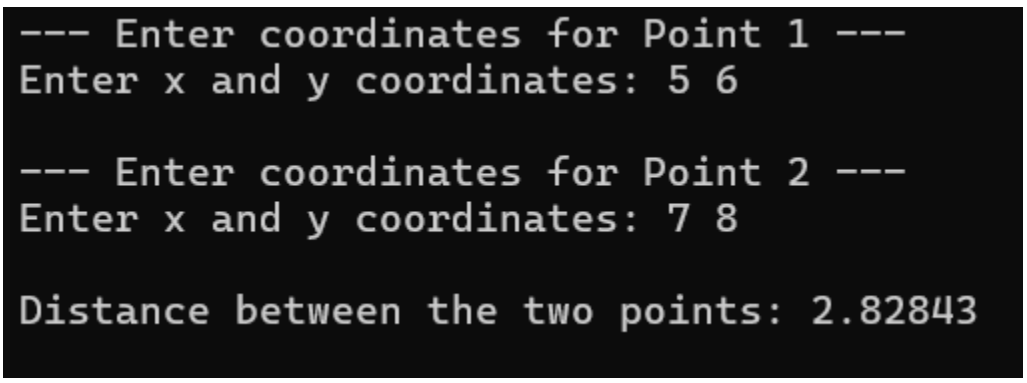
11. Write a program to define a class Point. Use pointers to dynamically allocate memory for two points and calculate the distance between them.

Source Code:

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

class Point {
private:
    float x, y;

public:
    Point(float a = 0.0, float b = 0.0) {
        x = a;
        y = b;
    }

    void input() {
        cout << "Enter x and y coordinates: ";
        cin >> x >> y;
    }
    friend float distance(Point* p1, Point* p2);
};
float distance(Point* p1, Point* p2) {
    return sqrt(pow(p2->x - p1->x, 2) + pow(p2->y - p1->y, 2));
}
```

```
int main() {
    Point* point1 = new Point();
    Point* point2 = new Point();

    cout << "--- Enter coordinates for Point 1 ---" << endl;
    point1->input();

    cout << "\n--- Enter coordinates for Point 2 ---" << endl;
    point2->input();

    float d = distance(point1, point2);
    cout << "\nDistance between the two points: " << d << endl;

    delete point1;
    delete point2;

    return 0;
}
```

Output:



12. Write a program to define a class Box. Use the this pointer in a member function to compare two boxes and return the one with the greater volume.

Source Code:

```
#include <iostream>

using namespace std;


class Box {

private:

   float length, width, height;


public:

   Box(float l = 1, float w = 1, float h = 1) {

      length = l;

      width = w;

      height = h;

   }


   void input() {

      cout << "Enter length, width, and height: ";

      cin >> length >> width >> height;

   }


   float volume() {

      return length * width * height;

   }


   Box compare(Box& b) {
```

```
        if (this->volume() > b.volume()) {
            return *this;
        } else {
            return b;
        }
    }

    void display() {
        cout << "Dimensions (LxWxH): " << length << " x " << width << " x " << height << endl;
        cout << "Volume: " << volume() << endl;
    }
};

int main() {
    Box box1, box2;
    cout << "--- Enter details for Box 1 ---" << endl;
    box1.input();
    cout << "\n--- Enter details for Box 2 ---" << endl;
    box2.input();

    Box bigger = box1.compare(box2);
    cout << "\n--- Box with Greater Volume ---" << endl;
    bigger.display();

    return 0;
}
```

Output:

```
--- Enter details for Box 1 ---
Enter length, width, and height: 2 3 4

--- Enter details for Box 2 ---
Enter length, width, and height: 3 3 2

--- Box with Greater Volume ---
Dimensions (LxWxH): 2 x 3 x 4
Volume: 24
```

DISCUSSION:

In this Lab, we gained knowledge of passing objects between functions, using arrays and pointers for efficient object management, and working with dynamic memory allocation. Understanding friend functions and classes taught us how to control access to private data, while exploring static and constant members highlighted their role in managing class-wide data and preventing changes. These concepts are fundamental in writing scalable and maintainable object-oriented programs.

CONCLUSION:

This lab helped us understand key concepts in object-oriented programming, such as using objects as function arguments and return values. We learned how to modify arrays and pointers to objects, and how dynamic memory allocation can manage objects. We also got to know about friend functions and classes to allow controlled access to private members.