# Lab 4: Friend function, This pointer, DMA, Namespace

## Objectives

* To understand how to define and use classes and objects in C++.
* To explore various types of constructors and destructors.
* To implement object referencing, friend functions, and this pointer.
* To use dynamic memory allocation and also namespace.

## Tools and Libraries Used

• Programming Language: C++

• IDE: Code::Blocks

• Libraries: #include <iostream>, #include <cmath>, #include <string>

## Theory

**Static Members**

Static members are shared among all objects. They are declared inside the class and defined outside using the :: scope resolution operator.

**Syntax:**

class ClassName {

static DataType VariableName;

static ReturnType FunctionName();

};

DataType ClassName::VariableName = InitialValue;

**Example:**

class Counter {

static int count;

public:

Counter() { count++; }

static void display() { cout << count; }

};

int Counter::count = 0;

**Friend Function**

A friend function is not a class member but can access its private and protected members.

**Syntax:**

class ClassName {

friend ReturnType FunctionName(ClassName &Object);

};

**Example:**

class Student {

string name;

int roll;

friend void update(Student &s);

};

void update(Student &s) {

s.name = "NewName";

s.roll = 10;

}

**This Pointer**

this is an implicit pointer to the current object, used inside non-static member functions.

**Syntax:**

ReturnType FunctionName() {

this->VariableName;

}

**Example:**

class Point {

int x, y;

public:

Point midpoint(Point p) {

int mx = (this->x + p.x) / 2;

int my = (this->y + p.y) / 2;

return Point(mx, my);

}

};

**Namespace**

Namespaces organize code and prevent name conflicts.

**Syntax:**

namespace NamespaceName {

class ClassName { };

}

**Example:**

namespace School {

class Student {

string name;

public:

void input() { cin >> name; }

void display() { cout << name; }

};

}

School::Student s;

s.input();

s.display();

**Const Cast**

const\_cast removes the const qualifier to allow modification of constant data.

**Syntax:**

const TypeName\* ptr;

TypeName\* modPtr = const\_cast<TypeName\*>(ptr);

**Example:**

class Student {

string name;

int roll;

public:

void modify() const {

Student\* s = const\_cast<Student\*>(this);

s->name = "Updated";

s->roll = 101;

}

};

const Student s;

s.modify();

**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.**

#include <iostream>

using namespace std;

class Counter {

private:

static int count; // Static data member to track object count

public:

// Constructor

Counter() {

count++;

cout << "Constructor called. Object number: " << count << 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;

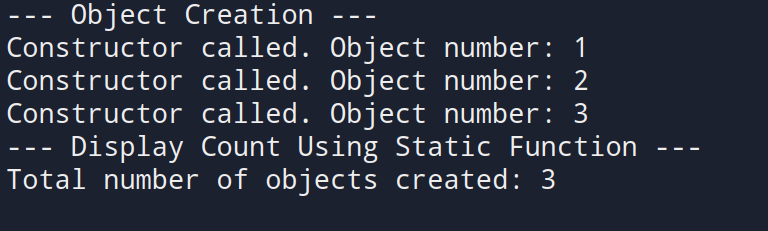
cout << "--- Display Count Using Static Function ---" << endl;

Counter::displayCount(); // Call static function without object

return 0;

}

**OUTPUT**



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

#include <iostream>

using namespace std;

class Student {

private:

int roll;

string name;

public:

// Parameterized constructor

Student(int r, string n) {

roll = r;

name = n;

cout << "Parameterized constructor called." << endl;

}

// Copy constructor

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();

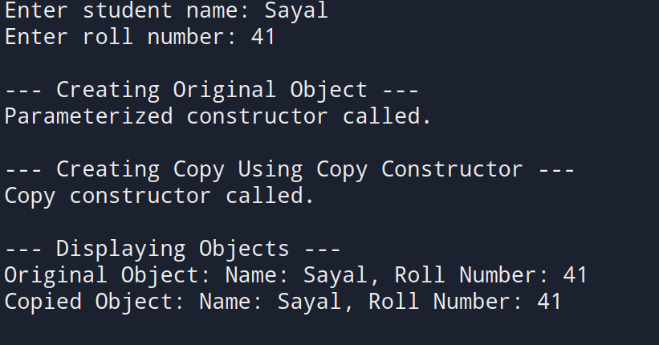
cout << "Copied Object: ";

s2.display();

return 0;

}

**OUTPUT**



**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.**

#include <iostream>

#include <string>

using namespace std;

class Student {

private:

string name;

int roll;

public:

// Default constructor

Student() {

name = "";

roll = 0;

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

}

// Parameterized constructor

Student(string n, int r) {

name = n;

roll = r;

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

}

void input() {

// To handle newline after integer input in main

// Note: Call cin.ignore() once before the first getline in main

cout << "Enter name: ";

getline(cin, name);

cout << "Enter roll number: ";

cin >> roll;

cin.ignore(); // ignore newline after reading roll number

}

void display() {

cout << "Name: " << name << ", Roll Number: " << roll << endl;

}

// Destructor

~Student() {

cout << "Destructor called for student: " << name << endl;

}

};

int main() {

// Dynamic allocation for single object

cout << "--- Dynamic Allocation for Single Object ---" << endl;

Student\* s1 = new Student;

// For first input, no leftover newline, so no ignore here

s1->input();

s1->display();

delete s1; // deallocate memory

// Dynamic allocation for array of objects

int n;

cout << "\nEnter number of students: ";

cin >> n;

cin.ignore(); // ignore leftover newline before getline in input()

Student\* sArray = new Student[n]; // array of objects

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();

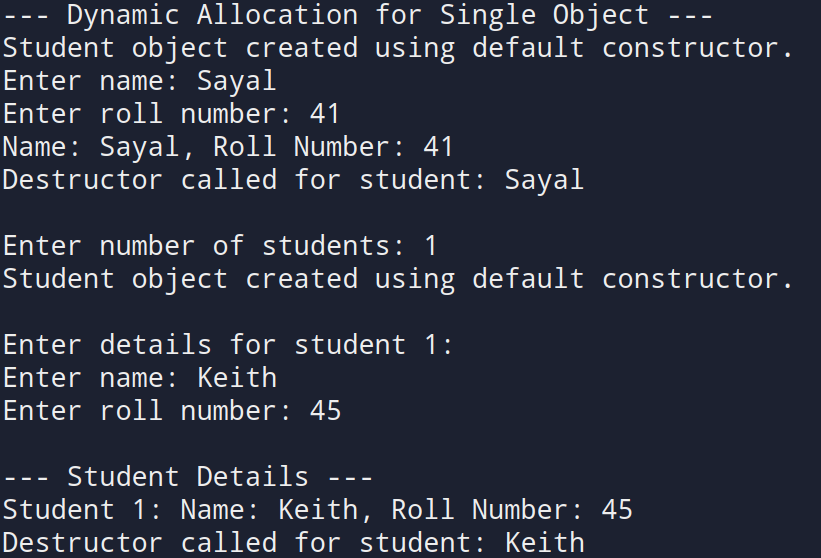
}

delete[] sArray; // deallocate array memory

return 0;

}

**OUTPUT**



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

#include <iostream>

using namespace std;

class Point {

private:

float x, y;

public:

// Default constructor

Point() {

x = 0;

y = 0;

cout << "Default constructor called.\n";

}

// Parameterized constructor

Point(float a, float b) {

x = a;

y = b;

cout << "Parameterized constructor called for (" << x << ", " << y << ")\n";

}

// Copy constructor

Point(const Point &p) {

x = p.x;

y = p.y;

cout << "Copy constructor called for (" << x << ", " << y << ")\n";

}

// Function to calculate midpoint using this pointer

Point midpoint(const Point &p) const {

float midX = (this->x + p.x) / 2;

float midY = (this->y + p.y) / 2;

return Point(midX, midY); // returning new object

}

void input() {

cout << "Enter x and y coordinates: ";

cin >> x >> y;

}

void display() const {

cout << "(" << x << ", " << y << ")" << endl;

}

};

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; // invokes copy constructor

cout << "\n--- Midpoint of Point 1 and Point 2 ---" << endl;

Point mid = p1.midpoint(p2); // uses this pointer

cout << "\nPoint 1: "; p1.display();

cout << "Point 2: "; p2.display();

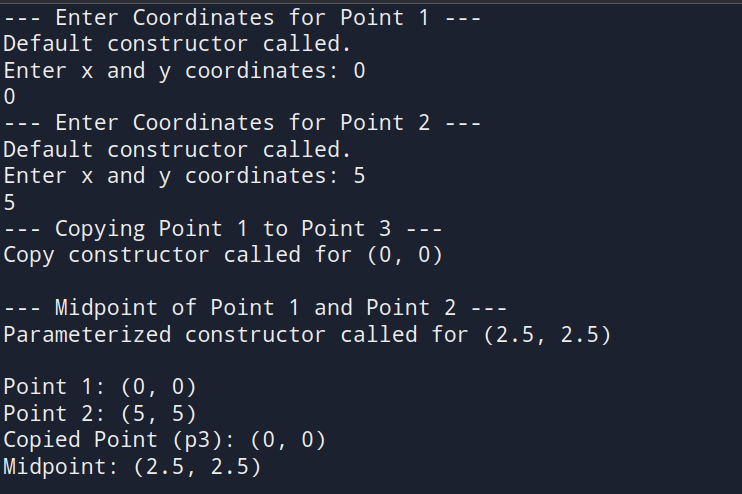
cout << "Copied Point (p3): "; p3.display();

cout << "Midpoint: "; mid.display();

return 0;

}

**OUTPUT**



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

#include <iostream>

using namespace std;

class Student {

private:

string name;

int roll;

public:

// Default constructor

Student() {

name = "";

roll = 0;

}

// Function to input initial data

void input() {

cout << "Enter name: ";

getline(cin, name);

cout << "Enter roll number: ";

cin >> roll;

cin.ignore(); // Clear input buffer

}

// Function to display data

void display() const {

cout << "Name: " << name << ", Roll: " << roll << endl;

}

// Grant friend function access to modify data

friend void modify(Student &s);

};

// Function to modify object passed by reference

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);

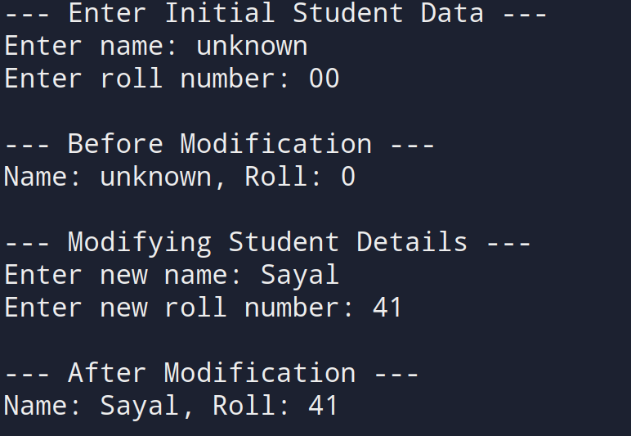
cout << "\n--- After Modification ---" << endl;

s.display();

return 0;

}

**OUTPUT**



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

#include <iostream>

#include <string>

using namespace std;

namespace School {

class Student {

private:

string name;

int roll;

public:

Student() {

name = "";

roll = 0;

}

void input() {

cout << "Enter student name: ";

getline(cin, name);

cout << "Enter roll number: ";

cin >> roll;

cin.ignore();

}

void display() const {

cout << "Name: " << name << ", Roll: " << roll << endl;}

friend void updateStudent(Student &s);

};

void updateStudent(Student &s) {

cout << "\n--- Modifying Student Data ---\n";

cout << "Enter new name: ";

getline(cin, s.name);

cout << "Enter new roll number: ";

cin >> s.roll;

cin.ignore();

}

}

int main() {

School::Student s;

cout << "--- Enter Student Data ---" << endl;

s.input();

cout << "\n--- Before Modification ---" << endl;

s.display();

School::updateStudent(s);

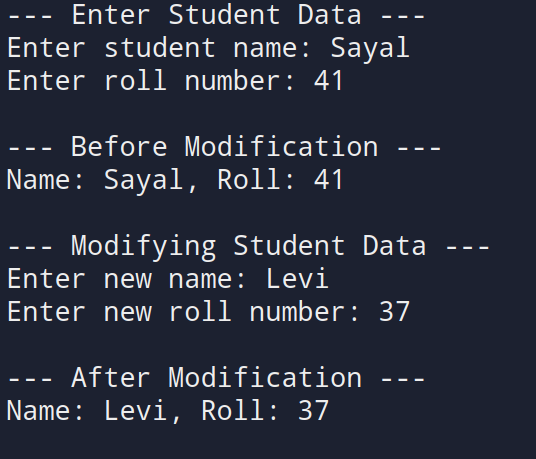
cout << "\n--- After Modification ---" << endl;

s.display();

return 0;

}

**OUTPUT**



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

#include <iostream>

using namespace std;

class Student {

private:

string name;

int roll;

public:

Student(string n = "N/A", int r = 0) {

name = n;

roll = r;

}

void display() const {

cout << "Name: " << name << ", Roll: " << roll << endl;

}

// Function to modify data using const\_cast

void modify() const {

cout << "\n--- Attempting to Modify Constant Object ---" << endl;

// Cast away const-ness to modify data

Student\* modifiable = const\_cast<Student\*>(this);

cout << "Enter new name: ";

getline(cin, modifiable->name);

cout << "Enter new roll number: ";

cin >> modifiable->roll;

cin.ignore(); // clear buffer

}

};

int main() {

// Creating a constant object

const Student s("Original", 101);

cout << "--- Constant Object (Before Modification) ---" << endl;

s.display();

// Modifying constant object using const\_cast

s.modify();

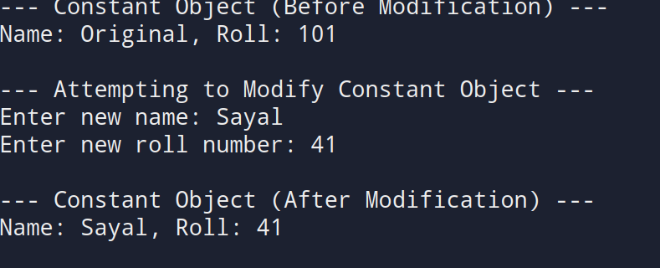
cout << "\n--- Constant Object (After Modification) ---" << endl;

s.display();

return 0;

}

**OUTPUT**



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

#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);

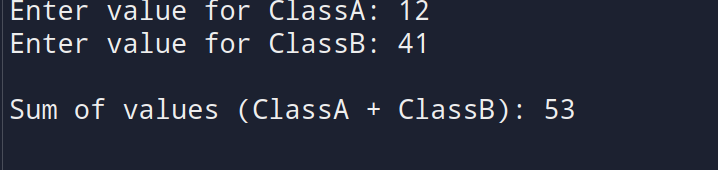
int result = addValues(objA, objB);

cout << "\nSum of values (ClassA + ClassB): " << result << endl;

return 0;

}

**OUTPUT**



**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.**

#include <iostream>

#include <cstring> // For strlen and strcpy

using namespace std;

class MyString {

private:

char\* str;

public:

MyString(const char\* s1, const char\* s2) {

int len = strlen(s1) + strlen(s2);

str = new char[len + 1]; // +1 for null terminator

strcpy(str, s1);

strcat(str, s2);

cout << "Dynamic constructor called. Strings joined." << endl;

}

void display() const {

cout << "Joined String: " << str << endl;

}

~MyString() {

delete[] str;

cout << "Destructor called. Memory released." << endl;

}

};

int main() {

char input1[100], input2[100];

cout << "Enter first string: ";

cin.getline(input1, 100);

cout << "Enter second string: ";

cin.getline(input2, 100);

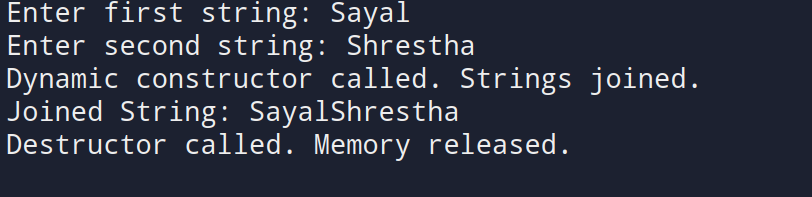
MyString s(input1, input2);

s.display();

return 0;

**}**

**OUTPUT**



**10. Write a program to define a class Rectangle with length and width. Implement a member function that takes another Rectangle object and returns a new object with combined dimensions.**

#include <iostream>

using namespace std;

class Rectangle {

private:

float length, width;

public:

Rectangle() {

length = 0;

width = 0;

}

Rectangle(float l, float w) {

length = l;

width = w;

}

void input() {

cout << "Enter length: ";

cin >> length;

cout << "Enter width: ";

cin >> width;

}

void display() const {

cout << "Length: " << length << ", Width: " << width << endl;

}

Rectangle combine(const Rectangle &r) const {

float newLength = length + r.length;

float newWidth = width + r.width;

return Rectangle(newLength, newWidth);

}

};

int main() {

Rectangle r1, r2, r3;

cout << "--- Enter Dimensions for Rectangle 1 ---" << endl;

r1.input();

cout << "\n--- Enter Dimensions for Rectangle 2 ---" << endl;

r2.input();

// Combine dimensions using object as argument and return

r3 = r1.combine(r2);

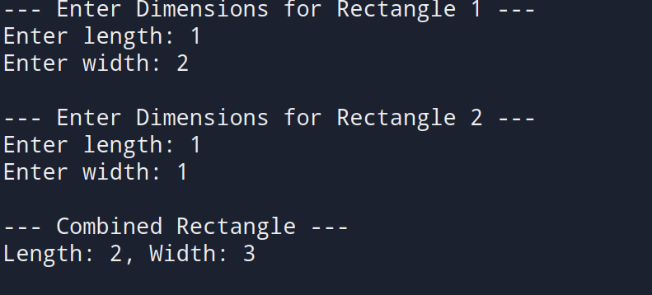
cout << "\n--- Combined Rectangle ---" << endl;

r3.display();

return 0;

}

**OUTPUT**



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

#include <iostream>

using namespace std;

class Employee {

private:

string name;

float salary;

public:

void input() {

cout << "Enter employee name: ";

cin.ignore(); // clear newline character

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();

if (emp[i].getSalary() > emp[highestIndex].getSalary()) {

highestIndex = i;

}}

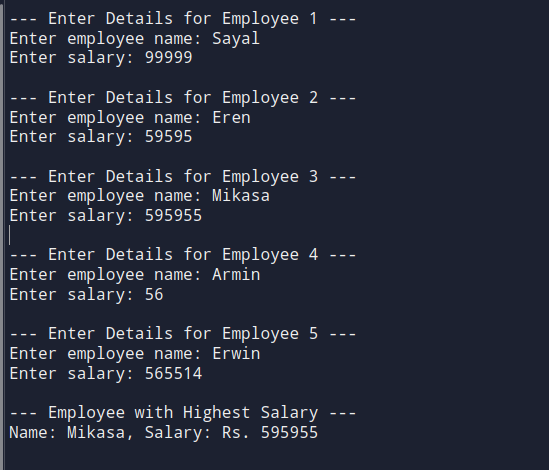
cout << "\n--- Employee with Highest Salary ---" << endl;

emp[highestIndex].display();

return 0;

}

**OUTPUT**



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

#include <iostream>

#include <cmath> // For sqrt and pow

using namespace std;

class Point {

private:

float x, y;

public:

// Constructor

Point(float a = 0, float b = 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;

// Deallocate memory

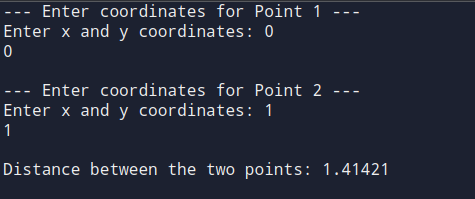
delete point1;

delete point2;

return 0;

}

**OUTPUT**



**13. 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.**

#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() const {

return length \* width \* height;

}

Box compare(const Box& b) const {

if (this->volume() > b.volume()) {

return \*this; // current object has greater volume

} else {

return b; // parameter object has greater volume

}

}

void display() const {

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); // compare using 'this' pointer

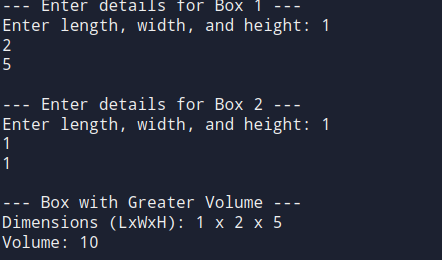
cout << "\n--- Box with Greater Volume ---" << endl;

bigger.display();

return 0;

}

**OUTPUT**



**DISCUSSION**

This lab demonstrated key object-oriented programming concepts in C++ through 13 programs. We practiced creating classes, using different constructors, managing memory with new/delete, and applying friend functions, namespaces, and the this pointer. Real-world examples helped reinforce abstract concepts like object copying, reference modification, and memory-safe design.

**CONCLUSION**

Through this lab, we strengthened our understanding of classes, constructors, destructors, and object management in C++. We also learned practical techniques for memory handling, code modularity, and safe data access—forming a solid base for further C++ programming.