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;

void modifyValues(int\* ptrA, int\* ptrB);

int main() {

int a = 5, b = 10;

cout << "Before modification - Value of a: " << a << ", Value of b: " << b << endl;

modifyValues(&a, &b);

cout << "After modification - Value of a: " << a << ", Value of b: " << b << endl;

return 0;

}

void modifyValues(int\* ptrA, int\* ptrB) {

\*ptrA = 15;

\*ptrB = 20;

}

OUTPUT:

Before modification - Value of a: 5, Value of b: 10

After modification - Value of a: 15, Value of b: 20

1. **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() {

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

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

}

};

class Employee : public Person {

private:

int empId;

double salary;

public:

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

: Person(n, addr), empId(id), salary(sal) {}

void displayEmployeeInfo() {

displayPersonInfo();

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

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

}

};

class DisplayInfo : public Employee {

public:

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

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

void displayAllInfo() {

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

displayEmployeeInfo();

}

};

int main() {

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

emp.displayAllInfo();

return 0;

}

OUTPUT:  
Employee Information:

Name: John Doe

Address: 123 Main St

Employee ID: 1001

Salary: $50000

**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() {

const int size = 5;

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

cout << "Using array indexing:" << endl;

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

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

}

cout << "\nUsing pointers and pointer arithmetic:" << endl;

int\* ptr = myArray;

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

cout << "\*(ptr + " << i << ") = " << \*(ptr + i) << endl;

}

return 0;

}

OUTPUT:

Using array indexing:

myArray[0] = 10

myArray[1] = 20

myArray[2] = 30

myArray[3] = 40

myArray[4] = 50

Using pointers and pointer arithmetic:

\*(ptr + 0) = 10

\*(ptr + 1) = 20

\*(ptr + 2) = 30

\*(ptr + 3) = 40

\*(ptr + 4) = 50

**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 << "BaseClass public method called." << endl;

}

protected:

int protectedVar;

void protectedMethod() {

cout << "BaseClass protected method called." << endl;

}

private:

int privateVar;

void privateMethod() {

cout << "BaseClass private method called." << endl;

}

};

public:

void accessBaseMembers() {

cout << "DerivedClass accessing base members:" << endl;

cout << "Public variable: " << publicVar << endl;

publicMethod();

cout << "Protected variable: " << protectedVar << endl;

protectedMethod();

}

};

int main() {

DerivedClass derivedObj;

derivedObj.accessBaseMembers();

return 0;

}

OUTPUT:  
DerivedClass accessing base members:

Public variable: 0

BaseClass public method called.

Protected variable: 0

BaseClass protected method called.