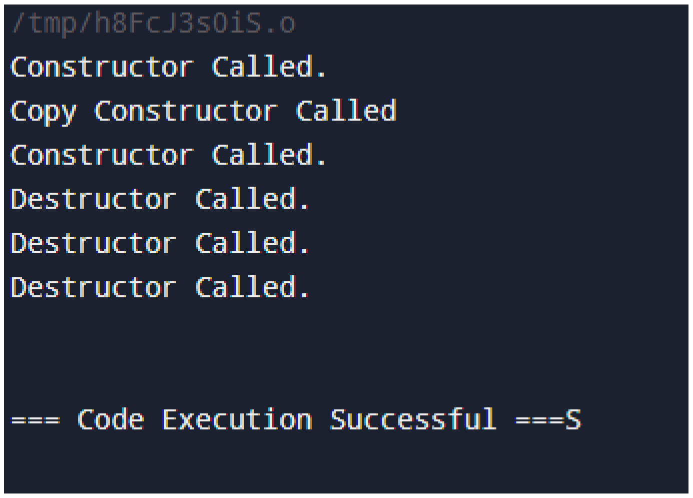
Activity No. 2.1	
ARRAYS, POINTERS AND DYNAMIC MEMORY ALLOCATION	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** September 11, 2024
Section: CPE21S4	**Date Submitted:** September 12, 2024
Name(s): Bona, Andrei Nycole So	**Instructor:** Prof. Maria Rizette Sayo

6. Output

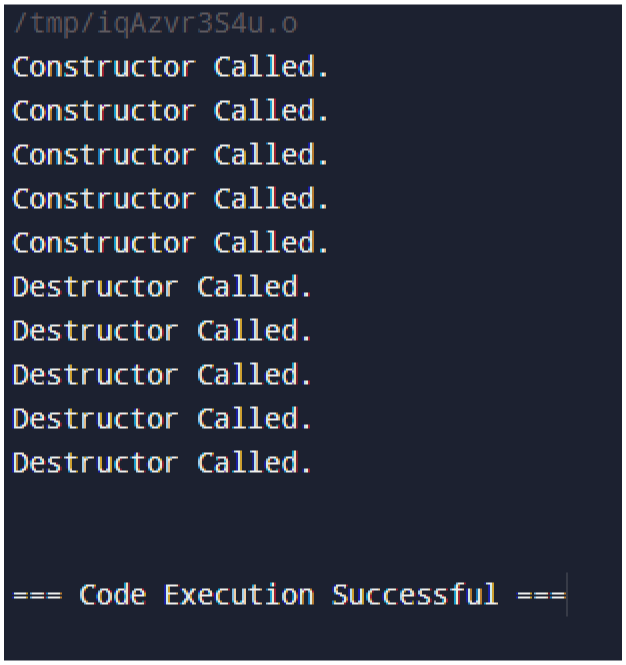
7. Supplementary Activity

ILO A: Implement static and dynamic memory allocation & ILO B: Create dynamically allocated objects using pointers and arrays.

Screenshot	
Observation	The output indicates the sequence of constructor and destructor calls for the Student objects, where the first "Constructor Called" is for student1, the "Copy Constructor Called" is for student2, which is a copy of student1, the second "Constructor Called" is for student3, which is created with default values, and the three "Destructor Called" messages indicate that the objects are being destroyed in the reverse order of their creation, which is the expected behavior.

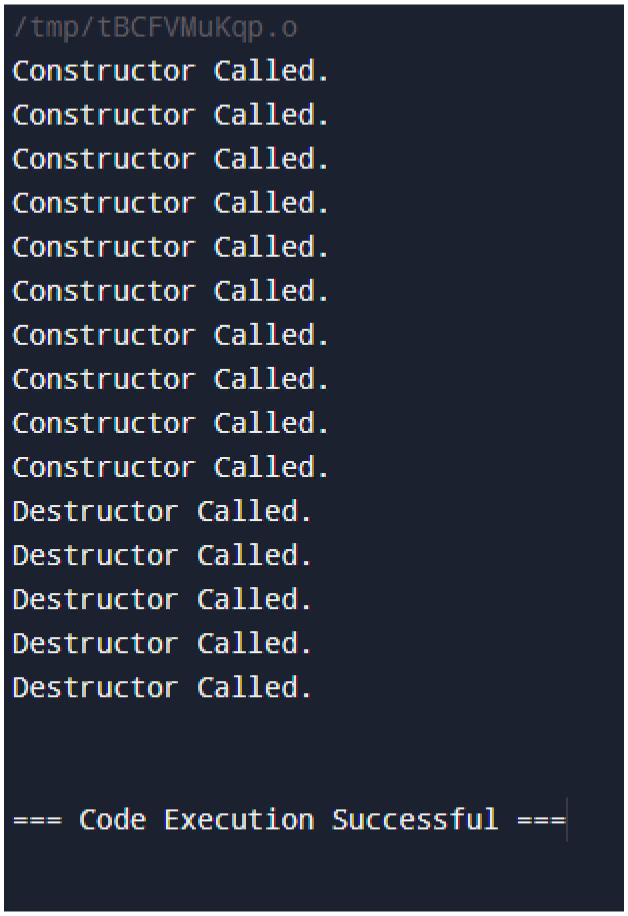
```
/tmp/h8FcJ3sOiS.o
Constructor Called.
Copy Constructor Called
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.


=== Code Execution Successful ===S
```

Table 2-1. Initial Driver Program

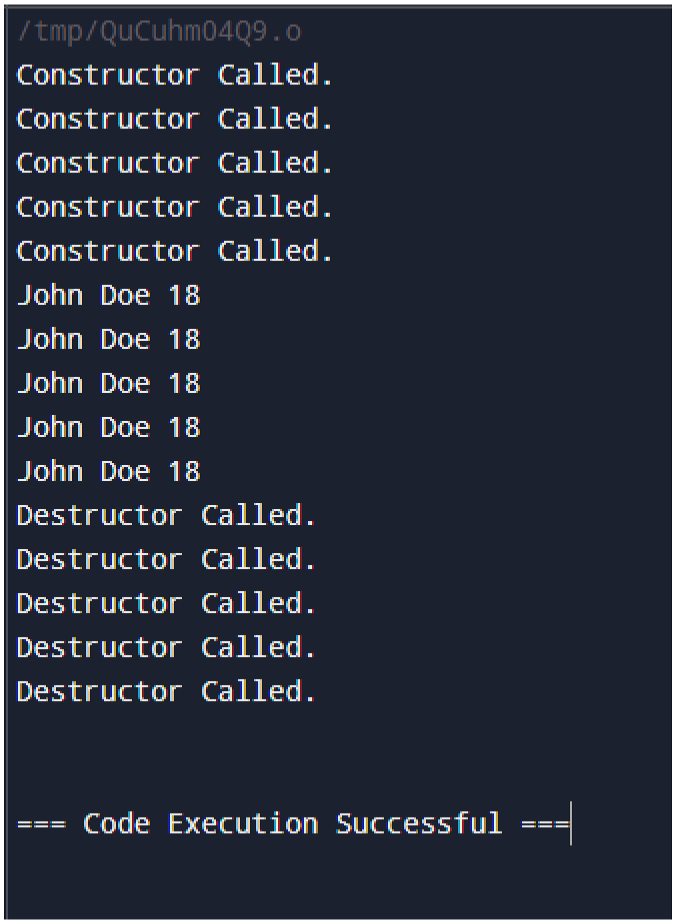
Screenshot	
	```
	/tmp/iqAzvr3S4u.o
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.


	=== Code Execution Successful ===
	```

Observation	The output shows that five student objects are made, and for each one, the program runs a constructor. After that, all five objects are deleted, which causes the program to call a destructor for each.

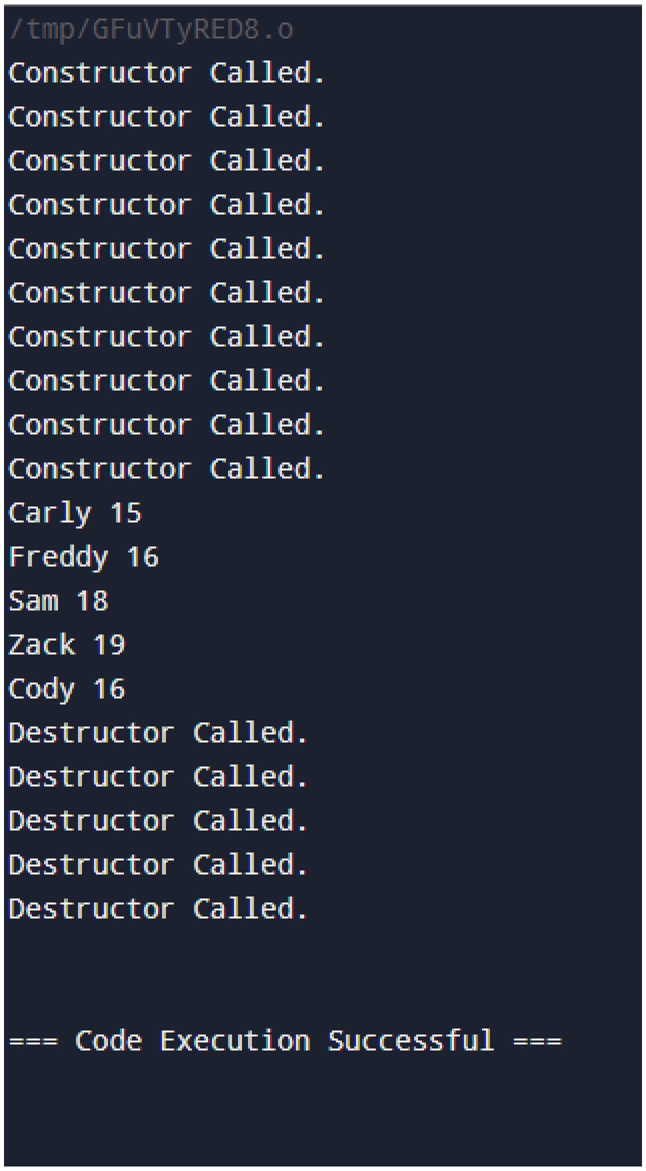
Table 2-2. Modified Driver Program with Student Lists

Loop A	
	```
	/tmp/tBCFVMuKqp.o
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.


	=== Code Execution Successful ===
	```

Observation	The output shows that ten student objects are created, with a constructor running for each one. None of the student names or ages are shown.

Loob B	
	```
	/tmp/QuCuhm04Q9.o
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	John Doe 18
	John Doe 18
	John Doe 18
	John Doe 18
	John Doe 18
	Destructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.


	=== Code Execution Successful ===
	```

Observation	The output shows that five student objects are created, with a constructor running each time. Then, the same student information, "John Doe 18", is shown five times, meaning the same data is being used repeatedly.

Output	
	```
	/tmp/GFuVTyRED8.o
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Constructor Called.
	Carly 15
	Freddy 16
	Sam 18
	Zack 19
	Cody 16
	Destructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.
	Destructor Called.


	=== Code Execution Successful ===
	```

Observation	The output shows that ten student objects are created, and for each one, a constructor is run. Out of those, five are used to display student names and ages.

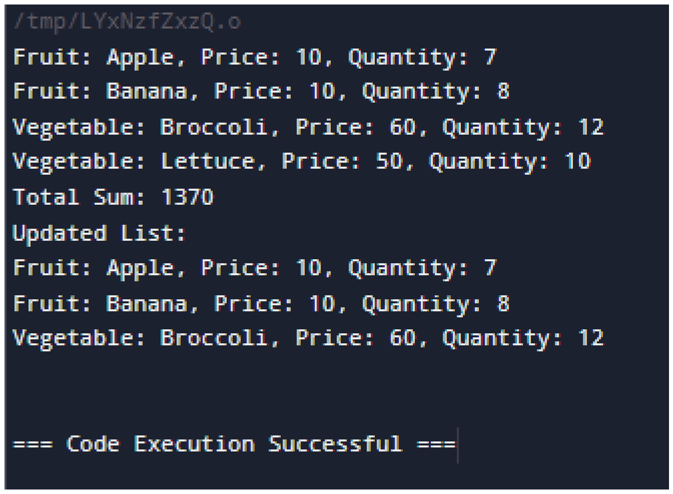
Table 2-3. Final Driver Program

Modifications	n/a
Observation	n/a

Table 2-4. Modifications/Corrections Necessary

ILO C: Solve programming problems using dynamic memory allocation, arrays and pointers

```
#include <iostream>
#include <string>

class Item {
protected:
    std::string name;
    double price;
    int quantity;

public:
    Item(const std::string& n, double p, int q) : name(n), price(p), quantity(q) {}
    virtual ~Item() {}
    Item(const Item& other) : name(other.name), price(other.price), quantity(other.quantity) {}
    Item& operator=(const Item& other) {
        if (this != &other) {
            name = other.name;
            price = other.price;
            quantity = other.quantity;
        }
        return *this;
    }

    virtual void display() const = 0;
    double calculateSum() const { return price * quantity; }
};

class Fruit : public Item {
public:
    Fruit(const std::string& n, double p, int q) : Item(n, p, q) {}
    ~Fruit() {}
    Fruit(const Fruit& other) : Item(other) {}
    Fruit& operator=(const Fruit& other) {
        Item::operator=(other);
        return *this;
    }

    void display() const override {
        std::cout << "Fruit: " << name << ", Price: " << price << ", Quantity: " << quantity << std::endl;
    }
};

class Vegetable : public Item {
public:
    Vegetable(const std::string& n, double p, int q) : Item(n, p, q) {}
    ~Vegetable() {}
    Vegetable(const Vegetable& other) : Item(other) {}
    Vegetable& operator=(const Vegetable& other) {
        Item::operator=(other);
        return *this;
    }

    void display() const override {
        std::cout << "Vegetable: " << name << ", Price: " << price << ", Quantity: " << quantity << std::endl;
    }
};

double TotalSum(Item** list, int size) {
    double sum = 0;
    for (int i = 0; i < size; i++) {
        sum += list[i]->calculateSum();
    }
    return sum;
}

int main() {
    int numItems = 4;
    Item** groceryList = new Item*[numItems];

    groceryList[0] = new Fruit("Apple", 10, 7);
    groceryList[1] = new Fruit("Banana", 10, 8);
    groceryList[2] = new Vegetable("Broccoli", 60, 12);
    groceryList[3] = new Vegetable("Lettuce", 50, 10);

    for (int i = 0; i < numItems; i++) {
        groceryList[i]->display();
    }

    double total = TotalSum(groceryList, numItems);
    std::cout << "Total Sum: " << total << std::endl;

    delete groceryList[3];
    groceryList[3] = nullptr;


    for (int i = 3; i < numItems - 1; i++) {
        groceryList[i] = groceryList[i + 1];
    }


    numItems--;


    std::cout << "Updated List:" << std::endl;
    for (int i = 0; i < numItems; i++) {
        groceryList[i]->display();
    }


    for (int i = 0; i < numItems; i++) {
        delete groceryList[i];
    }
    delete[] groceryList;

    return 0;
}
```



```
/tmp/LYxNzfZxzQ.o
Fruit: Apple, Price: 10, Quantity: 7
Fruit: Banana, Price: 10, Quantity: 8
Vegetable: Broccoli, Price: 60, Quantity: 12
Vegetable: Lettuce, Price: 50, Quantity: 10
Total Sum: 1370
Updated List:
Fruit: Apple, Price: 10, Quantity: 7
Fruit: Banana, Price: 10, Quantity: 8
Vegetable: Broccoli, Price: 60, Quantity: 12


=== Code Execution Successful ===
```

8. Conclusion

The laboratory demonstrates that the system correctly creates and destroys student objects, but may have an issue with storing and displaying individual student information, as it appears to be using the same data for all students.

9. Assessment Rubric