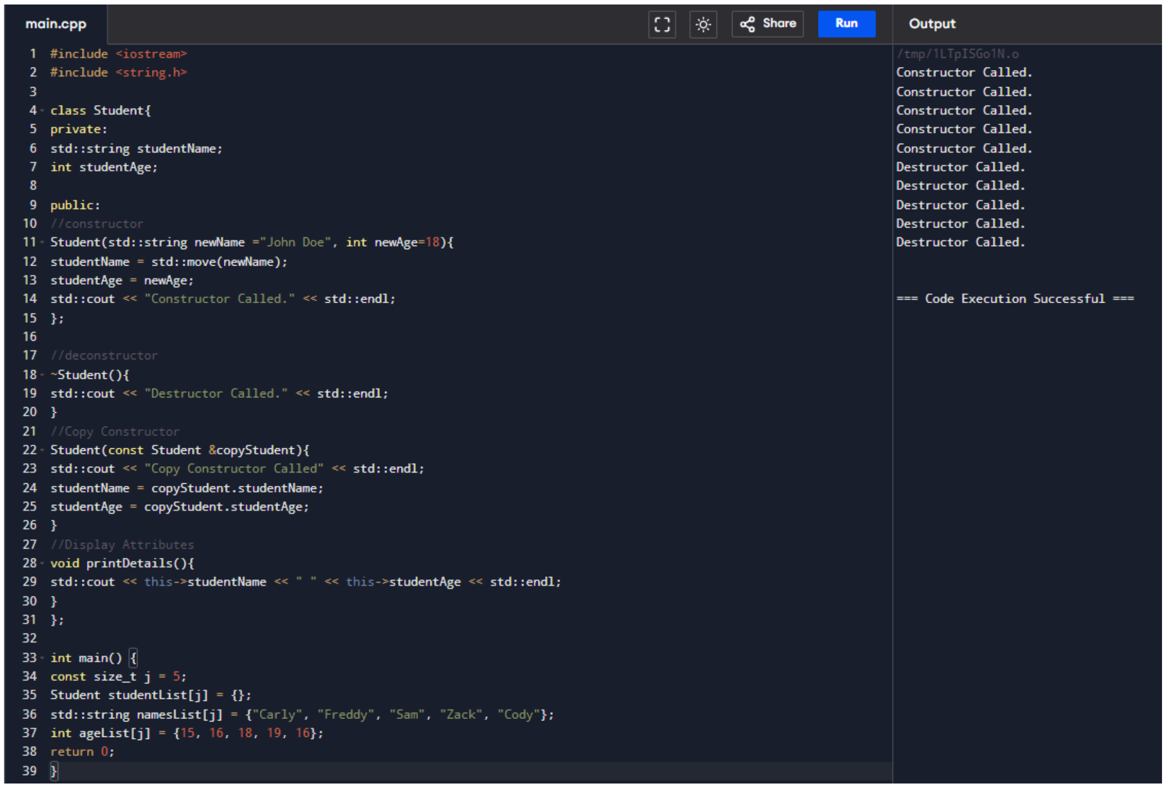
Activity No. 2	
Arrays, Pointers, and Dynamic Memory Allocation	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 09/11/24
Section: CPE21S4	**Date Submitted:** 09/11/24
Name(s): Magistrado, Aira Pauleen M.	**Instructor: Maria Rizette Sayo**
6. Output	

Screenshot

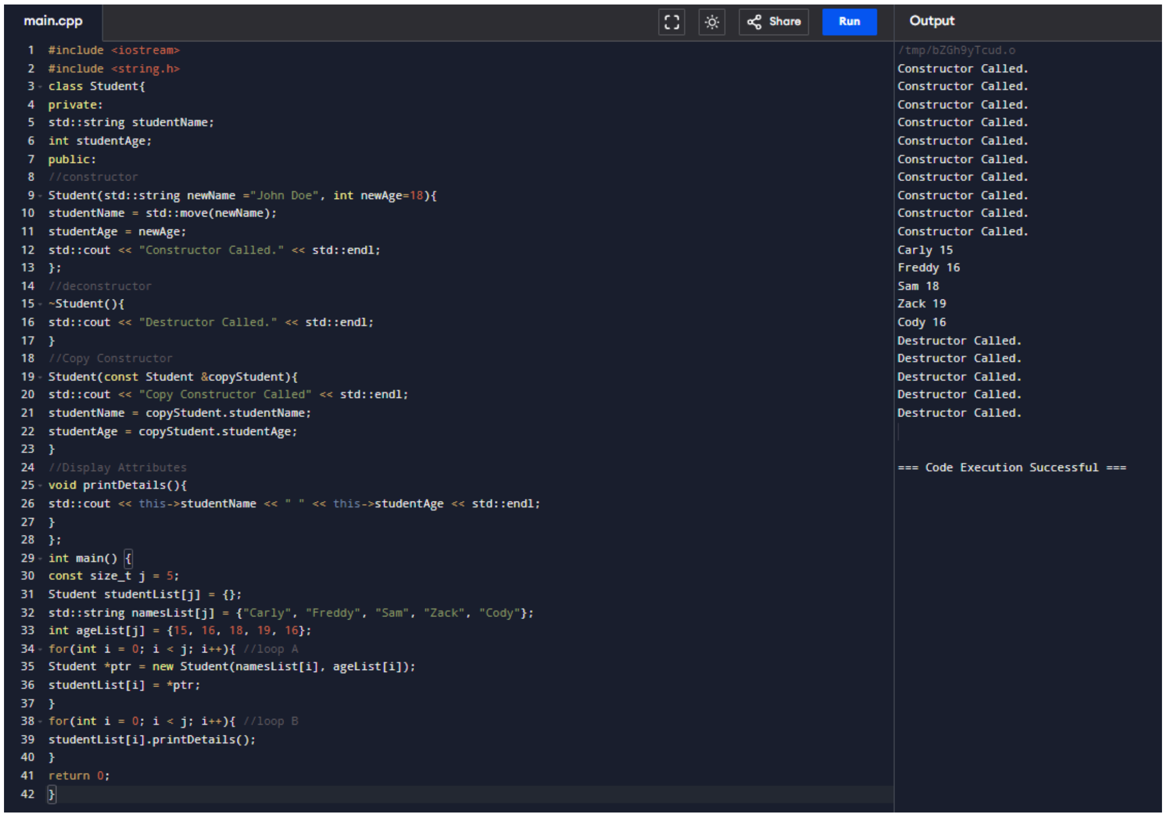


```cpp
#include <iostream>
#include <string.h>

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/1LTpISGo1N.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 code shows that whenever a Student object is created it will output Constructor Called

Table 2-1. Initial Driver Program

Screenshot	
Observation	In each Student it outputs Constructor Called and at the end of the program it outputs Destructor Called. Every student in studentList is assigned a name and age from the namesList and ageList arrays.

main.cpp

```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/bZGh9yTcud.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 ===
```

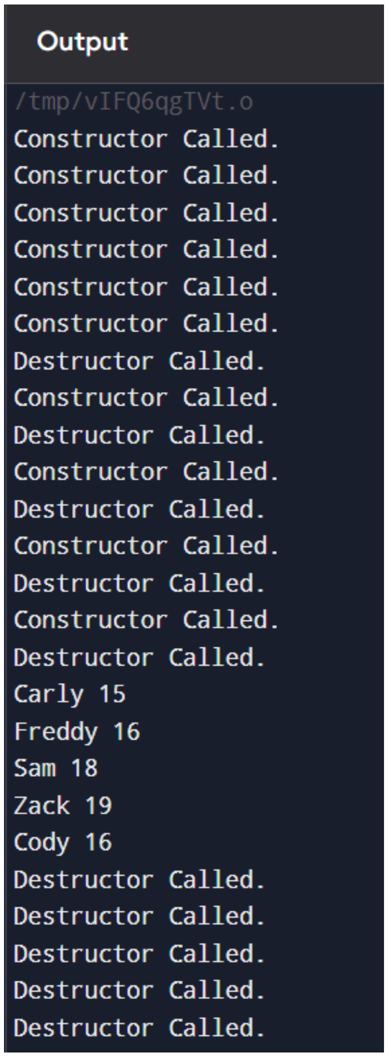
Table 2-2. Modified Driver Program with Student Lists

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

Observation	A list of students is generated based on the namesList and ageList arrays, and then saves each student to the studentList array.

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

Observation	It loops through the studentList array and calls the printDetails() method for each Student

Output	 Output
	`/tmp/vIFQ6qgTVt.o`
	`Constructor Called.`
	`Constructor Called.`
	`Constructor Called.`
	`Constructor Called.`
	`Constructor Called.`
	`Constructor Called.`
	`Destructor Called.`
	`Constructor Called.`
	`Destructor Called.`
	`Constructor Called.`
	`Destructor Called.`
	`Constructor Called.`
	`Destructor Called.`
	`Constructor Called.`
	`Destructor Called.`
	`Carly 15`
	`Freddy 16`
	`Sam 18`
	`Zack 19`
	`Cody 16`
	`Destructor Called.`
	`Destructor Called.`
	`Destructor Called.`
	`Destructor Called.`
	`Destructor Called.`

Observation	It outputs the name and age of each student stored in the array

Table 2-3. Final Driver Program

7. Supplementary Activity

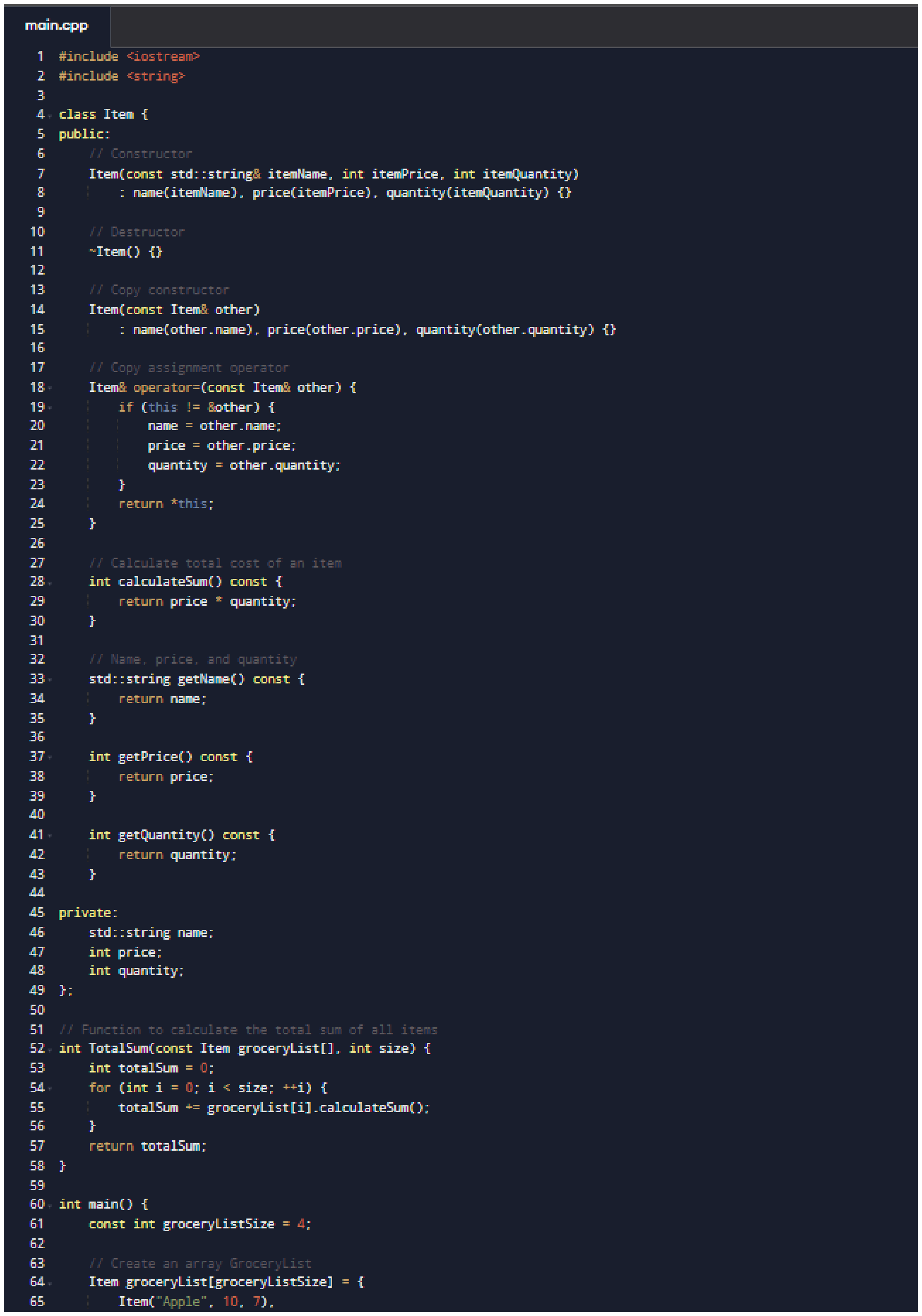
Jenna wants to buy the following fruits and vegetables for her daily consumption. However, she needs to distinguish between fruit and vegetable, as well as calculate the sum of prices that she has to pay in total.

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.

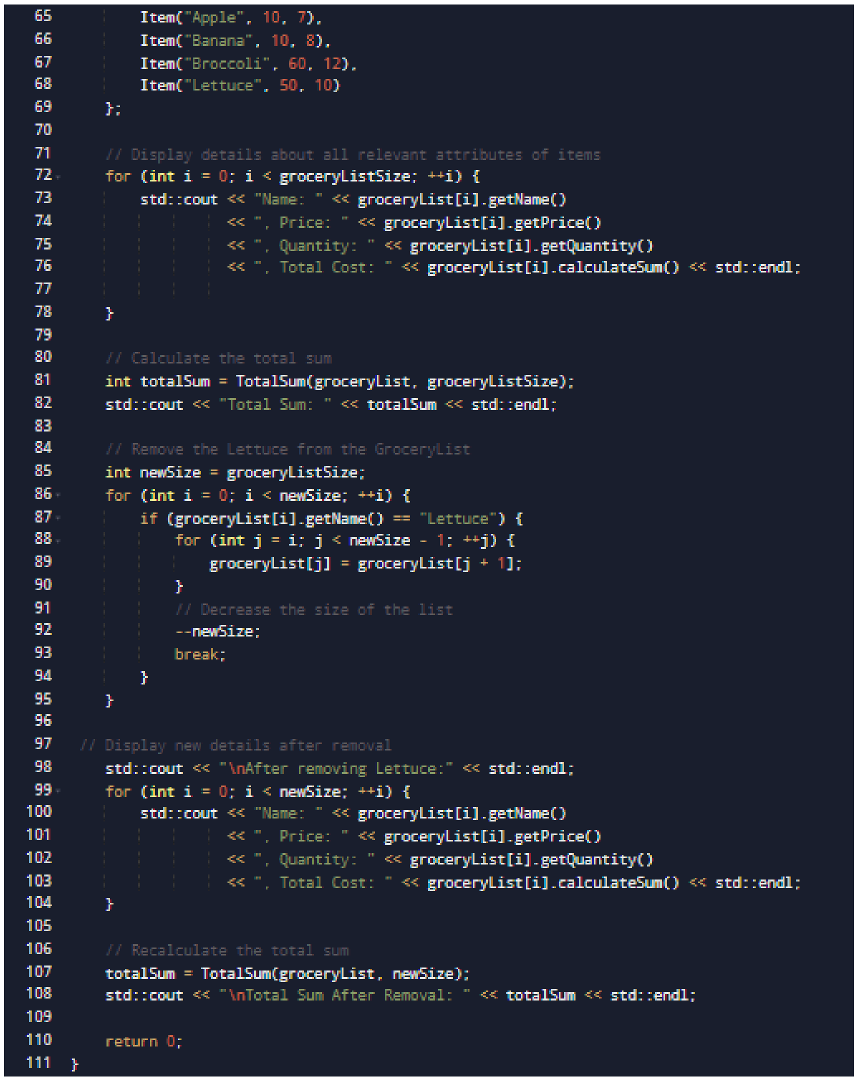
Problem 2: Create an array GroceryList in the driver code that will contain all items in Jenna's Grocery List. You must then access each saved instance and display all details about the items.

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

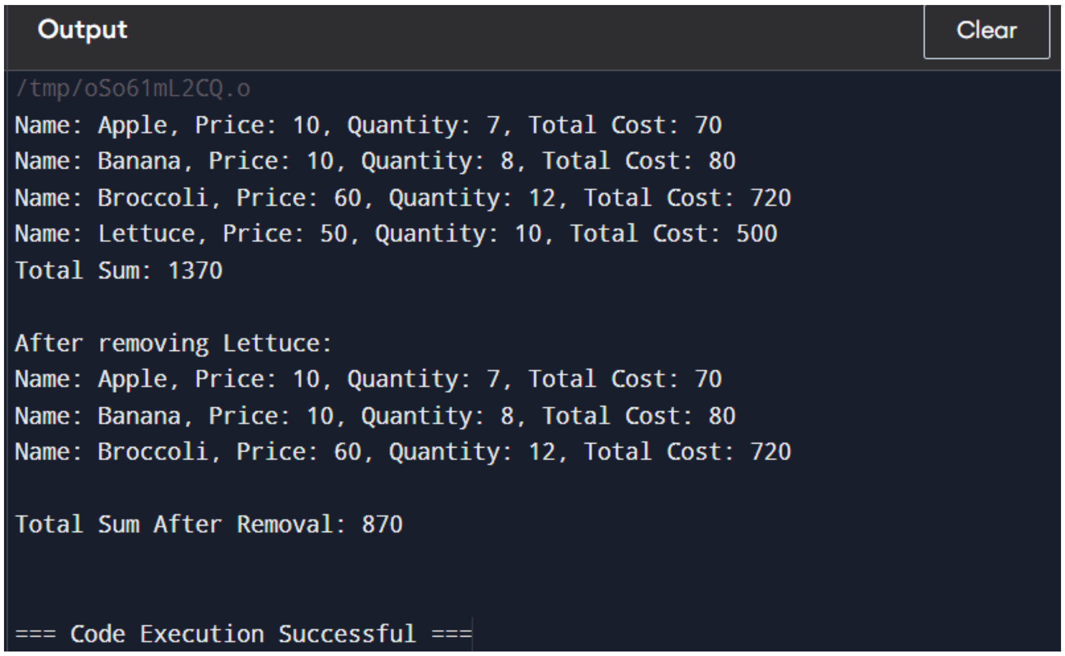
Problem 4: Delete the Lettuce from Jenna's GroceryList list and de-allocate the memory assigned.


main.cpp

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

class Item {
public:
    // Constructor
    Item(const std::string& itemName, int itemPrice, int itemQuantity)
        : name(itemName), price(itemPrice), quantity(itemQuantity) {}

    // Destructor
    ~Item() {}

    // Copy constructor
    Item(const Item& other)
        : name(other.name), price(other.price), quantity(other.quantity) {}

    // Copy assignment operator
    Item& operator=(const Item& other) {
        if (this != &other) {
            name = other.name;
            price = other.price;
            quantity = other.quantity;
        }
        return *this;
    }

    // Calculate total cost of an item
    int calculateSum() const {
        return price * quantity;
    }

    // Name, price, and quantity
    std::string getName() const {
        return name;
    }

    int getPrice() const {
        return price;
    }

    int getQuantity() const {
        return quantity;
    }

private:
    std::string name;
    int price;
    int quantity;
};

// Function to calculate the total sum of all items
int TotalSum(const Item groceryList[], int size) {
    int totalSum = 0;
    for (int i = 0; i < size; ++i) {
        totalSum += groceryList[i].calculateSum();
    }
    return totalSum;
}

int main() {
    const int groceryListSize = 4;

    // Create an array GroceryList
    Item groceryList[groceryListSize] = {
        Item("Apple", 10, 7),
```



```cpp
        Item("Apple", 10, 7),
        Item("Banana", 10, 8),
        Item("Broccoli", 60, 12),
        Item("Lettuce", 50, 10)
    };

    // Display details about all relevant attributes of items
    for (int i = 0; i < groceryListSize; ++i) {
        std::cout << "Name: " << groceryList[i].getName()
                  << ", Price: " << groceryList[i].getPrice()
                  << ", Quantity: " << groceryList[i].getQuantity()
                  << ", Total Cost: " << groceryList[i].calculateSum() << std::endl;

    }

    // Calculate the total sum
    int totalSum = TotalSum(groceryList, groceryListSize);
    std::cout << "Total Sum: " << totalSum << std::endl;

    // Remove the Lettuce from the GroceryList
    int newSize = groceryListSize;
    for (int i = 0; i < newSize; ++i) {
        if (groceryList[i].getName() == "Lettuce") {
            for (int j = i; j < newSize - 1; ++j) {
                groceryList[j] = groceryList[j + 1];
            }
            // Decrease the size of the list
            --newSize;
            break;
        }
    }

  // Display new details after removal
    std::cout << "\nAfter removing Lettuce:" << std::endl;
    for (int i = 0; i < newSize; ++i) {
        std::cout << "Name: " << groceryList[i].getName()
                  << ", Price: " << groceryList[i].getPrice()
                  << ", Quantity: " << groceryList[i].getQuantity()
                  << ", Total Cost: " << groceryList[i].calculateSum() << std::endl;
    }

    // Recalculate the total sum
    totalSum = TotalSum(groceryList, newSize);
    std::cout << "\nTotal Sum After Removal: " << totalSum << std::endl;

    return 0;
}
```



```
/tmp/oSo61mL2CQ.o
Name: Apple, Price: 10, Quantity: 7, Total Cost: 70
Name: Banana, Price: 10, Quantity: 8, Total Cost: 80
Name: Broccoli, Price: 60, Quantity: 12, Total Cost: 720
Name: Lettuce, Price: 50, Quantity: 10, Total Cost: 500
Total Sum: 1370

After removing Lettuce:
Name: Apple, Price: 10, Quantity: 7, Total Cost: 70
Name: Banana, Price: 10, Quantity: 8, Total Cost: 80
Name: Broccoli, Price: 60, Quantity: 12, Total Cost: 720

Total Sum After Removal: 870


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

8. Conclusion

Provide the following:
- Summary of lessons learned
- Analysis of the procedure

• Analysis of the supplementary activity • Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement? I've learned about constructors and destructors in C++. When an object is created, the constructor is called and when it is destroyed, the destructor is called. Meanwhile, the copy constructor and assignment operator ensure that objects are properly copied. In the supplementary activity, I explored array manipulation, created an item class with methods for calculating total cost and managing the list, and performed operations such as object removal. I have struggled but thankfully I have carried out this activity. An area for improvement would consist of better error handling and broadening my knowledge of C++.
9. Assessment Rubric