Subject:- C++
LAB Assignment - 9

1) Write a program of pointer to objects.

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

// Class definition
class Box {
public:
    // Member variable
    double length;

    // Constructor to initialize the length
    Box(double l) {
        length = l;
    }

    // Member function to calculate volume
    double volume() {
        return length * length * length; // Volume of a cube
    }
};

int main() {
    // Create an object of Box on the heap using a pointer
    Box* boxPtr = new Box(5.0); // Dynamically allocated Box object

    // Accessing members using pointer
    cout << "Length of the box: " << boxPtr->length << endl;
```

```
    cout << "Volume of the box: " << boxPtr->volume() << endl;

    // Deallocate memory
    delete boxPtr;

    return 0;
}
```

Output:

```
Length of the box: 5
Volume of the box: 125
```

2) Write a program of array of pointer to objects.

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

// Class definition
class Student {
public:
    string name;
    int age;

    // Constructor to initialize Student
    Student(string n, int a) {
        name = n;
        age = a;
    }
```

```
    // Member function to display student details
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    // Create an array of pointers to Student objects
    const int numStudents = 3;
    Student* students[numStudents];

    // Dynamically allocate memory for each Student object
    students[0] = new Student("Alice", 20);
    students[1] = new Student("Bob", 22);
    students[2] = new Student("Charlie", 21);

    // Display details of each student
    cout << "Student Details:" << endl;
    for (int i = 0; i < numStudents; i++) {
        students[i]->display();
    }

    // Deallocate memory
    for (int i = 0; i < numStudents; i++) {
        delete students[i]; // Free the allocated memory for each Student
object
    }

    return 0;
}
```

Output:

```
Student Details:
Name: Alice, Age: 20
Name: Bob, Age: 22
Name: Charlie, Age: 21
```

3) Write a program for this pointer.

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

int main() {
    int num = 42;          // A normal integer variable
    int* ptr = &num;       // Pointer to the integer variable

    // Display the value of num and its address
    cout << "Value of num: " << num << endl;           // Output: 42
    cout << "Address of num: " << &num << endl;        // Address of num
    cout << "Value of ptr (Address of num): " << ptr << endl; // Address of
num
    cout << "Value pointed by ptr: " << *ptr << endl;  // Dereference ptr to get
value of num

    // Change the value of num using the pointer
    *ptr = 100;  // Change the value of num through the pointer
    cout << "New value of num after modifying through ptr: " << num <<
endl; // Output: 100

    // Pointer arithmetic
    int arr[] = {10, 20, 30, 40, 50};  // Array of integers
    int* arrPtr = arr;                 // Pointer to the first element of the array
```

```
    // Display array elements using pointer arithmetic
    cout << "Array elements using pointer arithmetic: ";
    for (int i = 0; i < 5; i++) {
        cout << *(arrPtr + i) << " ";  // Accessing array elements using pointer
arithmetic
    }
    cout << endl;

    return 0;
}
```

Output:

```
Value of num: 42
Address of num: 0x7ffee3d4d93c
Value of ptr (Address of num): 0x7ffee3d4d93c
Value pointed by ptr: 42
New value of num after modifying through ptr: 100
Array elements using pointer arithmetic: 10 20 30 40 50
```

4) Write a program for pointer to derived classes.

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

// Base class
class Shape {
public:
    // Virtual function to be overridden in derived classes
    virtual void draw() {
        cout << "Drawing a shape." << endl;
```

```
    }
};

// Derived class: Circle
class Circle : public Shape {
public:
    void draw() override { // Override the draw function
        cout << "Drawing a circle." << endl;
    }
};

// Derived class: Square
class Square : public Shape {
public:
    void draw() override { // Override the draw function
        cout << "Drawing a square." << endl;
    }
};

int main() {
    // Create a pointer of type Shape
    Shape* shapePtr;

    // Pointing to a Circle object
    shapePtr = new Circle();
    shapePtr->draw(); // Calls Circle's draw method

    // Pointing to a Square object
    shapePtr = new Square();
    shapePtr->draw(); // Calls Square's draw method

    // Clean up memory
    delete shapePtr;

    return 0;
```

```
}
```

Output:

```
Drawing a circle.
Drawing a square.
```

5) Write a program for virtual function.

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

// Base class
class Animal {
public:
    // Virtual function
    virtual void sound() {
        cout << "Animal makes a sound." << endl;
    }
};

// Derived class: Dog
class Dog : public Animal {
public:
    void sound() override { // Override the base class function
        cout << "Dog barks." << endl;
    }
};

// Derived class: Cat
class Cat : public Animal {
```

```
public:
    void sound() override { // Override the base class function
        cout << "Cat meows." << endl;
    }
};

int main() {
    // Create pointers of type Animal
    Animal* animalPtr;

    // Point to a Dog object
    animalPtr = new Dog();
    animalPtr->sound(); // Calls Dog's sound method

    // Point to a Cat object
    animalPtr = new Cat();
    animalPtr->sound(); // Calls Cat's sound method

    // Clean up memory
    delete animalPtr;

    return 0;
}
```

Output:

```
Dog barks.
Cat meows.
```