**1.Write a C++ program that demonstrates the use of pointers to change the local values of variables defined in one function from within another function. Create two functions, main() and modifyValues(). The main() function should declare local variables, and the modifyValues() function should take pointers as parameters to modify the values of these local variables**

#include <iostream>

using namespace std;

// Function prototype

void modifyValues(int\* x, int\* y);

int main() {

int a = 5;

int b = 10;

cout << "Before calling modifyValues function:" << endl;

cout << "a: " << a << ", b: " << b << endl;

modifyValues(&a, &b);

cout << "After calling modifyValues function:" << endl;

cout << "a: " << a << ", b: " << b << endl;

return 0;

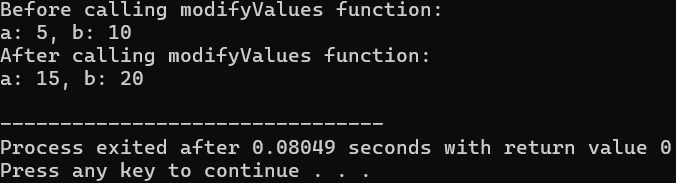
}

void modifyValues(int\* x, int\* y) {

\*x = 15;

\*y = 20;

}

****

**2.Create a C++ program that utilizes multiple inheritance to display employee information. Design a class hierarchy involving three classes: Person, Employee, and DisplayInfo. The Person class should contain basic information such as name and address, the Employee class should include details specific to employment, such as employee ID and salary, and the DisplayInfo class should handle the functionality to display the information**.

#include <iostream>

#include <string>

using namespace std;

class Person {

protected:

string name;

string address;

public:

Person(const string& n, const string& addr) : name(n), address(addr) {}

void displayPersonInfo() const {

cout << "Name: " << name << endl;

cout << "Address: " << address << endl;

}

};

class Employee {

protected:

int employeeID;

double salary;

public:

Employee(int id, double sal) : employeeID(id), salary(sal) {}

void displayEmployeeInfo() const {

cout << "Employee ID: " << employeeID << endl;

cout << "Salary: $" << salary << endl;

}

};

class DisplayInfo : public Person, public Employee {

public:

DisplayInfo(const string& n, const string& addr, int id, double sal)

: Person(n, addr), Employee(id, sal) {}

void displayAllInfo() const {

cout << "Employee Information:" << endl;

displayPersonInfo();

displayEmployeeInfo();

}

};

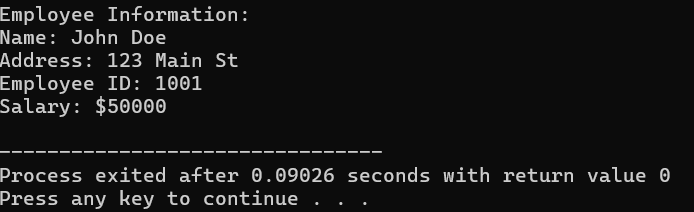
int main() {

DisplayInfo emp("John Doe", "123 Main St", 1001, 50000.0);

emp.displayAllInfo();

return 0;

}

****

**3.Design a C++ program to showcase the internal usage of pointer arithmetic by the compiler when accessing array elements. Create a simple array of integers and utilize pointers to iterate through the elements. The objective is to illustrate how the compiler translates array indexing into pointer arithmetic.**

#include <iostream>

using namespace std;

int main() {

int arr[] = {10, 20, 30, 40, 50};

cout << "Array elements using array indexing:" << endl;

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

cout << "arr[" << i << "] = " << arr[i] << endl;

}

cout << "\nArray elements using pointer arithmetic:" << endl;

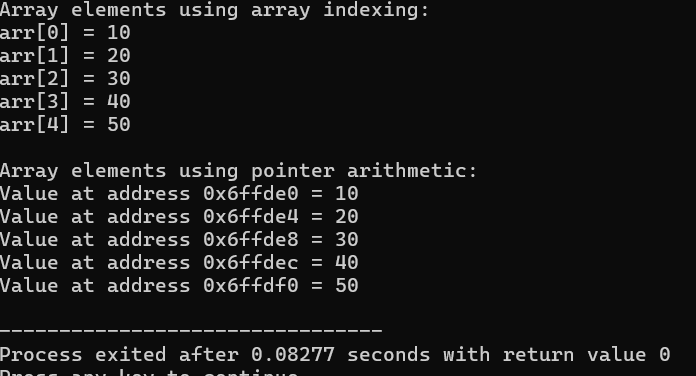
for (int\* ptr = arr; ptr < arr + 5; ++ptr) {

cout << "Value at address " << ptr << " = " << \*ptr << endl;

}

return 0;

}

****

**4. Develop a C++ program that illustrates the concepts of public, private, and protected members within a class hierarchy, showcasing their accessibility in both base and derived classes. Create a base class named BaseClass with a mix of public, private, and protected members. Then, derive a class named DerivedClass from the BaseClass to demonstrate how these members can be accessed through inheritance**

#include <iostream>

using namespace std;

class BaseClass {

public:

int publicVar;

void publicMethod() {

cout << "Public method of BaseClass" << endl;

}

private:

int privateVar;

void privateMethod() {

cout << "Private method of BaseClass" << endl;

}

protected:

int protectedVar;

void protectedMethod() {

cout << "Protected method of BaseClass" << endl;

}

};

class DerivedClass : public BaseClass {

public:

void accessBaseMembers() {

publicVar = 10;

publicMethod();

}

};

int main() {

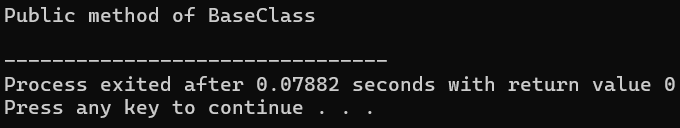
DerivedClass derivedObj;

derivedObj.publicVar = 100;

derivedObj.publicMethod();

return 0;

}

****