Activity No. 2.1	
Hands-on Activity 2.1 Arrays, Pointers and Dynamic Memory Allocation	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 09/11/2024**
Section: CPE21S1	**Date Submitted: 09/11/2024**
Name(s): Bautista, Jhon Hendricks T.	**Instructor: Maria Rizette Sayo**

6. Output

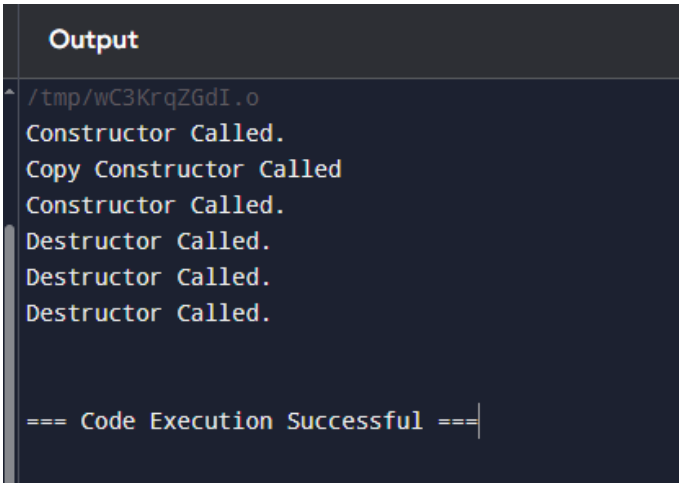
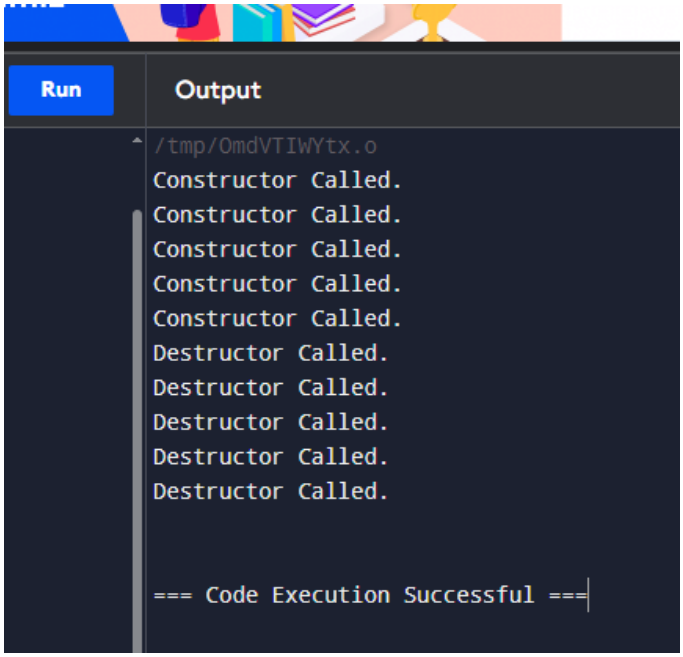
ScreenShot	
Observation	In the output, it shows that the constructor is called successfully.

Table 2.1 Initial Driver Program

Screenshot	

Observation	In this output, the constructor and destructor are called successfully many times because of the many instances we used in the array.

Table 2.2 Modified Driver Program with Student Lists

Loop A	```
	int main() {
	const size_t j = 5;
	Student studentList[j] = {};
	std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"};
	int ageList[j] = {15, 16, 18, 19, 16};
	for(int i = 0; i < j; i++){ //loop A
	Student *ptr = new Student(namesList[i], ageList[i]);
	studentList[i] = *ptr;
	}
	``

Observation	In the first loop it uses the pointer to keep the memory of the array allocated.

Loop B	```
	for(int i = 0; i < j; i++){ //loop B
	studentList[i].printDetails();
	}
	return 0;
	}
	``

Observation	The previously allocated memory is accessed by the loop B

Output	Output /tmp/VUnIDLx8bd.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 arrays are now used for static allocation then we used the class to access them and store it dynamically.

Table 2.3

7. Supplementary Activity

main.cpp

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

using namespace std;

class Fruit {
public:

    Fruit(string name = "", double price = 0.0, int quantity = 0)
        : name_(name), price_(price), quantity_(quantity) {}


    ~Fruit() {
        cout << "Fruit destructor called" << endl;
    }

    Fruit(const Fruit& other)
        : name_(other.name_), price_(other.price_), quantity_(other.quantity_) {
        cout << "Fruit copy constructor called" << endl;
    }

    Fruit& operator=(const Fruit& other) {
        if (this != &other) {
            name_ = other.name_;
            price_ = other.price_;
            quantity_ = other.quantity_;
        }
        cout << "Fruit copy assignment operator called" << endl;
        return *this;
    }

    string name_;
    double price_;
    int quantity_;
    double calculateSum() const {
        return price_ * quantity_;
    }

    void displayInfo() const {
        cout << "Fruit: " << name_ << ", Price: " << price_ << ", Quantity: " << quantity_ << endl;
    }
};

class Vegetable {
public:
```

Output

```
/tmp/ywQcFzdjGN.o
Grocery List of Jenna:
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 cost: 1370

Lettuce removed from the list.

Grocery List after removal of Lettuce:
Fruit: Apple, Price: 10, Quantity: 7
Fruit: Banana, Price: 10, Quantity: 8
Vegetable: Broccoli, Price: 60, Quantity: 12
Vegetable destructor called
Vegetable destructor called
Fruit destructor called
Fruit destructor called


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

8. Conclusion

In this activity I learned about creating classes and constructors in C++. They are very useful in handling data. I learned about dealing with static and dynamic storing of data in an array. There is also the use of pointers to be able to access the memory allocated by the object. The activity takes it step by step to create the program utilizing dynamic memory. I learned that constructors are important to easily put attributes and create objects, while deconstructors are helpful in deallocating objects in the memory.

9. Assessment Rubric