Modern Programming Principles & Practice

Feb-May 2025

sruthi.padathara@dorset.ie

1. Write a program to swap two numbers using a temporary variable and without using one.
2. Create a 2D array to represent a matrix and perform addition.
3. Find maximum element in an array.
4. Demonstrate use of new operator for dynamic memory allocation.
5. Use this pointer to access data members.
6. Create base class Animal, derived classes Dog, Cat, show single inheritance.
7. Multilevel inheritance: Vehicle → Car → ElectricCar.
8. Function overriding: Call derived method using base pointer (with & without virtual).
9. Create abstract class Shape, implement draw() in Circle, Square.
10. Create a generic function swapValues(T a, T b).
11. Write and read a string from a file using ofstream and ifstream.

1. Swap Two Numbers

With a Temporary Variable:

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

int main() {
    int a = 5, b = 10, temp;
    temp = a;
    a = b;
    b = temp;

    cout << "After swapping: a = " << a << ", b = " << b << endl;
    return 0;
}
```

PseudoCode

START
SET a = 5, b = 10
SET temp = a
SET a = b
SET b = temp
PRINT a, b
END

Without a Temporary Variable:

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

int main() {
    int a = 5, b = 10;
    a = a + b;
    b = a - b;
    a = a - b;

    cout << "After swapping: a = " << a << ", b = " << b << endl;
    return 0;
}
```

START
SET a = 5, b = 10
a = a + b
b = a - b
a = a - b
PRINT a, b
END

2. 2D Matrix Addition

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

int main() {
    int a[2][2] = {{1, 2}, {3, 4}};
    int b[2][2] = {{5, 6}, {7, 8}};
    int sum[2][2];

    for (int i = 0; i < 2; ++i)
        for (int j = 0; j < 2; ++j)
            sum[i][j] = a[i][j] + b[i][j];

    cout << "Sum of matrices:\n";
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 2; ++j)
            cout << sum[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

START
DECLARE 2D arrays a[2][2], b[2][2], sum[2][2]
INITIALIZE a and b with values
FOR i = 0 to 1
  FOR j = 0 to 1
    sum[i][j] = a[i][j] + b[i][j]
PRINT sum matrix
END

3. Find Maximum in Array

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

int main() {
    int arr[] = {3, 7, 2, 9, 5};
    int max = arr[0];

    for (int i = 1; i < 5; ++i)
        if (arr[i] > max)
            max = arr[i];

    cout << "Maximum element: " << max << endl;
    return 0;
}
```

```
START
DECLARE array arr[5] = {values}
SET max = arr[0]
FOR i = 1 to 4
  IF arr[i] > max
    SET max = arr[i]
PRINT max
END
```

4. new Operator for Dynamic Memory

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

int main() {
    int* ptr = new int; // allocate memory
    *ptr = 42;
    cout << "Value: " << *ptr << endl;
    delete ptr; // free memory
    return 0;
}
```

START
ALLOCATE memory for integer pointer ptr
SET *ptr = 42
PRINT *ptr
DEALLOCATE memory
END

5. this Pointer

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

class Demo {
    int x;
public:
    void setX(int x) {
        this->x = x;
    }
    void show() {
        cout << "x = " << x << endl;
    }
};

int main() {
    Demo d;
    d.setX(100);
    d.show();
    return 0;
}
```

```
START
CREATE class Demo with data member x
DEFINE method setX(x): this.x = x
DEFINE method show(): print x
CREATE object d
CALL d.setX(100)
CALL d.show()
END
```

6. Single Inheritance

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

class Animal {
public:
    void speak() {
        cout << "Animal sound\n";
    }
};

class Dog : public Animal {
public:
    void bark() {
        cout << "Dog barks\n";
    }
};

int main() {
    Dog d;
    d.speak();
    d.bark();
    return 0;
}
```

START
DEFINE class Animal with speak()
DEFINE class Dog inheriting Animal, add bark()
CREATE object of Dog
CALL object.speak()
CALL object.bark()
END

7. Multilevel Inheritance

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

class Vehicle {
public:
    void start() {
        cout << "Vehicle starts\n";
    }
};

class Car : public Vehicle {
public:
    void drive() {
        cout << "Car drives\n";
    }
};

class ElectricCar : public Car {
public:
    void charge() {
        cout << "Charging electric car\n";
    }
};

int main() {
    ElectricCar ec;
    ec.start();
    ec.drive();
    ec.charge();
    return 0;
}
```

START
DEFINE class Vehicle with start()
DEFINE class Car inheriting Vehicle with drive()
DEFINE class ElectricCar inheriting Car with charge()
CREATE ElectricCar object
CALL object.start()
CALL object.drive()
CALL object.charge()
END

8. Function Overriding With & Without Virtual

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

class Animal {
public:
    virtual void speak() {
        cout << "Animal speaks\n";
    }
};

class Dog : public Animal {
public:
    void speak() override {
        cout << "Dog barks\n";
    }
};

int main() {
    Animal* a;
    Dog d;
    a = &d;
    a->speak();  // Calls Dog's speak due to
virtual function
    return 0;
}
```

START
DEFINE base class Animal with virtual speak()
DEFINE derived class Dog with override speak()
CREATE base pointer animalPtr
CREATE Dog object dog
ASSIGN animalPtr = &dog
CALL animalPtr->speak() // Calls Dog’s speak()
END

9. Abstract Class & draw()

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

class Shape {
public:
    virtual void draw() = 0;  // Pure virtual function
};

class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing Circle\n";
    }
};

class Square : public Shape {
public:
    void draw() override {
        cout << "Drawing Square\n";
    }
};
```

```
int main() {
    Circle c;
    Square s;
    Shape* shape;
    shape = &c;
    shape->draw();
    shape = &s;
    shape->draw();
    return 0;
}
```

START
DEFINE abstract class Shape with pure virtual draw()
DEFINE Circle inheriting Shape, implement draw()
DEFINE Square inheriting Shape, implement draw()
CREATE object c, s
CREATE Shape pointer
ASSIGN shape = &c → call draw()
ASSIGN shape = &s → call draw()
END

10. Generic Swap Function

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

template <typename T>
void swapValues(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 10, y = 20;
    swapValues(x, y);
    cout << "Swapped: x = " << x << ", y = " << y << endl;

    double p = 1.1, q = 2.2;
    swapValues(p, q);
    cout << "Swapped: p = " << p << ", q = " << q << endl;

    return 0;
}
```

START
DEFINE template function swapValues(T a, T b)
SWAP a and b using temp
CALL swapValues with int
CALL swapValues with double
PRINT swapped values
END

11. File Write and Read (String)

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

template <typename T>
void swapValues(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 10, y = 20;
    swapValues(x, y);
    cout << "Swapped: x = " << x << ", y = " << y << endl;

    double p = 1.1, q = 2.2;
    swapValues(p, q);
    cout << "Swapped: p = " << p << ", q = " << q << endl;

    return 0;
}
```

START
DECLARE string text = "Hello File Handling"
OPEN file in write mode
WRITE text to file
CLOSE file

OPEN file in read mode
READ text from file
PRINT text
CLOSE file
END