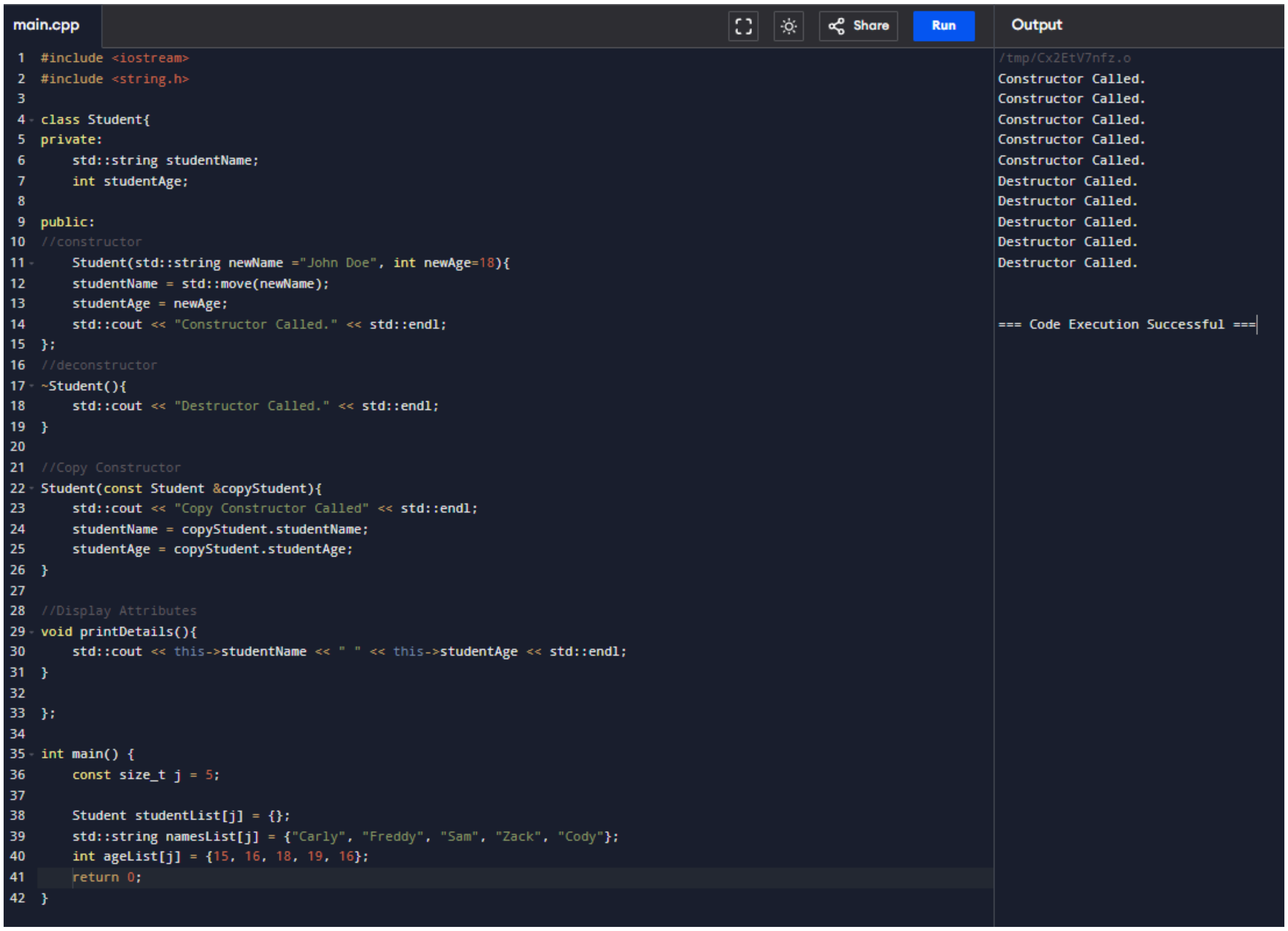
Activity No. 2	
ARRAYS, POINTERS AND DYNAMIC MEMORY ALLOCATION	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: Sept. 11, 2024**
Section: CPE21S4	**Date Submitted: Sept. 12, 2024**
Name(s): San Juan, Edson Ray E.	**Instructor: Prof Rizzette Sayo**

6. Output

Discuss what is done by loop A and loop B in table 2-3. Additionally, discuss the output and whether the functions are working as intended. If any corrections were made, further provide your modification and analysis in table 2-4.

Screenshot:



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

class Student{
private:
    std::string studentName;
    int studentAge;

public:
//constructor
    Student(std::string newName ="John Doe", int newAge=18){
    studentName = std::move(newName);
    studentAge = newAge;
    std::cout << "Constructor Called." << std::endl;
};
//deconstructor
~Student(){
    std::cout << "Destructor Called." << std::endl;
}

//Copy Constructor
Student(const Student &copyStudent){
    std::cout << "Copy Constructor Called" << std::endl;
    studentName = copyStudent.studentName;
    studentAge = copyStudent.studentAge;
}

//Display Attributes
void printDetails(){
    std::cout << this->studentName << " " << this->studentAge << std::endl;
}

};

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};
    return 0;
}
```

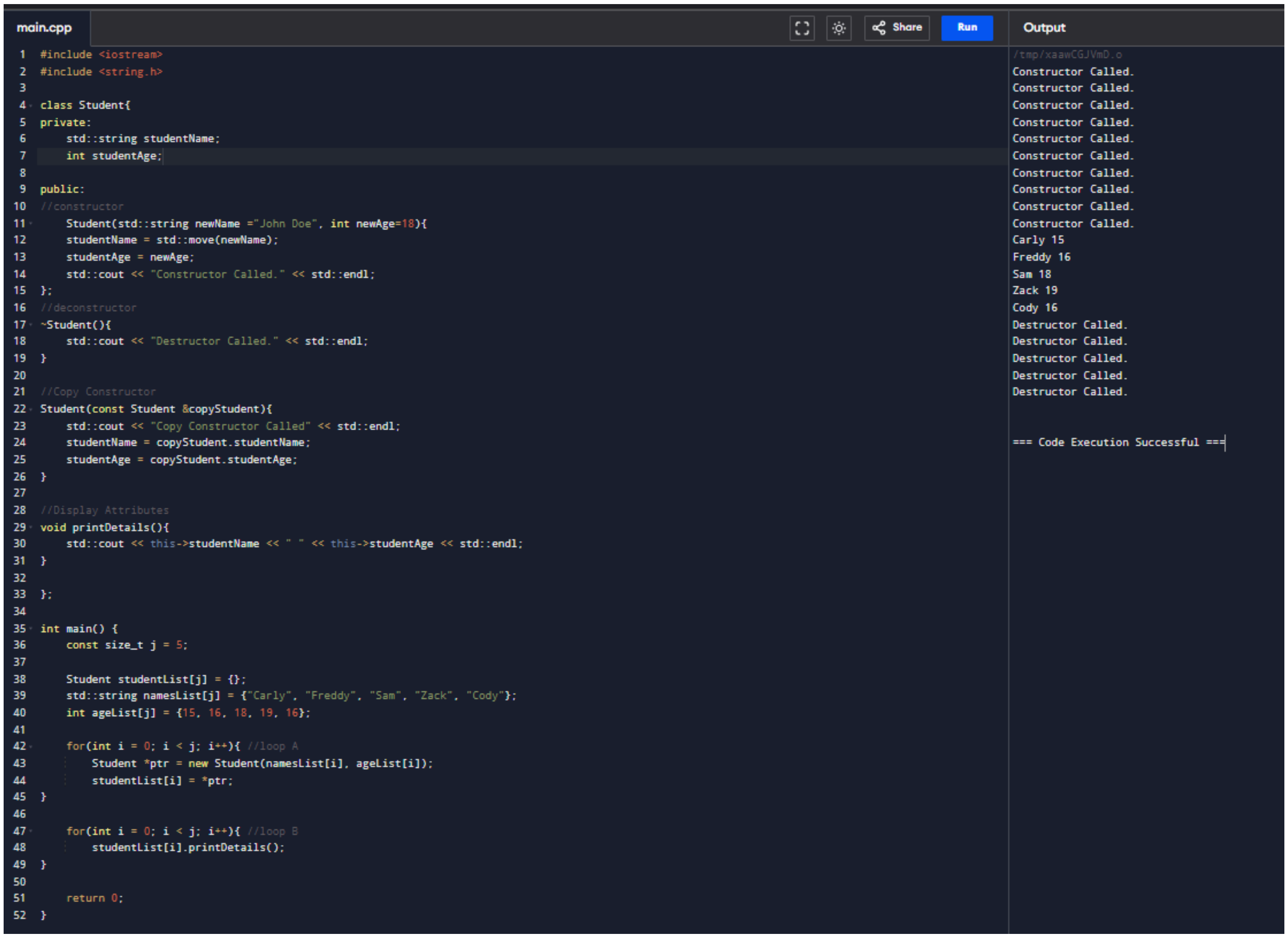
```
/tmp/Cx2EtV7nfz.o
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.


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

Observation:

Based on my observation the code was, if a student name is being called, the output would say Constructor Called, and if there's an empty list the code will say in the output Destructor Called.

Table 2-1. Initial Driver Program

Screenshot:



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

class Student{
private:
    std::string studentName;
    int studentAge;

public:
//constructor
    Student(std::string newName ="John Doe", int newAge=18){
    studentName = std::move(newName);
    studentAge = newAge;
    std::cout << "Constructor Called." << std::endl;
};
//deconstructor
~Student(){
    std::cout << "Destructor Called." << std::endl;
}

//Copy Constructor
Student(const Student &copyStudent){
    std::cout << "Copy Constructor Called" << std::endl;
    studentName = copyStudent.studentName;
    studentAge = copyStudent.studentAge;
}

//Display Attributes
void printDetails(){
    std::cout << this->studentName << " " << this->studentAge << std::endl;
}

};

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;
}

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

    return 0;
}
```

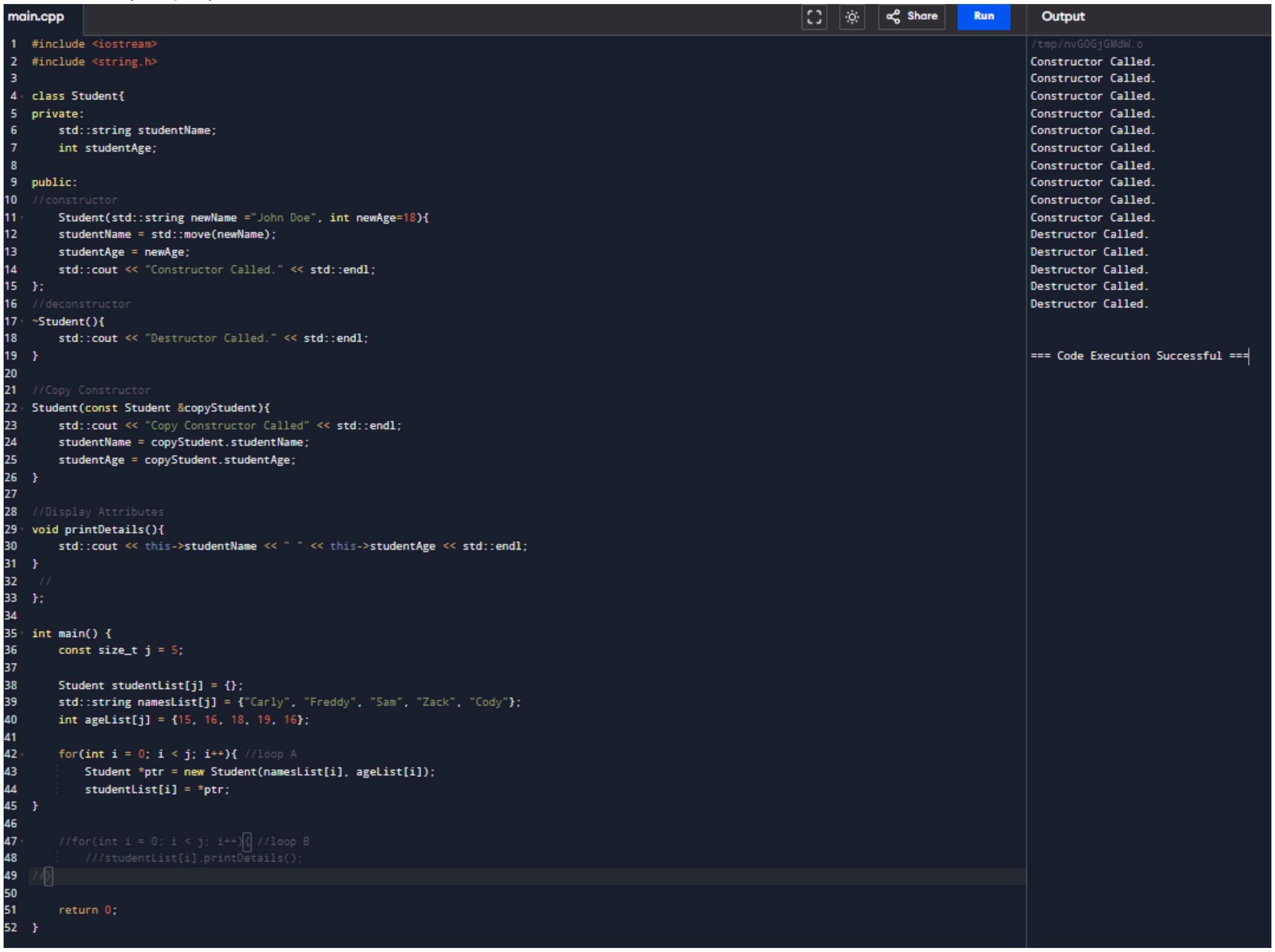
```
/tmp/xaawCGJVmD.o
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Carly 15
Freddy 16
Sam 18
Zack 19
Cody 16
Destructor Called.
Destructor Called.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


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

Observation:

The program defines a Studentclass with constructors and destructors, creating five Student objects initialized with default values of "John Doe" and age 18, while two arrays containing names and ages are declared but not used. At the end, destructors are called for each object.

Table 2-2. Modified Driver Program with Student Lists

Screenshot (Loop A):



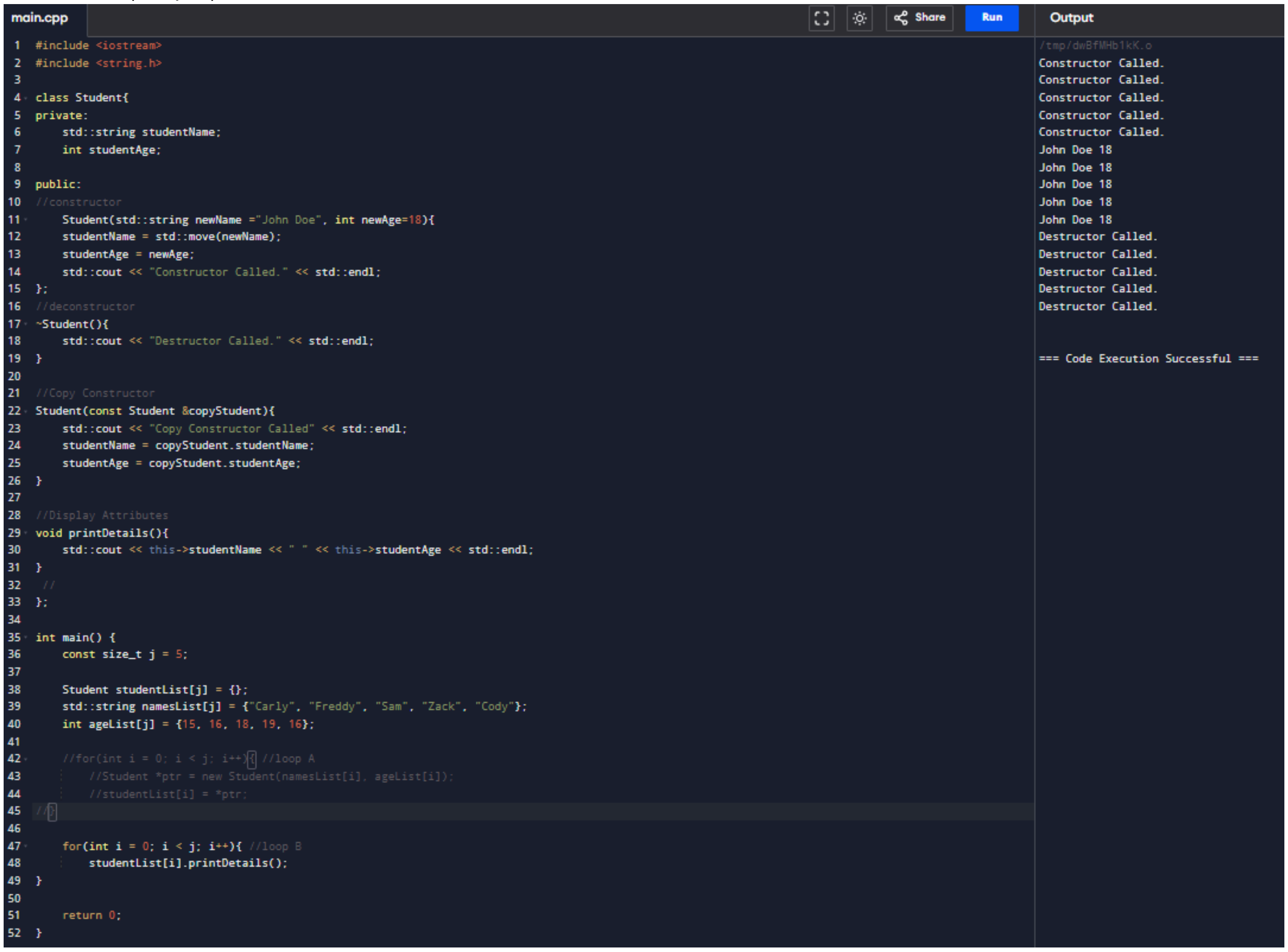
```cpp
#include <iostream>
#include <string.h>

class Student{
private:
    std::string studentName;
    int studentAge;

public:
//constructor
    Student(std::string newName ="John Doe", int newAge=18){
    studentName = std::move(newName);
    studentAge = newAge;
    std::cout << "Constructor Called." << std::endl;
};
//deconstructor
~Student(){
    std::cout << "Destructor Called." << std::endl;
}

//Copy Constructor
Student(const Student &copyStudent){
    std::cout << "Copy Constructor Called" << std::endl;
    studentName = copyStudent.studentName;
    studentAge = copyStudent.studentAge;
}

//Display Attributes
void printDetails(){
    std::cout << this->studentName << " " << this->studentAge << std::endl;
}
 //
};

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;
}

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

    return 0;
}
```

```
/tmp/nvG0GjGMdW.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:

Base on my observation, the students name and age are not being outputted in the code, because it is not being called. And the code automatically prints Constructor Called in the list of name and age.

Screenshot (Loop B):



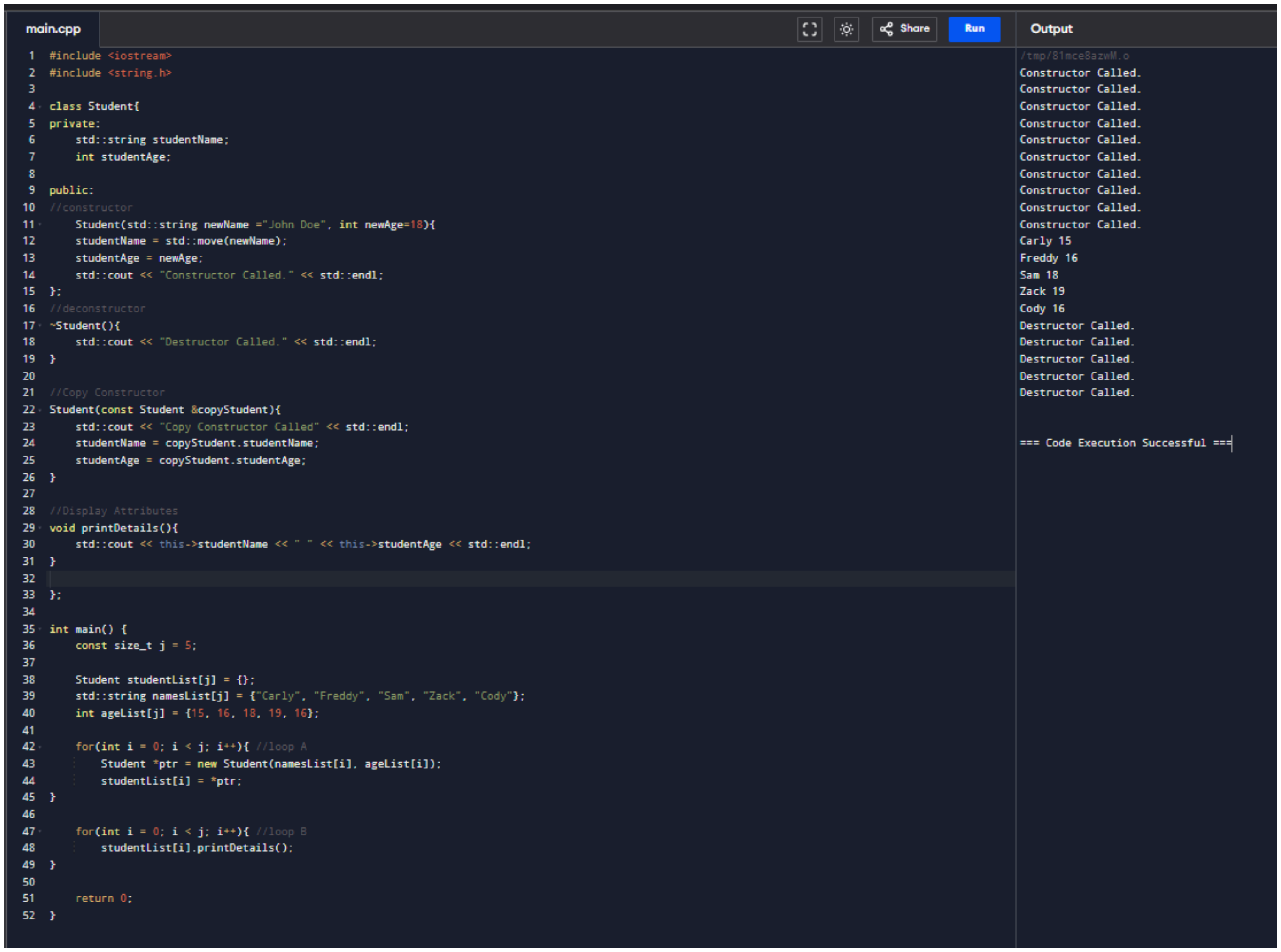
```cpp
#include <iostream>
#include <string.h>

class Student{
private:
    std::string studentName;
    int studentAge;

public:
//constructor
    Student(std::string newName ="John Doe", int newAge=18){
    studentName = std::move(newName);
    studentAge = newAge;
    std::cout << "Constructor Called." << std::endl;
};
//deconstructor
~Student(){
    std::cout << "Destructor Called." << std::endl;
}

//Copy Constructor
Student(const Student &copyStudent){
    std::cout << "Copy Constructor Called" << std::endl;
    studentName = copyStudent.studentName;
    studentAge = copyStudent.studentAge;
}

//Display Attributes
void printDetails(){
    std::cout << this->studentName << " " << this->studentAge << std::endl;
}
 //
};

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;
//}

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

    return 0;
}
```

```
/tmp/dwBfMHb1kK.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:

Based on my observation on loop b the program only prints one name and until the loop stops. because the program cannot store and read the data. Due to this logical error the program only outputs the default name and age. Concluding this it is lacking a pointer syntax that stores the memory addresses of other variables.

Output:



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

class Student{
private:
    std::string studentName;
    int studentAge;

public:
//constructor
    Student(std::string newName ="John Doe", int newAge=18){
    studentName = std::move(newName);
    studentAge = newAge;
    std::cout << "Constructor Called." << std::endl;
};
//deconstructor
~Student(){
    std::cout << "Destructor Called." << std::endl;
}

//Copy Constructor
Student(const Student &copyStudent){
    std::cout << "Copy Constructor Called" << std::endl;
    studentName = copyStudent.studentName;
    studentAge = copyStudent.studentAge;
}

//Display Attributes
void printDetails(){
    std::cout << this->studentName << " " << this->studentAge << std::endl;
}

};

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;
}

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

    return 0;
}
```

```
/tmp/81mce8azwM.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:

Base on my observation, In loop A the students name and age are not being outputted in the code, because it is not being called. And the code automatically prints Constructor Called in the list of name and age. In loop B Based on my observation, the program only prints one name and until the loop stops. because the program cannot store and read the data

Table 2-3. Final Driver Program

7. Supplementary Activity

Jenna wants to buy the following fruits and vegetables for her daily consumption. However, she needs to distinguish between fruit and vegetable, as well as calculate the sum of prices that she has to pay in total.

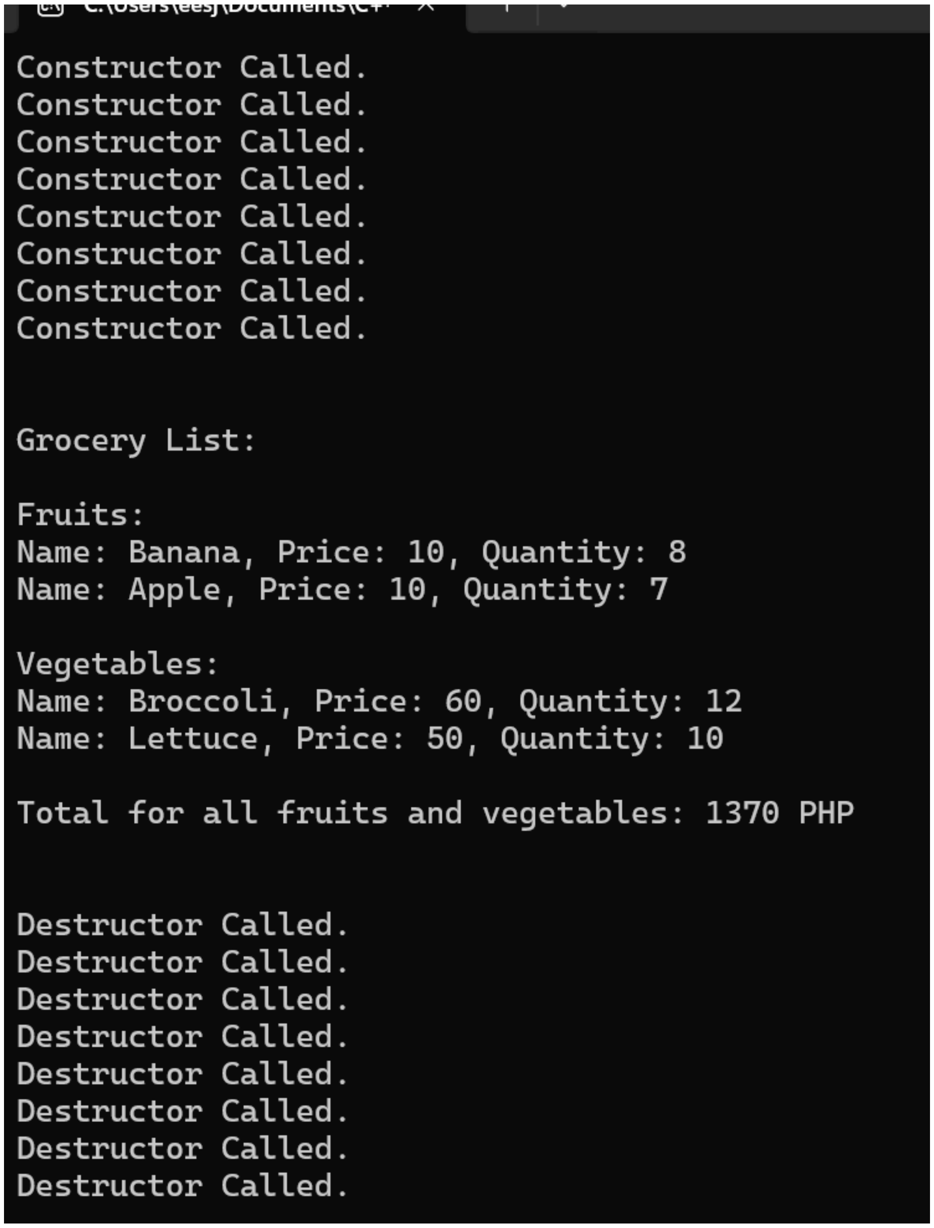
Problem 1: Create a class for the fruit and the vegetable classes. Each class must have a constructor, deconstructor, copy constructor and copy assignment operator. They must also have all relevant attributes (such as name, price and quantity) and functions (such as calculate sum) as presented in the problem description above.
Problem 2: Create an array GroceryList in the driver code that will contain all items in Jenna's Grocery List. You must then access each saved instance and display all details about the items.
Problem 3: Create a function TotalSum that will calculate the sum of all objects listed in Jenna's Grocery List.

(Joined all 3 problem in the code below)

```
#include<iostream>
#include <string.h>
using namespace std;

class Fruit_and_Vegetable {
        private:
                string name;
                int price;
                int quantity;

        public:
                // Constructor
                Fruit_and_Vegetable(string name =" ", int price=0, int quantity =0):
name(name), price(price), quantity(quantity) {
                std::cout << "Constructor Called." << std::endl;
        };

        // Deconstructor
        ~Fruit_and_Vegetable(){
                std::cout << "Destructor Called." << std::endl;
                }

        // Copy Constructor
        Fruit_and_Vegetable(const Fruit_and_Vegetable &other){
                std::cout << "Copy Constructor Called" << std::endl;
                name = other.name;
                price = other.price;
                quantity = other.quantity;
                }

        // Copy Assignment Operator
                Fruit_and_Vegetable& operator=(const Fruit_and_Vegetable &other) {
                        cout << "Copy Assignment Operator Called" << endl;
                        if (this == &other) return *this; // check for self-assignment
                        this->name = other.name;
                        this->price = other.price;
                        this->quantity = other.quantity;
                        return *this;
                }
        // Calculating the sum
        int calculateSum() {
                        return price * quantity;

        }
        // Display Attributes
        void printDetails(){
```

```
		std::cout << "Name: " << this->name << ", Price: " << this->price << ",
Quantity: " << this-> quantity<< std::endl;
	}

	};

class Fruit : public Fruit_and_Vegetable {
	public:
		// Constructor
		Fruit(string name =" ", int price=0, int quantity =0): Fruit_and_Vegetable(name,
price, quantity) {
		std::cout << "Constructor Called." << std::endl;
	};

	// Deconstructor
	~Fruit(){
		std::cout << "Destructor Called." << std::endl;
		}

	// Copy Constructor
	Fruit(const Fruit& other):Fruit_and_Vegetable(other){
		std::cout << "Copy Constructor Called" << std::endl;
		}

	// Copy Assignment Operator
		Fruit& operator=(const Fruit &other) {
			Fruit_and_Vegetable::operator=(other);
			cout << "Copy Assignment Operator Called" << endl;
			return* this;
		}
};

class Vegetable : public Fruit_and_Vegetable{
	public:
		// Constructor
		Vegetable(string name =" ", int price=0, int quantity =0):
Fruit_and_Vegetable(name, price, quantity){
		std::cout << "Constructor Called." << std::endl;
	};

	// Deconstructor
	~Vegetable(){
		std::cout << "Destructor Called." << std::endl;
		}

	// Copy Constructor
	Vegetable(const Vegetable& other):Fruit_and_Vegetable(other){
		std::cout << "Copy Constructor Called" << std::endl;
		}
```

```
        // Copy Assignment Operator
                Vegetable& operator=(const Vegetable &other) {
                        Fruit_and_Vegetable::operator=(other);
                        cout << "Copy Assignment Operator Called" << endl;
                        return* this;
                }

};

int main() {
        const size_t fruitCount = 2;
        const size_t vegetableCount = 2;

        Fruit fruitList[fruitCount] = {
                Fruit("Banana", 10, 8),
                Fruit("Apple", 10, 7)
        };

        Vegetable vegetableList[vegetableCount] = {
                Vegetable("Broccoli", 60, 12),
                Vegetable("Lettuce", 50, 10)
        };
        cout<< "\n" << endl;

        cout << "Grocery List:" << endl;

        int totalFruit = 0;
        int totalVegetable = 0;

        cout << "\nFruits:" << endl;
        for (size_t i = 0; i < fruitCount; ++i) {
                fruitList[i].printDetails();
                totalFruit += fruitList[i].calculateSum();
        }

        cout << "\nVegetables:" << endl;
        for (size_t i = 0; i < vegetableCount; ++i) {
                vegetableList[i].printDetails();
                totalVegetable += vegetableList[i].calculateSum();  // Accumulate the total
        }

        int grandTotal = totalFruit + totalVegetable;
        cout << "\nTotal for all fruits and vegetables: " << grandTotal << " PHP" << endl;

                cout<< "\n"<< endl;

        return 0;
}
```



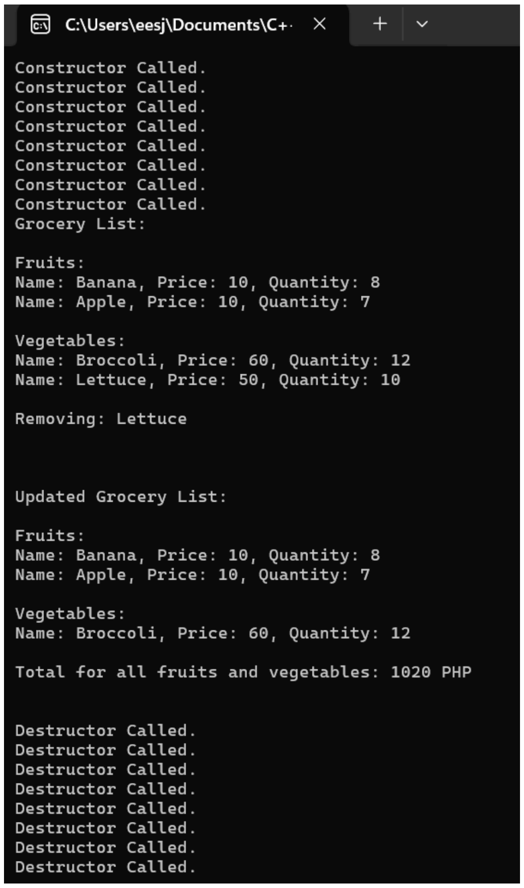
```
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.


Grocery List:

Fruits:
Name: Banana, Price: 10, Quantity: 8
Name: Apple, Price: 10, Quantity: 7

Vegetables:
Name: Broccoli, Price: 60, Quantity: 12
Name: Lettuce, Price: 50, Quantity: 10

Total for all fruits and vegetables: 1370 PHP


Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

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

```
#include <iostream>
#include <string.h>
using namespace std;

class Fruit_and_Vegetable {
private:
    string name;
    int price;
    int quantity;
```

```
public:
    // Constructor
    Fruit_and_Vegetable(string name = " ", int price = 0, int quantity = 0)
        : name(name), price(price), quantity(quantity) {
        std::cout << "Constructor Called." << std::endl;
    }

    // Destructor
    ~Fruit_and_Vegetable() {
        std::cout << "Destructor Called." << std::endl;
    }

    // Copy Constructor
    Fruit_and_Vegetable(const Fruit_and_Vegetable &other) {
        std::cout << "Copy Constructor Called" << std::endl;
        name = other.name;
        price = other.price;
        quantity = other.quantity;
    }

    // Copy Assignment Operator
    Fruit_and_Vegetable& operator=(const Fruit_and_Vegetable &other) {
        cout << "Copy Assignment Operator Called" << endl;
        if (this == &other) return *this; // check for self-assignment
        this->name = other.name;
        this->price = other.price;
        this->quantity = other.quantity;
        return *this;
    }

    // Calculating the sum
    int calculateSum() const {
        return price * quantity;
    }

    // Display Attributes
    void printDetails() const {
        std::cout << "Name: " << this->name << ", Price: " << this->price << ", Quantity: " <<
this->quantity << std::endl;
    }

    // Get name (getter)
    string getName() const {
        return name;
    }
};

class Fruit : public Fruit_and_Vegetable {
public:
    // Constructor
    Fruit(string name = " ", int price = 0, int quantity = 0)
        : Fruit_and_Vegetable(name, price, quantity) {
        std::cout << "Constructor Called." << std::endl;
    }

    // Destructor
    ~Fruit() {
        std::cout << "Destructor Called." << std::endl;
    }

    // Copy Constructor
    Fruit(const Fruit& other) : Fruit_and_Vegetable(other) {
        std::cout << "Copy Constructor Called" << std::endl;
    }

    // Copy Assignment Operator
    Fruit& operator=(const Fruit &other) {
        Fruit_and_Vegetable::operator=(other);
        cout << "Copy Assignment Operator Called" << endl;
        return *this;
    }
};

class Vegetable : public Fruit_and_Vegetable {
public:
    // Constructor
    Vegetable(string name = " ", int price = 0, int quantity = 0)
        : Fruit_and_Vegetable(name, price, quantity) {
        std::cout << "Constructor Called." << std::endl;
    }

    // Destructor
    ~Vegetable() {
        std::cout << "Destructor Called." << std::endl;
    }

    // Copy Constructor
    Vegetable(const Vegetable& other) : Fruit_and_Vegetable(other) {
        std::cout << "Copy Constructor Called" << std::endl;
    }

    // Copy Assignment Operator
    Vegetable& operator=(const Vegetable &other) {
        Fruit_and_Vegetable::operator=(other);
        cout << "Copy Assignment Operator Called" << endl;
        return *this;
    }
};
```

```
int main() {
    const size_t initialFruitCount = 2;
    const size_t initialVegetableCount = 2;

    Fruit fruitList[initialFruitCount] = {
        Fruit("Banana", 10, 8),
        Fruit("Apple", 10, 7)
    };

    Vegetable vegetableList[initialVegetableCount] = {
        Vegetable("Broccoli", 60, 12),
        Vegetable("Lettuce", 50, 10)
    };

    cout << "Grocery List:" << endl;

    int totalFruit = 0;
    int totalVegetable = 0;

    cout << "\nFruits:" << endl;
    for (size_t i = 0; i < initialFruitCount; ++i) {
        fruitList[i].printDetails();
        totalFruit += fruitList[i].calculateSum();
    }

    cout << "\nVegetables:" << endl;
    for (size_t i = 0; i < initialVegetableCount; ++i) {
        vegetableList[i].printDetails();
        totalVegetable += vegetableList[i].calculateSum();
    }

    // Remove lettuce
    int newVegetableCount = initialVegetableCount;
    int indexRemove = -1;
    for (size_t i = 0; i < newVegetableCount; ++i) {
        if (vegetableList[i].getName() == "Lettuce") {
            indexRemove = i;
            break;
        }
    }

    if (indexRemove != -1) {
        cout << "\nRemoving: " << vegetableList[indexRemove].getName() << endl;
        for (size_t i = indexRemove; i < newVegetableCount - 1; ++i) {
            vegetableList[i] = vegetableList[i + 1];
        }
        --newVegetableCount;  // Decrease the vegetable count
    }
    totalVegetable = 0; // Reset vegetable total
    cout << "\n" << endl;
    cout << "\nUpdated Grocery List:" << endl;

    cout << "\nFruits:" << endl;
    for (size_t i = 0; i < initialFruitCount; ++i) {
        fruitList[i].printDetails();
        totalFruit += fruitList[i].calculateSum();
    }

    cout << "\nVegetables:" << endl;
    for (size_t i = 0; i < newVegetableCount; ++i) {
        vegetableList[i].printDetails();
        totalVegetable += vegetableList[i].calculateSum();
    }

    int grandTotal = totalFruit + totalVegetable;
    cout << "\nTotal for all fruits and vegetables: " << grandTotal << " PHP"<< endl;
    cout << "\n" << endl;

    return 0;
}
```



C:\Users\eesj\Documents\C+·

```
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Grocery List:

Fruits:
Name: Banana, Price: 10, Quantity: 8
Name: Apple, Price: 10, Quantity: 7

Vegetables:
Name: Broccoli, Price: 60, Quantity: 12
Name: Lettuce, Price: 50, Quantity: 10

Removing: Lettuce



Updated Grocery List:

Fruits:
Name: Banana, Price: 10, Quantity: 8
Name: Apple, Price: 10, Quantity: 7

Vegetables:
Name: Broccoli, Price: 60, Quantity: 12

Total for all fruits and vegetables: 1020 PHP


Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

8. Conclusion

Overall, In this activity, I learned how to design and implement class with constructors, destructors, and copy operations, as well as how to use inheritance to create specialized classes. I practiced managing arrays of objects, iterating over them, and performing operations like deletion and recalculation of totals. In the supplementary activity a simple inventory management system is implemented using C++ classes to represent fruits and vegetables. The Fruit_and_Vegtable base class handles common attributes and operations, while derived classes Fruit and Vegetable inherit and extend its functionality. The program initializes arrays of Fruit and Vegetable objects, calculates and displays their totals, and demonstrates how to remove an item (Lettuce) from the vegetable list. After removal, it recalculates and displays the updated totals. Output shows that several objects are being created, as indicated by multiple constructor calls. Finally, multiple destructor calls indicate that objects are being destroyed, suggesting the program was finished.

9. Assessment Rubric