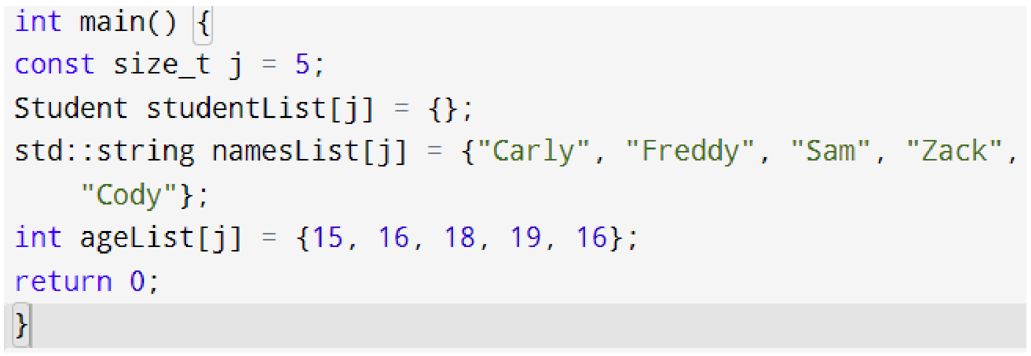
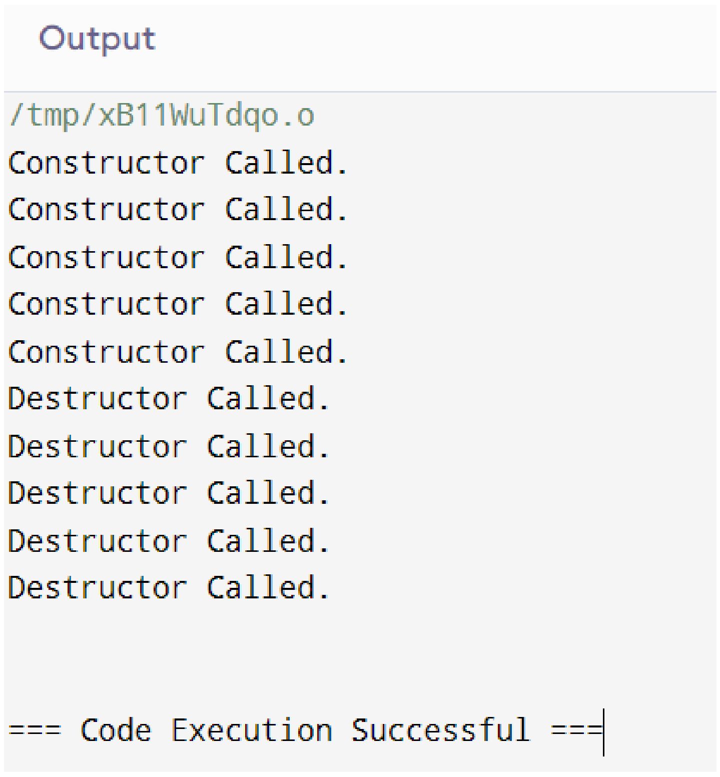
Activity No. 2.1	
Arrays, Pointers and Dynamic Memory Allocation	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 9/11/24
Section: CPE21S4	**Date Submitted:** 9/11/24
Name(s): CALVIN EARL PLANTA	**Instructor:** Dr. Sayo

6. Output

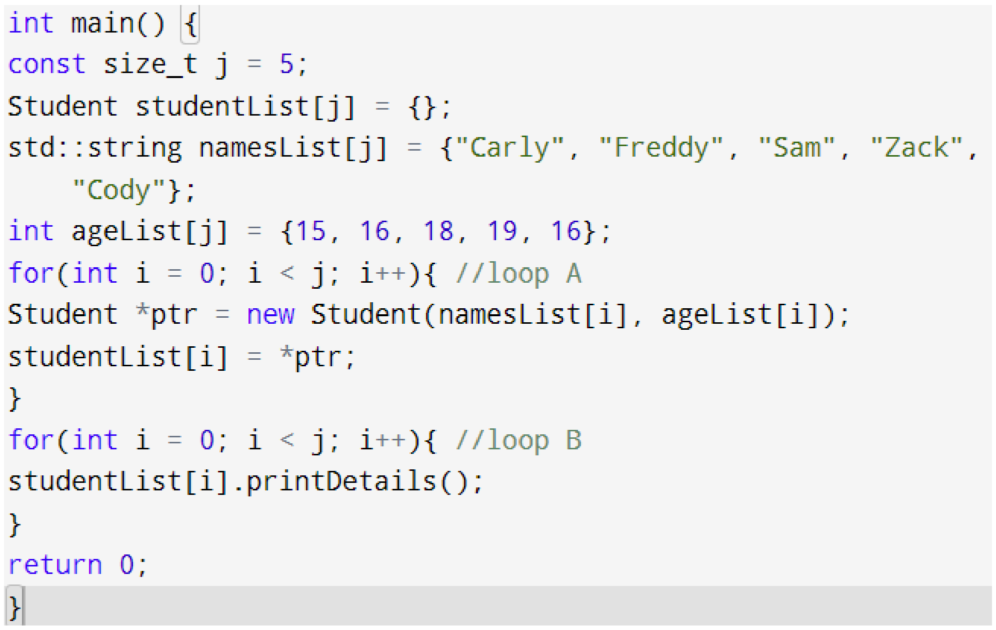
Screenshot	 
Observation	At the end of the program, destructors for “student1”, “student2”, and “student3” will be called when they go out of scope.

```
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;
}
```

Output

```
/tmp/xB11WuTdqo.o
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.


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

Table 2-1. Initial Driver Program

Screenshot	 `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;`
	`}`

Observation	At the end of the program, when the “studentList” array goes out of scope, the destructors for each “Student” object in the array will be called.

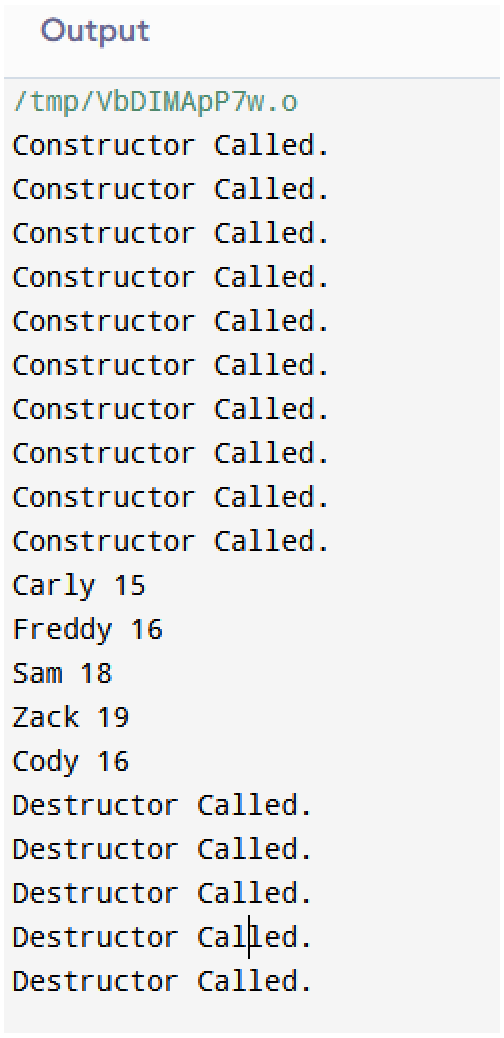
Table 2-2. Modified Driver Program with Student Lists

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

Observation	Inefficient due to unnecessary dynamic memory allocation and subsequent copying. It also leads to memory leaks as the allocated memory is not deallocated.

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

Observation	Efficient in terms of memory usage, as it just accesses and prints the already initialized Student objects.

Output	 Output
	```
	/tmp/VbDIMApP7w.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.
	```

Observation	After the program ends, destructors for the objects in “studentList” will be called automatically (since they are part of the array).

Table 2-3. Final Driver Program

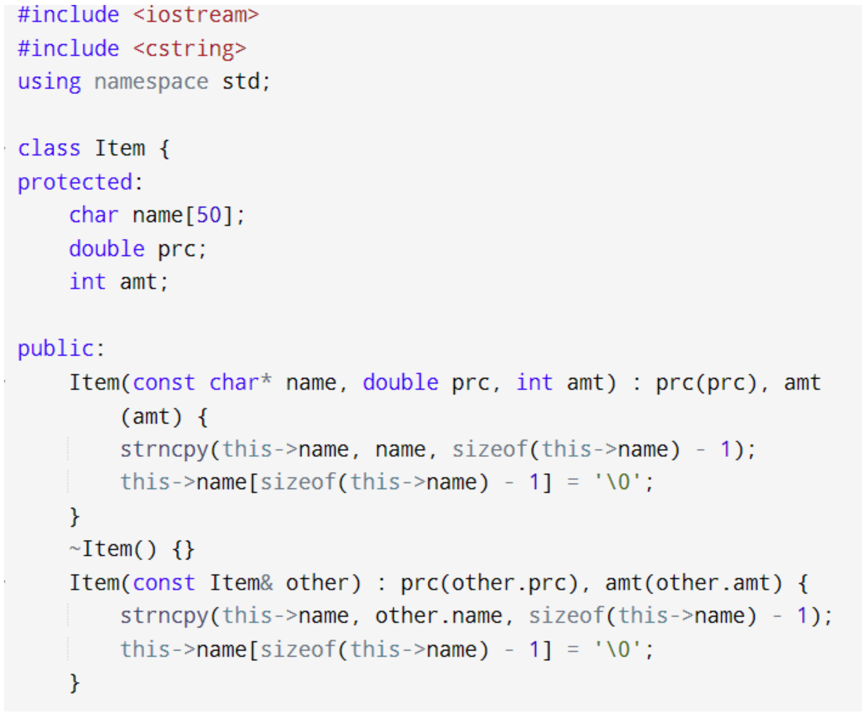
7. Supplementary Activity

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

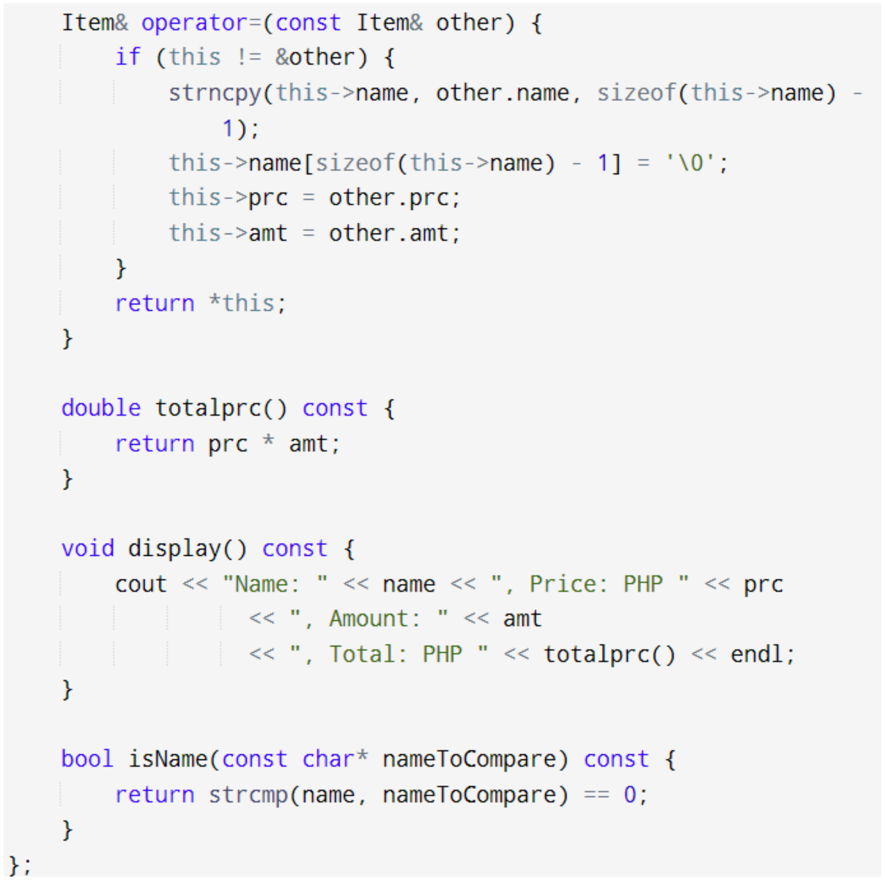
Jenna’s Grocery list		
Apple	PHP 10	x7
Banana	PHP 10	x8
Broccoli	PHP 60	x12
Lettuce	PHP 50	x10

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.

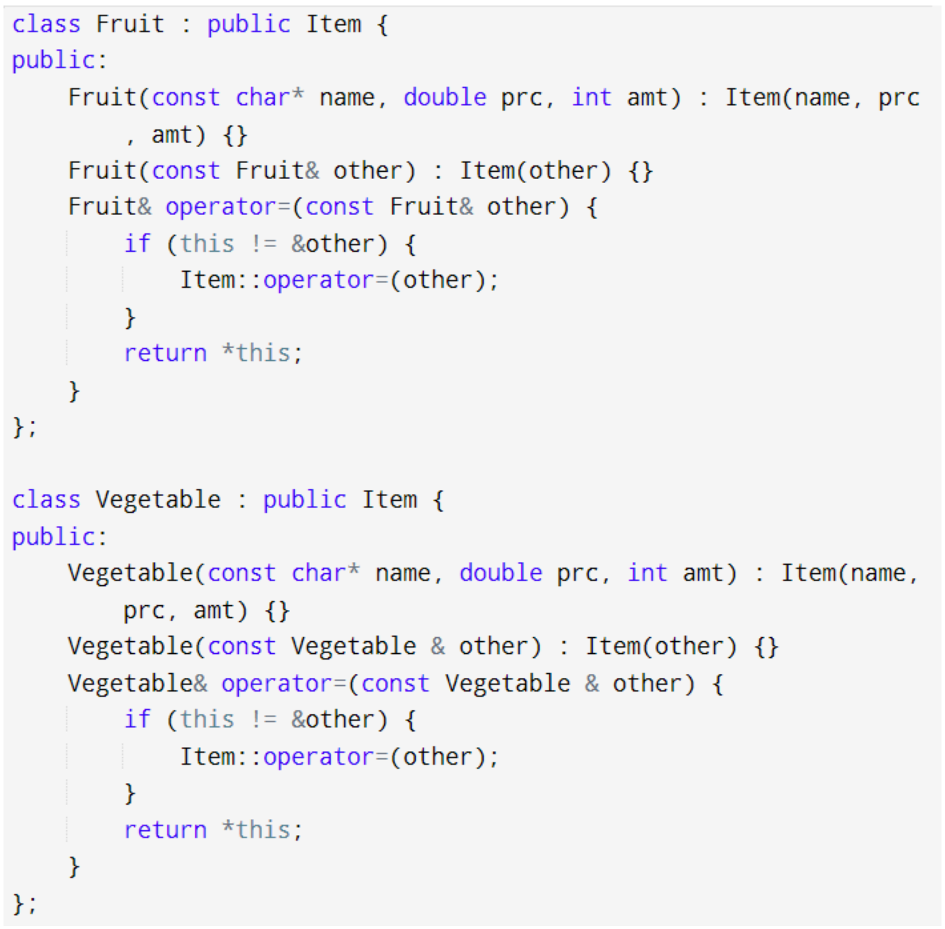
Problem 1: Create a class for the fruit and the vegetable classes. Each class must have a constructor, deconstructor, copy constructor and copy assignment operator. They must also have all relevant attributes (such as name, price and quantity) and functions (such as calculate sum) as presented in the problem description above.



```cpp
#include <iostream>
#include <cstring>
using namespace std;

class Item {
protected:
    char name[50];
    double prc;
    int amt;

public:
    Item(const char* name, double prc, int amt) : prc(prc), amt(amt) {
        strncpy(this->name, name, sizeof(this->name) - 1);
        this->name[sizeof(this->name) - 1] = '\0';
    }
    ~Item() {}
    Item(const Item& other) : prc(other.prc), amt(other.amt) {
        strncpy(this->name, other.name, sizeof(this->name) - 1);
        this->name[sizeof(this->name) - 1] = '\0';
    }
```

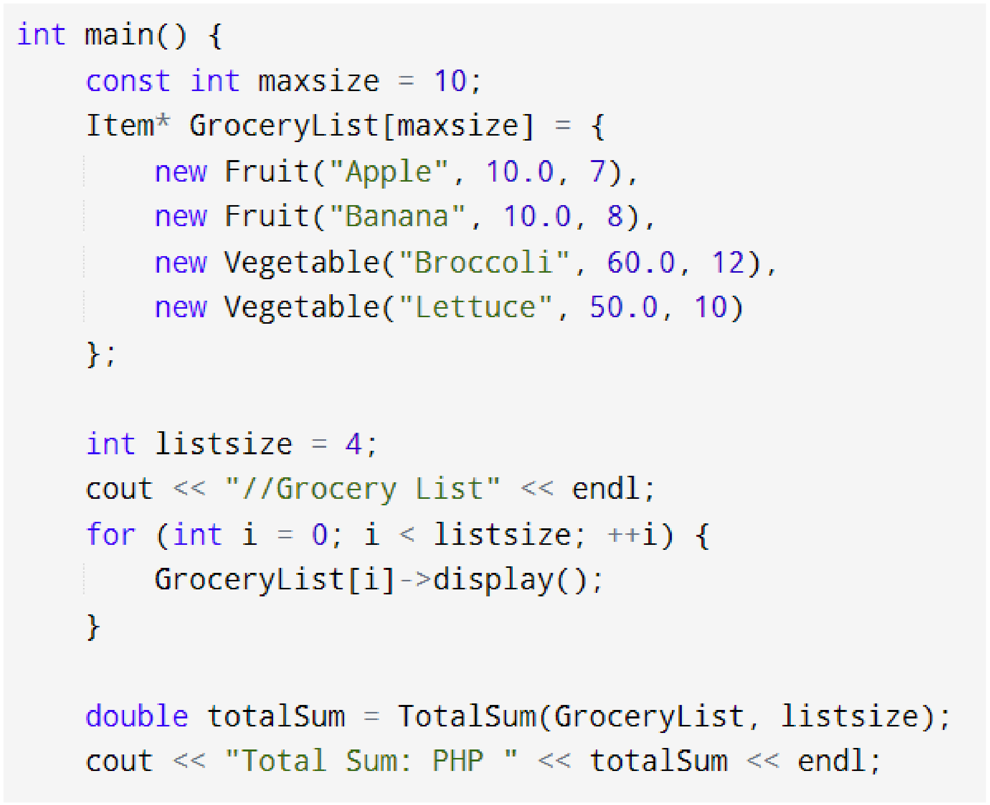


```cpp
    Item& operator=(const Item& other) {
        if (this != &other) {
            strncpy(this->name, other.name, sizeof(this->name) - 1);
            this->name[sizeof(this->name) - 1] = '\0';
            this->prc = other.prc;
            this->amt = other.amt;
        }
        return *this;
    }

    double totalprc() const {
        return prc * amt;
    }

    void display() const {
        cout << "Name: " << name << ", Price: PHP " << prc
                    << ", Amount: " << amt
                    << ", Total: PHP " << totalprc() << endl;
    }

    bool isName(const char* nameToCompare) const {
        return strcmp(name, nameToCompare) == 0;
    }
};
```



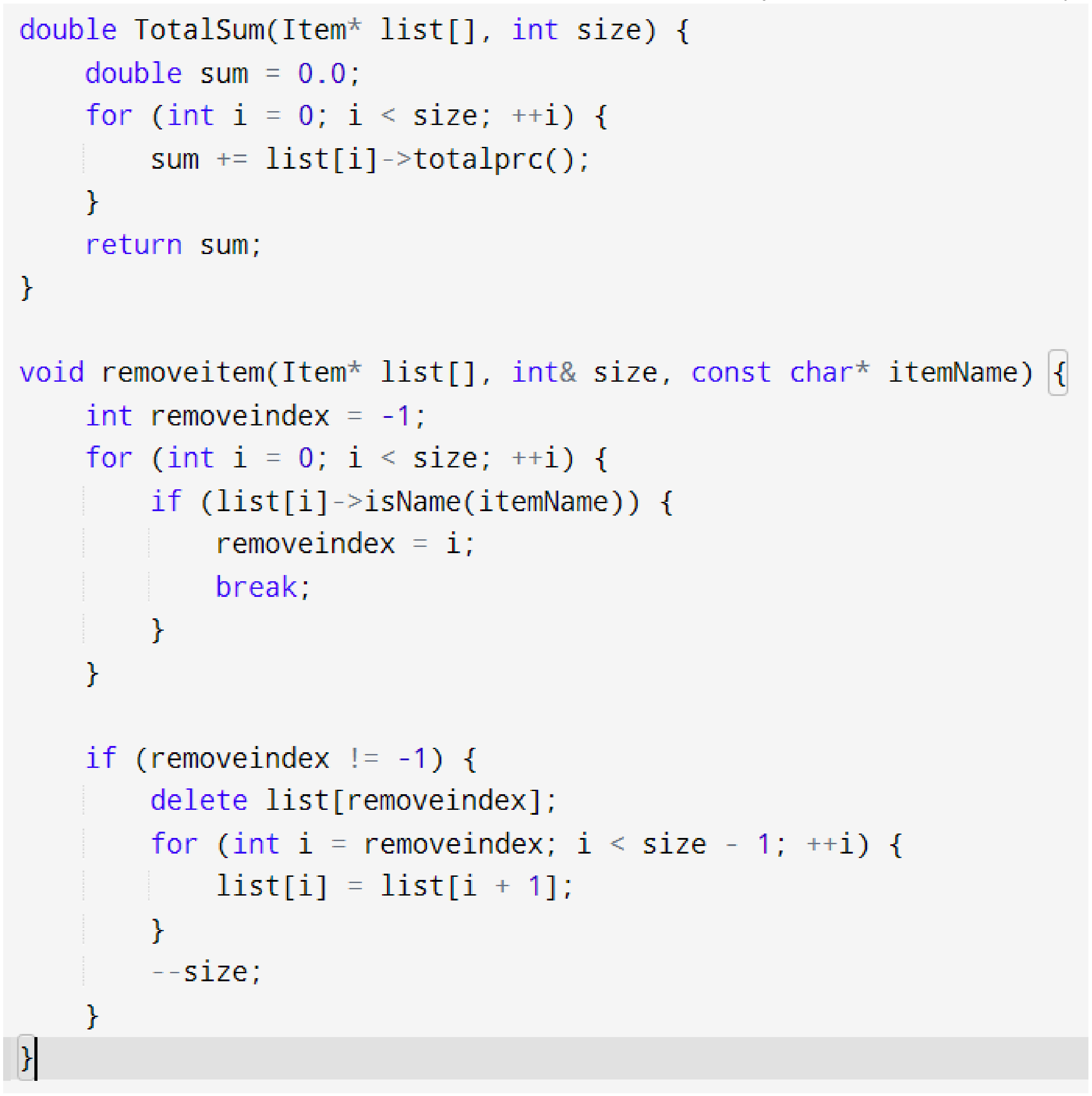
```
class Fruit : public Item {
public:
    Fruit(const char* name, double prc, int amt) : Item(name, prc
        , amt) {}
    Fruit(const Fruit& other) : Item(other) {}
    Fruit& operator=(const Fruit& other) {
        if (this != &other) {
            Item::operator=(other);
        }
        return *this;
    }
};

class Vegetable : public Item {
public:
    Vegetable(const char* name, double prc, int amt) : Item(name,
        prc, amt) {}
    Vegetable(const Vegetable & other) : Item(other) {}
    Vegetable& operator=(const Vegetable & other) {
        if (this != &other) {
            Item::operator=(other);
        }
        return *this;
    }
};
```

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.



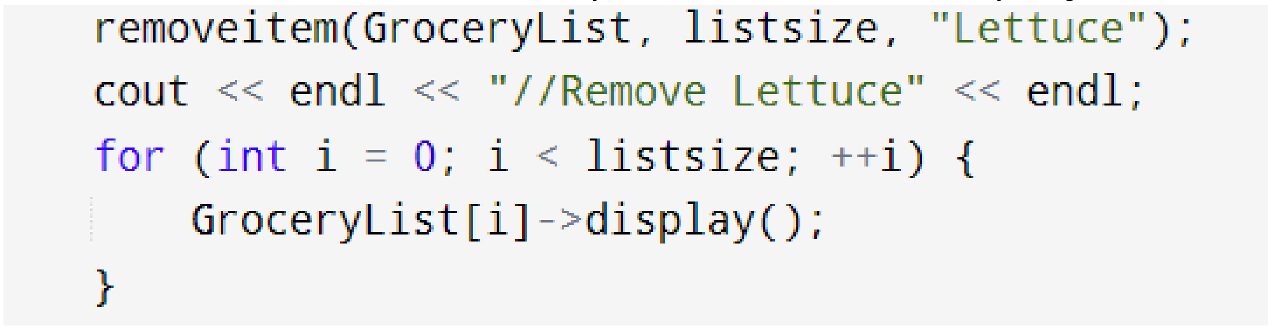
```
int main() {
    const int maxsize = 10;
    Item* GroceryList[maxsize] = {
        new Fruit("Apple", 10.0, 7),
        new Fruit("Banana", 10.0, 8),
        new Vegetable("Broccoli", 60.0, 12),
        new Vegetable("Lettuce", 50.0, 10)
    };

    int listsize = 4;
    cout << "//Grocery List" << endl;
    for (int i = 0; i < listsize; ++i) {
        GroceryList[i]->display();
    }

    double totalSum = TotalSum(GroceryList, listsize);
    cout << "Total Sum: PHP " << totalSum << endl;
```

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



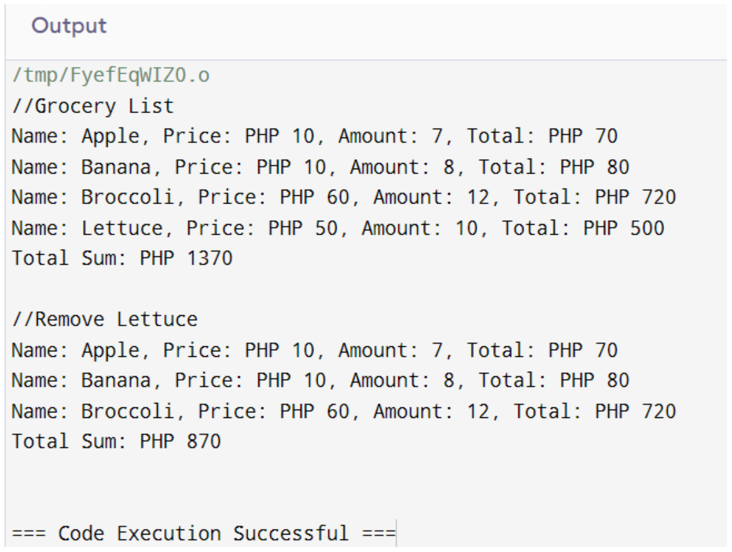
```cpp
double TotalSum(Item* list[], int size) {
    double sum = 0.0;
    for (int i = 0; i < size; ++i) {
        sum += list[i]->totalprc();
    }
    return sum;
}

void removeitem(Item* list[], int& size, const char* itemName) {
    int removeindex = -1;
    for (int i = 0; i < size; ++i) {
        if (list[i]->isName(itemName)) {
            removeindex = i;
            break;
        }
    }

    if (removeindex != -1) {
        delete list[removeindex];
        for (int i = removeindex; i < size - 1; ++i) {
            list[i] = list[i + 1];
        }
        --size;
    }
}
```

Problem 4: Delete the Lettuce from Jenna's GroceryList list and deallocate the memory assigned



```cpp
    removeitem(GroceryList, listsize, "Lettuce");
    cout << endl << "//Remove Lettuce" << endl;
    for (int i = 0; i < listsize; ++i) {
        GroceryList[i]->display();
    }
```

Output


Output

```
/tmp/FyefEqWIZ0.o
//Grocery List
Name: Apple, Price: PHP 10, Amount: 7, Total: PHP 70
Name: Banana, Price: PHP 10, Amount: 8, Total: PHP 80
Name: Broccoli, Price: PHP 60, Amount: 12, Total: PHP 720
Name: Lettuce, Price: PHP 50, Amount: 10, Total: PHP 500
Total Sum: PHP 1370

//Remove Lettuce
Name: Apple, Price: PHP 10, Amount: 7, Total: PHP 70
Name: Banana, Price: PHP 10, Amount: 8, Total: PHP 80
Name: Broccoli, Price: PHP 60, Amount: 12, Total: PHP 720
Total Sum: PHP 870


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

8. Conclusion

By implementing static and dynamic variables in this project, I have improved my understanding of how to use pointers and arrays.
dynamic memory in C++. I created a `Student} class and saw how constructors, destructors, and other
Important processes operate. I was able to create a variety of pupils and provide each student dynamic memory allocation. I also discovered that, in order to prevent problems, memory must be cleaned out after usage.

9. Assessment Rubric