**Exercise 1:**

**Question: 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.**

**Code:**

#include <iostream>

using namespace std;

class Counter {

private:

static int count;

public:

Counter() {

count++;

}

~Counter() {

count--;

}

static void displayCount() {

cout << "Number of objects: " << count << endl;

}

};

int Counter::count = 0;

int main() {

Counter c1;

Counter::displayCount();

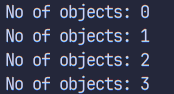
Counter c2, c3;

Counter::displayCount();

return 0;

}

**Output:**



**Exercise 2:**

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

**Code:**

#include <iostream>

using namespace std;

class Number {

int num;

public:

Number(int n = 0) {

num = n;

}

Number(const Number &obj) {

cout << "Copy constructor called" << endl;

num = obj.num;

}

void display() {

cout << "Number = " << num << endl;

}

};

int main() {

Number n1(10);

Number n2(n1);

cout << "Original Number: ";

n1.display();

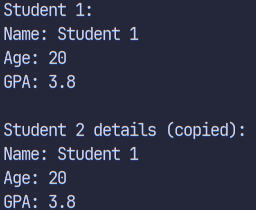
cout << "Copied Number: ";

n2.display();

return 0;

}

**Output:**



**Exercise 3:**

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

**Code:**

#include <iostream>

using namespace std;

class Student {

string name;

int age;

public:

Student() {

name = "Unknown";

age = 0;

}

void setData(string n, int a) {

name = n;

age = a;

}

void display() {

cout << "Name: " << name << ", Age: " << age << endl;

}

};

int main() {

// Single object

Student\* ptr = new Student();

ptr->setData("John", 20);

ptr->display();

delete ptr;

// Array of objects

Student\* arr = new Student[3];

arr[0].setData("Alice", 21);

arr[1].setData("Bob", 22);

arr[2].setData("Charlie", 20);

for(int i = 0; i < 3; i++) {

arr[i].display();

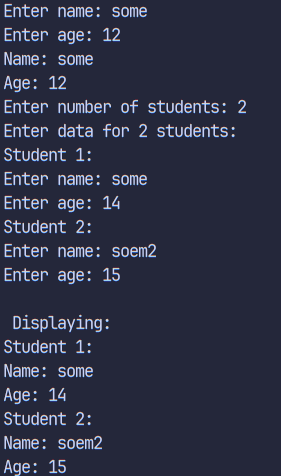
}

delete[] arr;

return 0;

}

**Output:**



**Exercise 4:**

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

**Code:**

#include <iostream>

using namespace std;

class Point {

double x, y;

public:

// Default constructor

Point() : x(0), y(0) {}

// Parameterized constructor

Point(double x, double y) {

this->x = x;

this->y = y;

}

// Copy constructor

Point(const Point &p) {

x = p.x;

y = p.y;

}

Point midpoint(const Point &p) {

return Point((this->x + p.x)/2, (this->y + p.y)/2);

}

void display() {

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

}

};

int main() {

Point p1(2, 3);

Point p2(4, 5);

Point mid = p1.midpoint(p2);

cout << "Point 1: ";

p1.display();

cout << "Point 2: ";

p2.display();

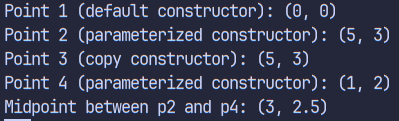
cout << "Midpoint: ";

mid.display();

return 0;

}

**Output:**



**Exercise 5:**

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

**Code:**

#include <iostream>

using namespace std;

class Rectangle {

double length;

double width;

public:

Rectangle(double l = 0, double w = 0) {

length = l;

width = w;

}

void display() {

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

}

friend void modifyDimensions(Rectangle &rect);

};

void modifyDimensions(Rectangle &rect) {

rect.length \*= 2;

rect.width \*= 2;

}

int main() {

Rectangle rect(5, 3);

cout << "Original dimensions:" << endl;

rect.display();

modifyDimensions(rect);

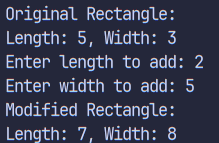
cout << "Modified dimensions:" << endl;

rect.display();

return 0;

}

**Output:**



**Exercise 6:**

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

**Code:**

#include <iostream>

using namespace std;

namespace Mathematics {

class Calculator {

int value;

public:

Calculator(int v = 0) : value(v) {}

void setValue(int &v) {

value = v;

}

int getValue() {

return value;

}

void add(int &num) {

value += num;

}

void display() {

cout << "Value: " << value << endl;

}

};

}

int main() {

Mathematics::Calculator calc(10);

int num = 5;

cout << "Initial ";

calc.display();

calc.add(num);

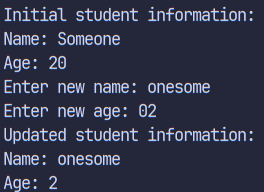
cout << "After adding " << num << ": ";

calc.display();

return 0;

}

**Output:**



**Exercise 7:**

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

**Code:**

#include <iostream>

using namespace std;

class Data {

int value;

public:

Data(int v = 0) : value(v) {}

void display() const {

cout << "Value: " << value << endl;

}

void setValue(int v) {

value = v;

}

static void modifyConstObject(const Data& d) {

Data& non\_const\_data = const\_cast<Data&>(d);

non\_const\_data.setValue(100);

}

};

int main() {

const Data d(50);

cout << "Original ";

d.display();

Data::modifyConstObject(d);

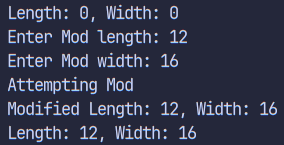
cout << "Modified ";

d.display();

return 0;

}

**Output:**



**Exercise 8:**

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

**Code:**

#include <iostream>

using namespace std;

class ClassB;

class ClassA {

int valueA;

public:

ClassA(int v = 0) : valueA(v) {}

friend int addValues(ClassA, ClassB);

};

class ClassB {

int valueB;

public:

ClassB(int v = 0) : valueB(v) {}

friend int addValues(ClassA, ClassB);

};

int addValues(ClassA objA, ClassB objB) {

return objA.valueA + objB.valueB;

}

int main() {

ClassA a(5);

ClassB b(10);

cout << "Sum of values: " << addValues(a, b) << endl;

return 0;

}

**Output:**



**Exercise 9:**

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

**Code:**

#include <iostream>

#include <cstring>

using namespace std;

class String {

char\* str;

public:

String() {

str = new char[1];

str[0] = '\0';

}

String(const char\* s) {

str = new char[strlen(s) + 1];

strcpy(str, s);

}

String join(const String& s) {

char\* temp = new char[strlen(str) + strlen(s.str) + 1];

strcpy(temp, str);

strcat(temp, s.str);

return String(temp);

}

void display() {

cout << str << endl;

}

~String() {

delete[] str;

}

};

int main() {

String s1("Hello ");

String s2("World!");

String s3 = s1.join(s2);

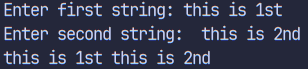
cout << "Joined string: ";

s3.display();

return 0;

}

**Output:**



**Exercise 10:**

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

**Code:**

include <iostream>

#include <string>

using namespace std;

class Employee {

private:

string name;

double salary;

public:

Employee(string n = "", double s = 0.0) {

this->name = n;

this->salary = s;

}

void display() const {

cout << "Name: " << name << ", Salary: " << salary << endl;

}

double getsalary() const {

return salary;

}

string getname() const {

return name;

}

void setEmployee(string n, double s) {

name = n;

salary = s;

}

};

int main() {

Employee e[5];

string name;

double salary;

for(int i = 0; i < 5; i++) {

cout << "Enter name for employee " << i+1 << ": ";

cin >> name;

cout << "Enter salary for employee " << i+1 << ": ";

cin >> salary;

e[i].setEmployee(name, salary);

}

cout << "\nEmployee Details:" << endl;

for(int i = 0; i < 5; i++) {

e[i].display();

}

cout << "\nHighest Salary Employee:" << endl;

int maxIndex = 0;

for(int i = 1; i < 5; i++) {

if(e[i].getsalary() > e[maxIndex].getsalary()) {

maxIndex = i;

}

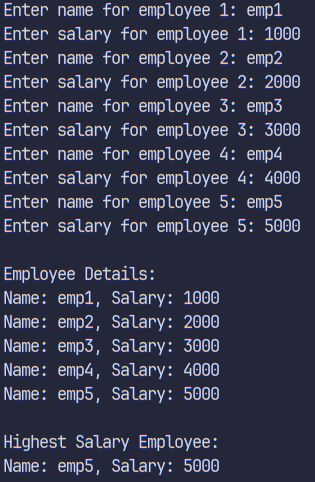
}

e[maxIndex].display();

return 0;

}

**Output:**



**Exercise 11:**

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

**Code:**

#include <iostream>

#include <cmath>

using namespace std;

class Point {

private:

int x, y;

public:

Point(int x = 0, int y = 0) {

this->x = x;

this->y = y;

}

friend double distance(const Point& p1, const Point& p2) {

return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));

}

void display() const {

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

}

};

int main() {

Point p1(1, 2);

Point p2(3, 4);

cout << "Distance between points: " << distance(p1, p2) << endl;

return 0;

}

**Output:**



**Exercise 12:**

**Question: 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.**

**Code:**

#include <iostream>

using namespace std;

class Box {

double length, width, height;

public:

Box(double l = 0, double w = 0, double h = 0) {

length = l;

width = w;

height = h;

}

double volume() {

return length \* width \* height;

}

Box\* compareVolume(Box\* other) {

return (this->volume() > other->volume()) ? this : other;

}

void display() {

cout << "Volume: " << volume() << endl;

}

};

int main() {

Box box1(2, 3, 4);

Box box2(3, 4, 5);

Box\* larger = box1.compareVolume(&box2);

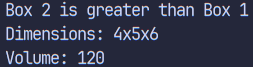
cout << "Larger box ";

larger->display();

return 0;

}

**Output:**



**Discussion:**

The lab exercises demonstrate essential Object-Oriented Programming concepts in C++ through twelve focused programs. The exercises progress from fundamental concepts like static members and constructors to intermediate topics such as reference parameters and namespaces, culminating in practical applications involving dynamic memory management and object relationships. Each program builds upon previous concepts while introducing new features, creating a comprehensive learning progression in C++ OOP principles.

**Conclusion:**

These programs effectively showcase C++'s OOP capabilities, from basic class implementation to advanced memory management. The exercises emphasize crucial programming concepts like resource management, encapsulation, and code organization. Through practical implementations, the programs demonstrate how C++ combines powerful OOP features with efficient resource control, making it suitable for building robust software systems.