Activity No. 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: CPE21S4	**Date Submitted:** 09-13-2024
Name(s): Masangkay, Frederick D.	**Instructor:** Ma'am RIzette Sayo

6. Output

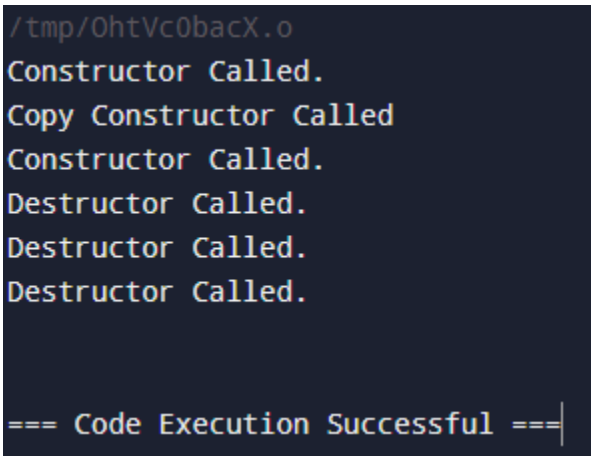
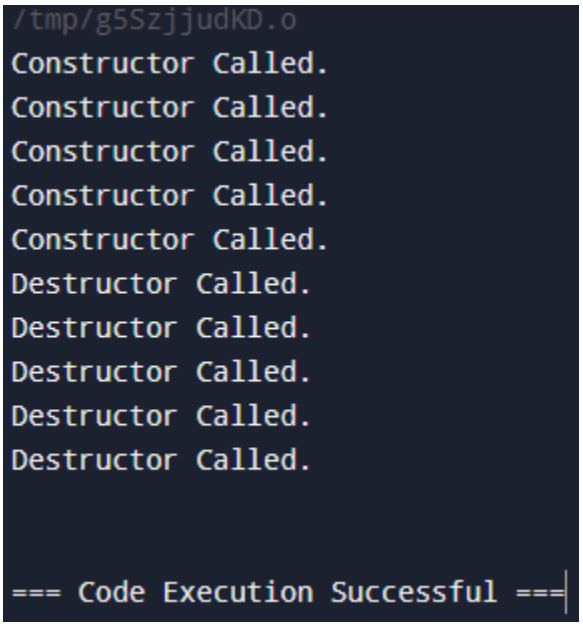
Screenshot:	
Observation:	When student2 is constructed as a copy of student1, the copy constructor is called, which prints "Copy Constructor Called". When student3 is assigned the value of student2, the assignment operator is called, which checks for self-assignment and then copies the values from student2 to student3.

Table 2-1. Initial Driver Program

Screenshot:	

Observation:	The code declares a constant size_t variable j with the value 5.It then declares an array studentList of Student objects with j elements, but it is not initialized with any values.It declares two arrays, namesList and ageList, with j elements, containing names and ages, respectively.

Table 2-2. Modified Driver Program with Student Lists

Loop A	`for(int i = 0; i < j; i++){ //loop A`
	`  Student *ptr = new Student(namesList[i], ageList[i]);`
	`  studentList[i] = *ptr;`
	`}`

Observation	It initializes the studentList array with Student objects, each with a name and age from the corresponding elements of namesList and ageList.

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

Observation	Loop B prints the details of each student in the studentList array to the console.
Output	/tmp/7sLt3g6kcS.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 namesList and ageList arrays are declared but not used effectively. Instead, the Student objects are created using these arrays, but the arrays themselves are not utilized.

Table 2-3. Final Driver Program

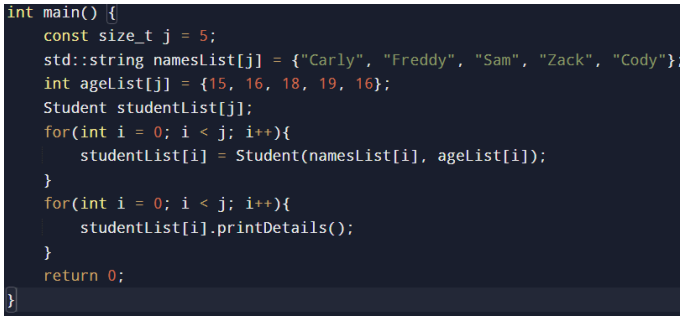
Screenshot:	
Observation:	It removes the unnecessary dynamic memory allocation and redundant assignments, and uses the namesList and ageList arrays to initialize the studentList array directly.

Table 2-4. Modifications/Corrections Necessary

7. Supplementary Activity

Problem 1: Create a class for the fruit and the vegetable classes

```
using namespace std;

class GroceryItem {
public:
    // Constructor
    GroceryItem(string name, double price, int quantity)
        : name_(name), price_(price), quantity_(quantity) {}

    // Destructor
    ~GroceryItem() {
        cout << name_ << " destructor called" << endl;
    }

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

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

    // Attributes
    string name_;
    double price_;
    int quantity_;
```

```
    // Functions
    double calculateSum() {
        return price_ * quantity_;
    }

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

class Fruit : public GroceryItem {
public:
    Fruit(string name, double price, int quantity) : GroceryItem(name, price, quantity) {}
};

class Vegetable : public GroceryItem {
public:
    Vegetable(string name, double price, int quantity) : GroceryItem(name, price, quantity) {}
};
```

Problem 2: Create an array GroceryList in the driver code that will contain all items in Jenna's Grocery List

```
int main() {
    GroceryItem* groceryList[5];

    groceryList[0] = new Fruit("Apple", 1.99, 5);
    groceryList[1] = new Vegetable("Carrot", 0.99, 10);
    groceryList[2] = new Fruit("Banana", 0.99, 7);
    groceryList[3] = new Vegetable("Lettuce", 1.49, 3);
    groceryList[4] = new Fruit("Orange", 2.49, 4);

    for (int i = 0; i < 5; i++) {
        groceryList[i]->displayInfo();
        cout << "Total cost: " << groceryList[i]->calculateSum() << endl;
    }
}
```

Problem 3: Create a function TotalSum that will calculate the sum of all objects listed in Jenna's Grocery List

```
double TotalSum(GroceryItem* groceryList[], int size) {
    double totalSum = 0.0;
    for (int i = 0; i < size; i++) {
        totalSum += groceryList[i]->calculateSum();
    }
    return totalSum;
}

int main() {
    double total = TotalSum(groceryList, 5);
    cout << "Total sum: " << total << endl;
}
```

Problem 4: Delete the Lettuce from Jenna's GroceryList list and de-allocate the memory assigned

```
int main() {
    delete groceryList[3]; // delete the Lettuce object
    groceryList[3] = nullptr; // set the pointer to nullptr to avoid dangling pointer
}
```

8. Conclusion

This activity highlights the importance of careful coding practices, including memory management, efficient use of resources, and effective utilization of variables. It also demonstrates the value of code review and analysis in identifying and addressing potential issues, leading to more robust and maintainable code. After analyzing the code, we identified the problems and provided improvement suggestions to remove the unnecessary dynamic memory allocation and redundant assignments, and to utilize the namesList and ageList arrays more effectively. We also provided an improved version of the code that addressed these issues.

9. Assessment Rubric