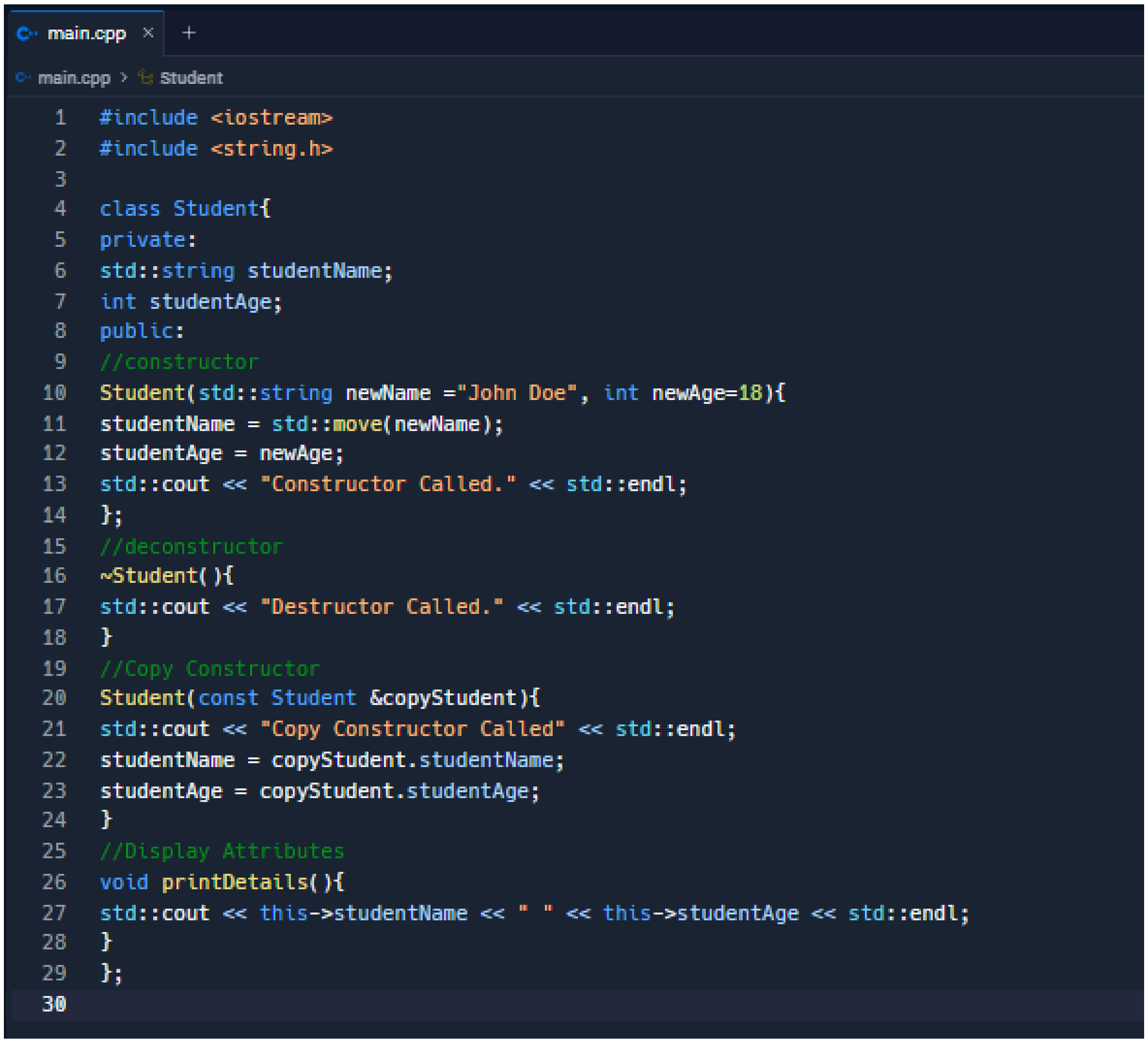
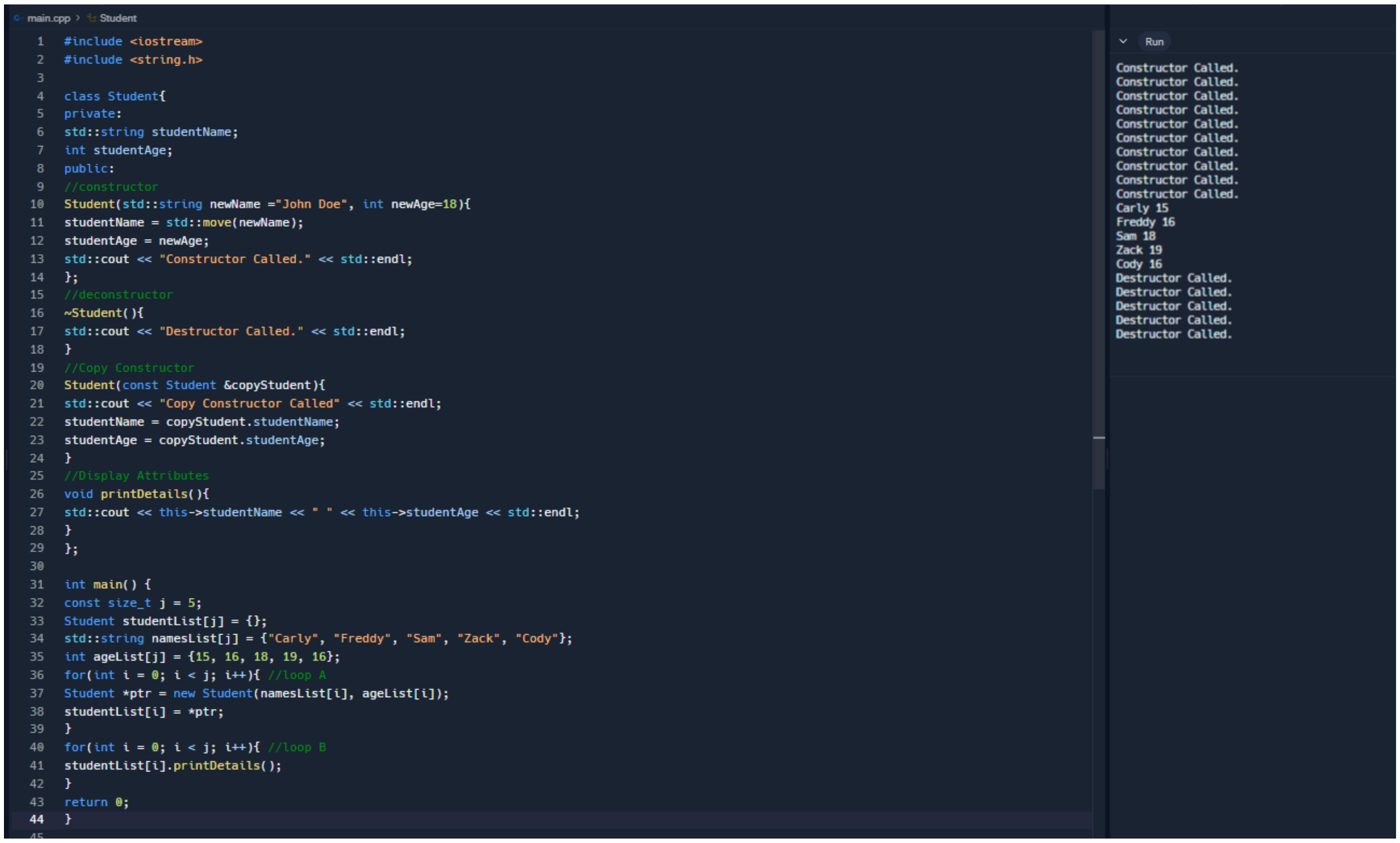
Activity No. 2	
Arrays, Pointers and Dynamic Memory Allocation	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: SEPTEMBER 11, 2024**
Section: CPE21S1	**Date Submitted: SEPTEMBER 11, 2024**
Name(s): GASPAR, AARON ROWEN O.	**Instructor: MA'AM MARIA RIZETTE SAYO**

6. 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;
}
};

```

This is the initial driver program and will not output anything.

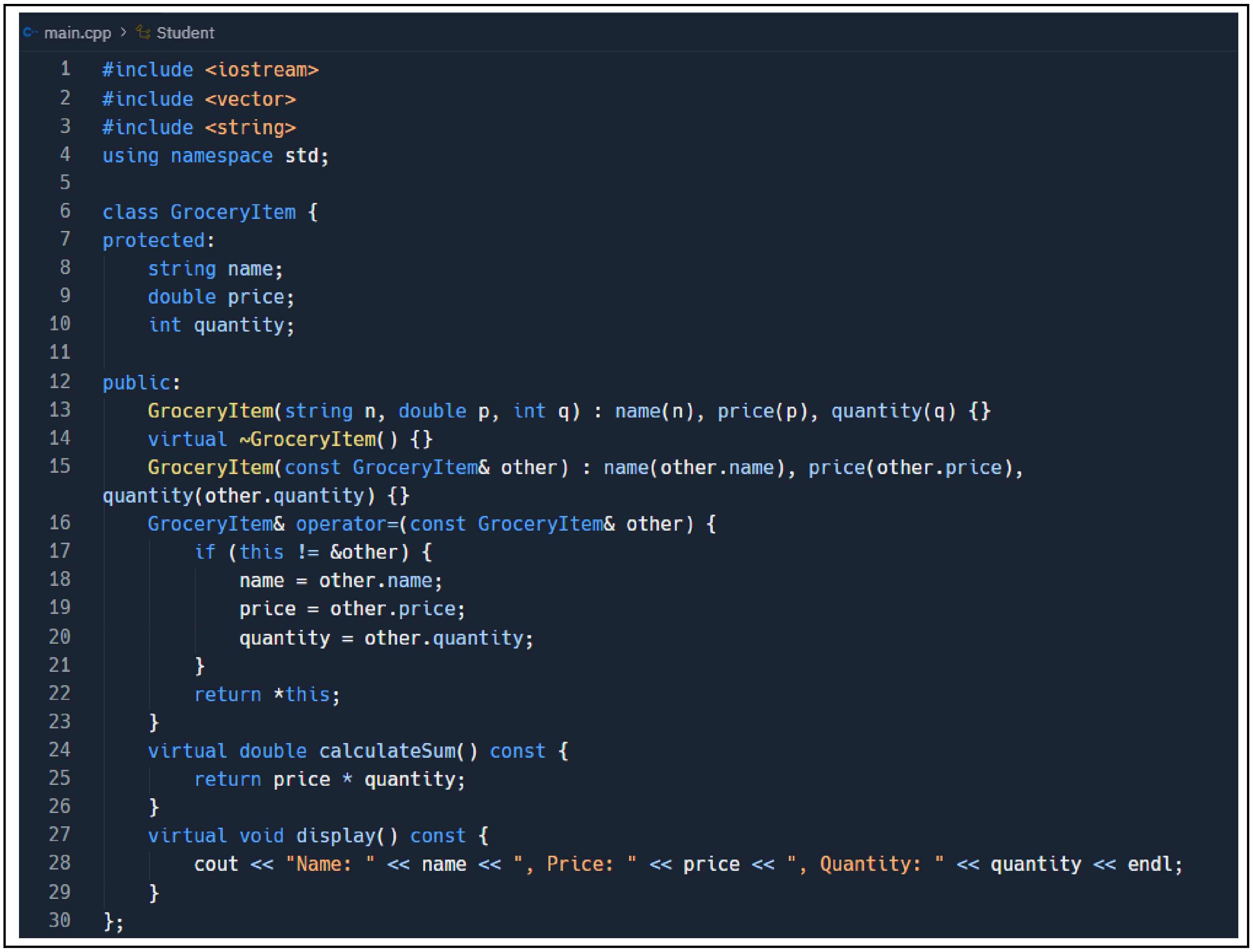


```cpp
#include <iostream>
#include <string.h>

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;
}
```

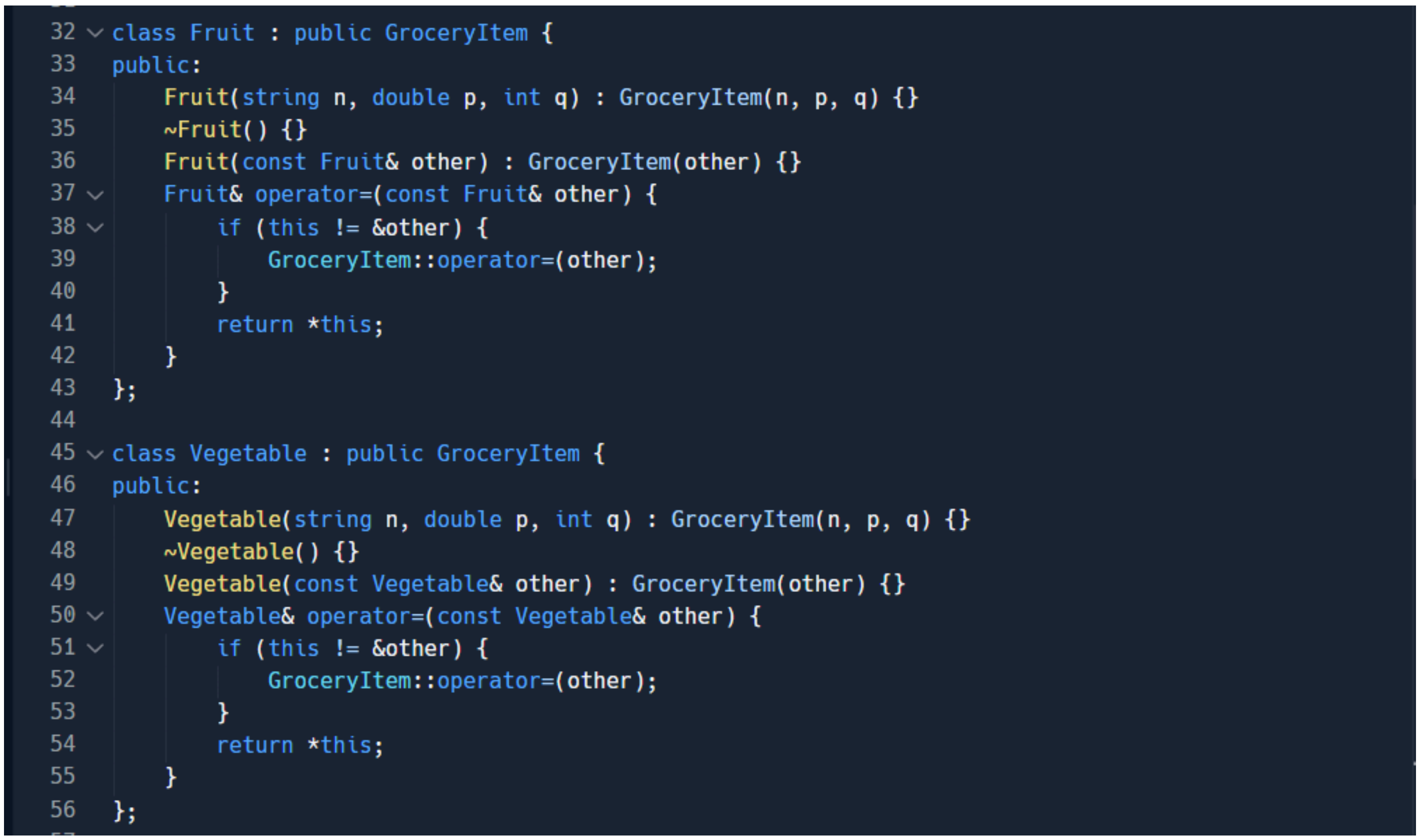
```
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Carly 15
Freddy 16
Sam 18
Zack 19
Cody 16
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

Upon adding the code, it outputs all the syntax with cout function in it.

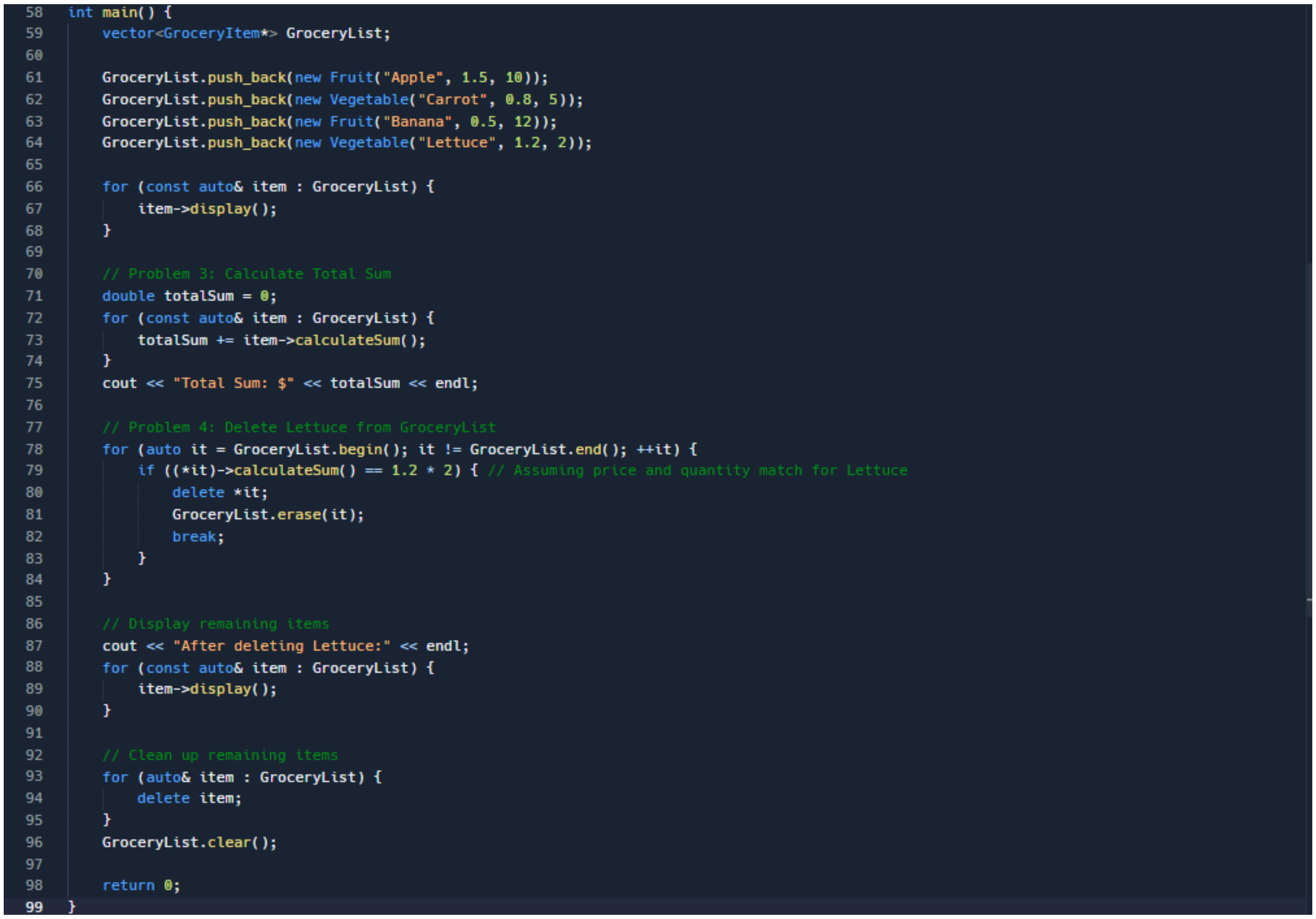
7. Supplementary Activity


main.cpp > Student

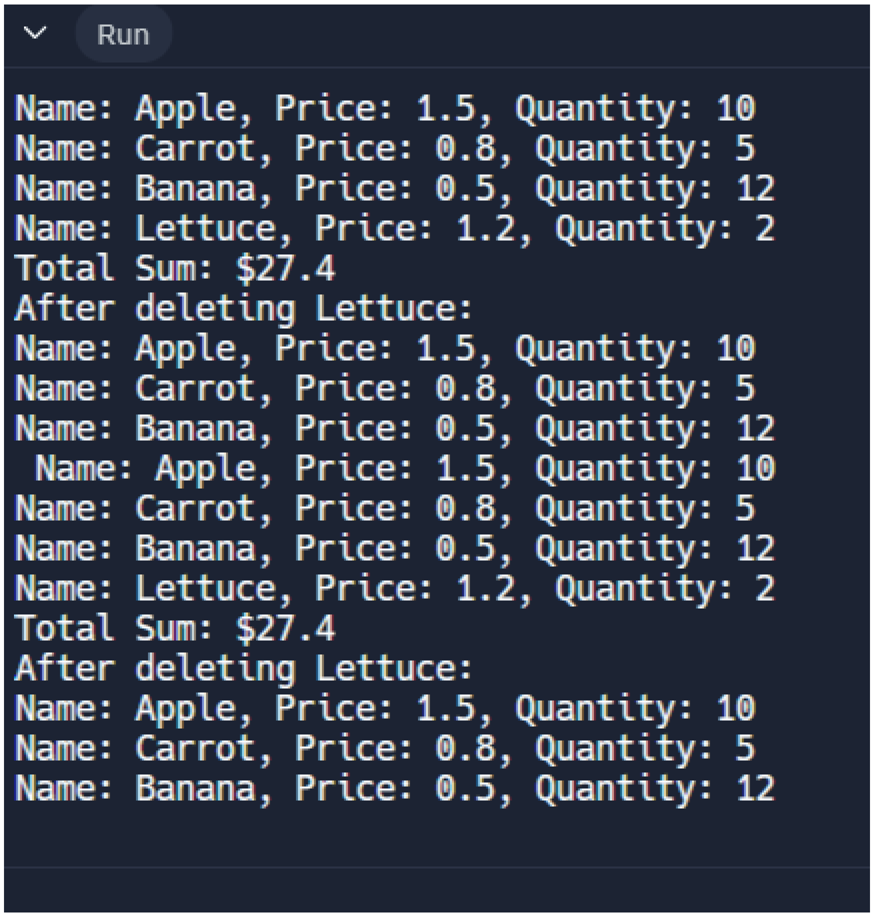
```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class GroceryItem {
protected:
    string name;
    double price;
    int quantity;

public:
    GroceryItem(string n, double p, int q) : name(n), price(p), quantity(q) {}
    virtual ~GroceryItem() {}
    GroceryItem(const GroceryItem& other) : name(other.name), price(other.price),
quantity(other.quantity) {}
    GroceryItem& operator=(const GroceryItem& other) {
        if (this != &other) {
            name = other.name;
            price = other.price;
            quantity = other.quantity;
        }
        return *this;
    }
    virtual double calculateSum() const {
        return price * quantity;
    }
    virtual void display() const {
        cout << "Name: " << name << ", Price: " << price << ", Quantity: " << quantity << endl;
    }
};
```



```cpp
class Fruit : public GroceryItem {
public:
    Fruit(string n, double p, int q) : GroceryItem(n, p, q) {}
    ~Fruit() {}
    Fruit(const Fruit& other) : GroceryItem(other) {}
    Fruit& operator=(const Fruit& other) {
        if (this != &other) {
            GroceryItem::operator=(other);
        }
        return *this;
    }
};

class Vegetable : public GroceryItem {
public:
    Vegetable(string n, double p, int q) : GroceryItem(n, p, q) {}
    ~Vegetable() {}
    Vegetable(const Vegetable& other) : GroceryItem(other) {}
    Vegetable& operator=(const Vegetable& other) {
        if (this != &other) {
            GroceryItem::operator=(other);
        }
        return *this;
    }
};
```



```cpp
int main() {
    vector<GroceryItem*> GroceryList;

    GroceryList.push_back(new Fruit("Apple", 1.5, 10));
    GroceryList.push_back(new Vegetable("Carrot", 0.8, 5));
    GroceryList.push_back(new Fruit("Banana", 0.5, 12));
    GroceryList.push_back(new Vegetable("Lettuce", 1.2, 2));

    for (const auto& item : GroceryList) {
        item->display();
    }

    // Problem 3: Calculate Total Sum
    double totalSum = 0;
    for (const auto& item : GroceryList) {
        totalSum += item->calculateSum();
    }
    cout << "Total Sum: $" << totalSum << endl;

    // Problem 4: Delete Lettuce from GroceryList
    for (auto it = GroceryList.begin(); it != GroceryList.end(); ++it) {
        if ((*it)->calculateSum() == 1.2 * 2) { // Assuming price and quantity match for Lettuce
            delete *it;
            GroceryList.erase(it);
            break;
        }
    }

    // Display remaining items
    cout << "After deleting Lettuce:" << endl;
    for (const auto& item : GroceryList) {
        item->display();
    }

    // Clean up remaining items
    for (auto& item : GroceryList) {
        delete item;
    }
    GroceryList.clear();

    return 0;
}
```



```
Name: Apple, Price: 1.5, Quantity: 10
Name: Carrot, Price: 0.8, Quantity: 5
Name: Banana, Price: 0.5, Quantity: 12
Name: Lettuce, Price: 1.2, Quantity: 2
Total Sum: $27.4
After deleting Lettuce:
Name: Apple, Price: 1.5, Quantity: 10
Name: Carrot, Price: 0.8, Quantity: 5
Name: Banana, Price: 0.5, Quantity: 12
 Name: Apple, Price: 1.5, Quantity: 10
Name: Carrot, Price: 0.8, Quantity: 5
Name: Banana, Price: 0.5, Quantity: 12
Name: Lettuce, Price: 1.2, Quantity: 2
Total Sum: $27.4
After deleting Lettuce:
Name: Apple, Price: 1.5, Quantity: 10
Name: Carrot, Price: 0.8, Quantity: 5
Name: Banana, Price: 0.5, Quantity: 12
```

8. Conclusion

In this lesson, we delved into object-oriented programming in C++ by creating classes for fruits and vegetables. By mastering these concepts, you have gained a solid understanding of class design, inheritance, and dynamic memory management in C++. These skills are essential for building robust and efficient programs. Keep practicing and experimenting with these techniques to further enhance your programming abilities.

9. Assessment Rubric