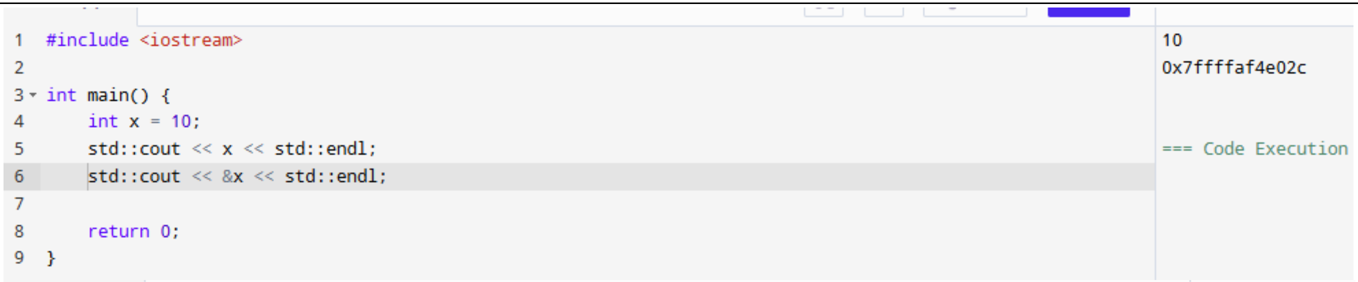
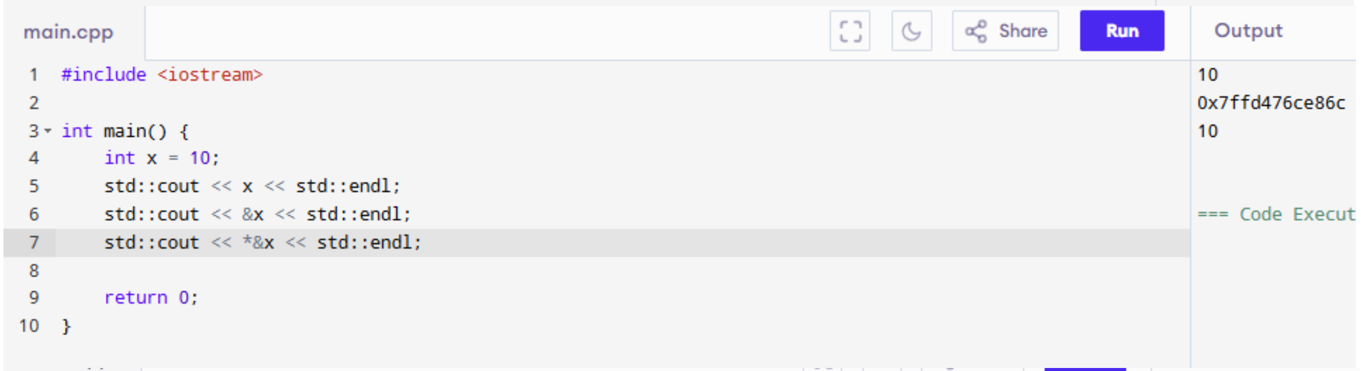
Activity No. 2.1	
Arrays, Pointers and Dynamic Memory Allocation	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:8/7/25**
Section:CpE12S4	**Date Submitted:8/7/25**
Name(s):Crishen Luper S. Pulgado	**Instructor: Jimlord Quejado**
6. Output	



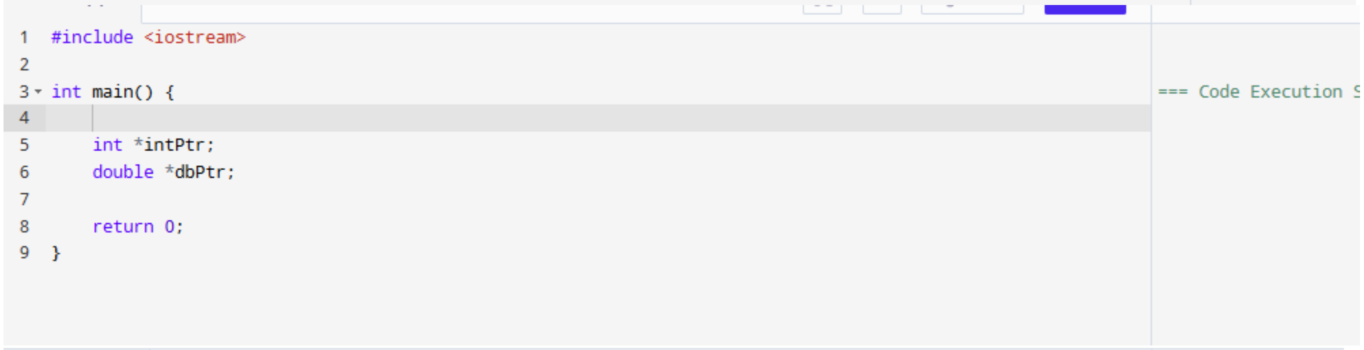
```cpp
#include <iostream>

int main() {
    int x = 10;
    std::cout << x << std::endl;
    std::cout << &x << std::endl;

    return 0;
}
```

```
10
0x7ffffaf4e02c


=== Code Execution
```



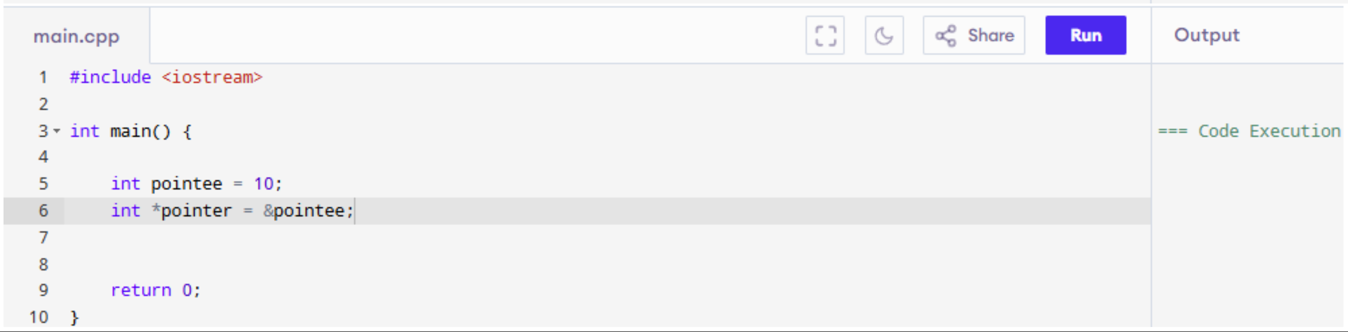
```cpp
#include <iostream>

int main() {
    int x = 10;
    std::cout << x << std::endl;
    std::cout << &x << std::endl;
    std::cout << *&x << std::endl;

    return 0;
}
```

```
10
0x7ffd476ce86c
10


=== Code Execut
```



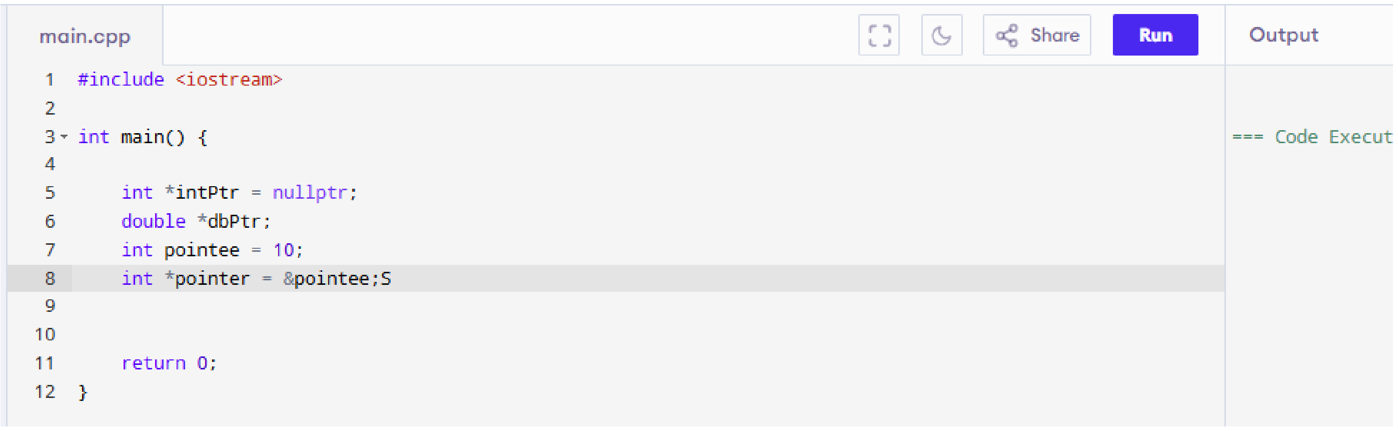
```cpp
#include <iostream>

int main() {

    int *intPtr;
    double *dbPtr;

    return 0;
}
```

```
=== Code Execution S
```



```cpp
#include <iostream>

int main() {

    int pointee = 10;
    int *pointer = &pointee;


    return 0;
}
```

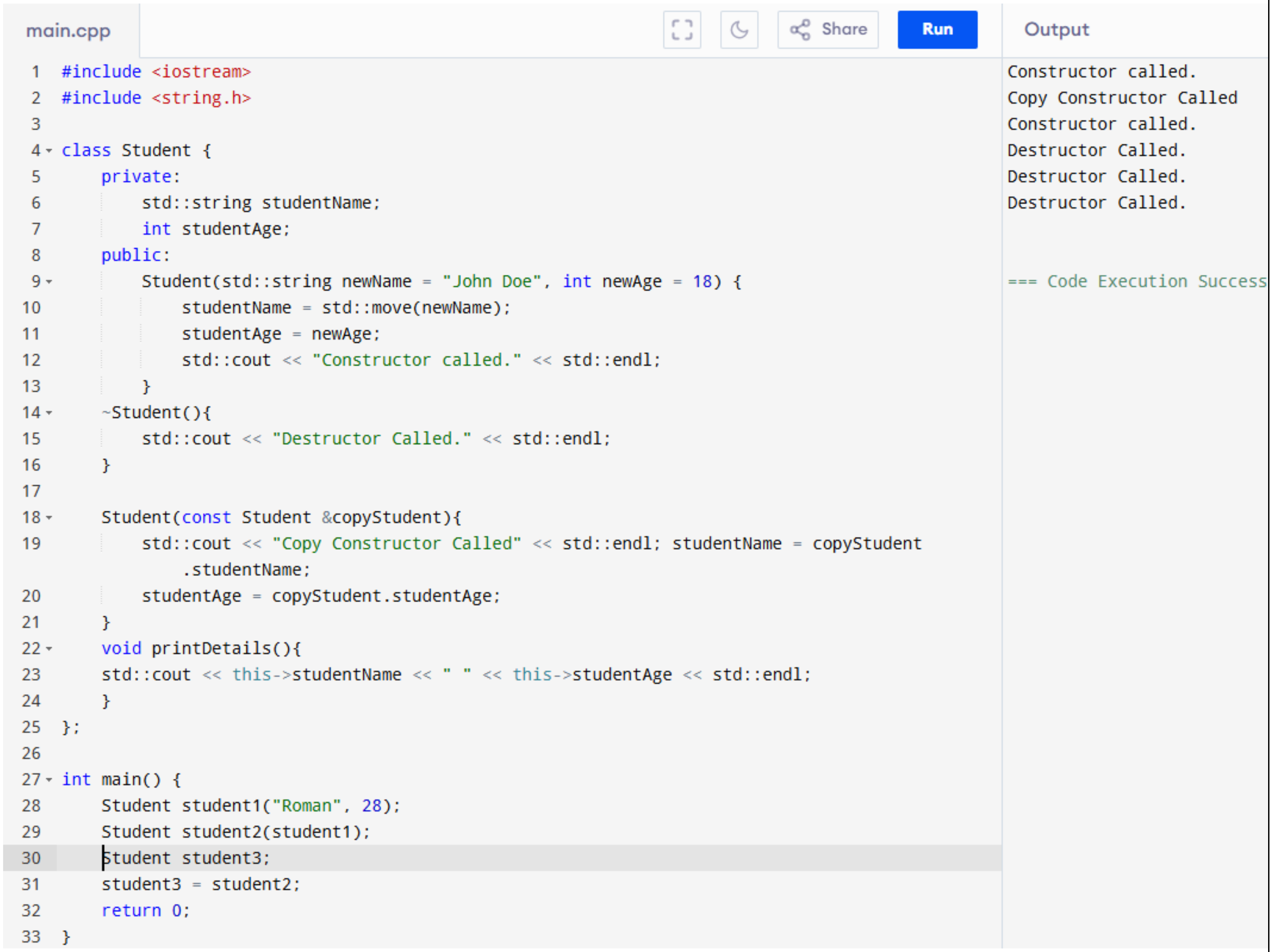
```
=== Code Execution
```



```cpp
#include <iostream>

int main() {

    int *intPtr = nullptr;
    double *dbPtr;
    int pointee = 10;
    int *pointer = &pointee;S


    return 0;
}
```

```
=== Code Execut
```

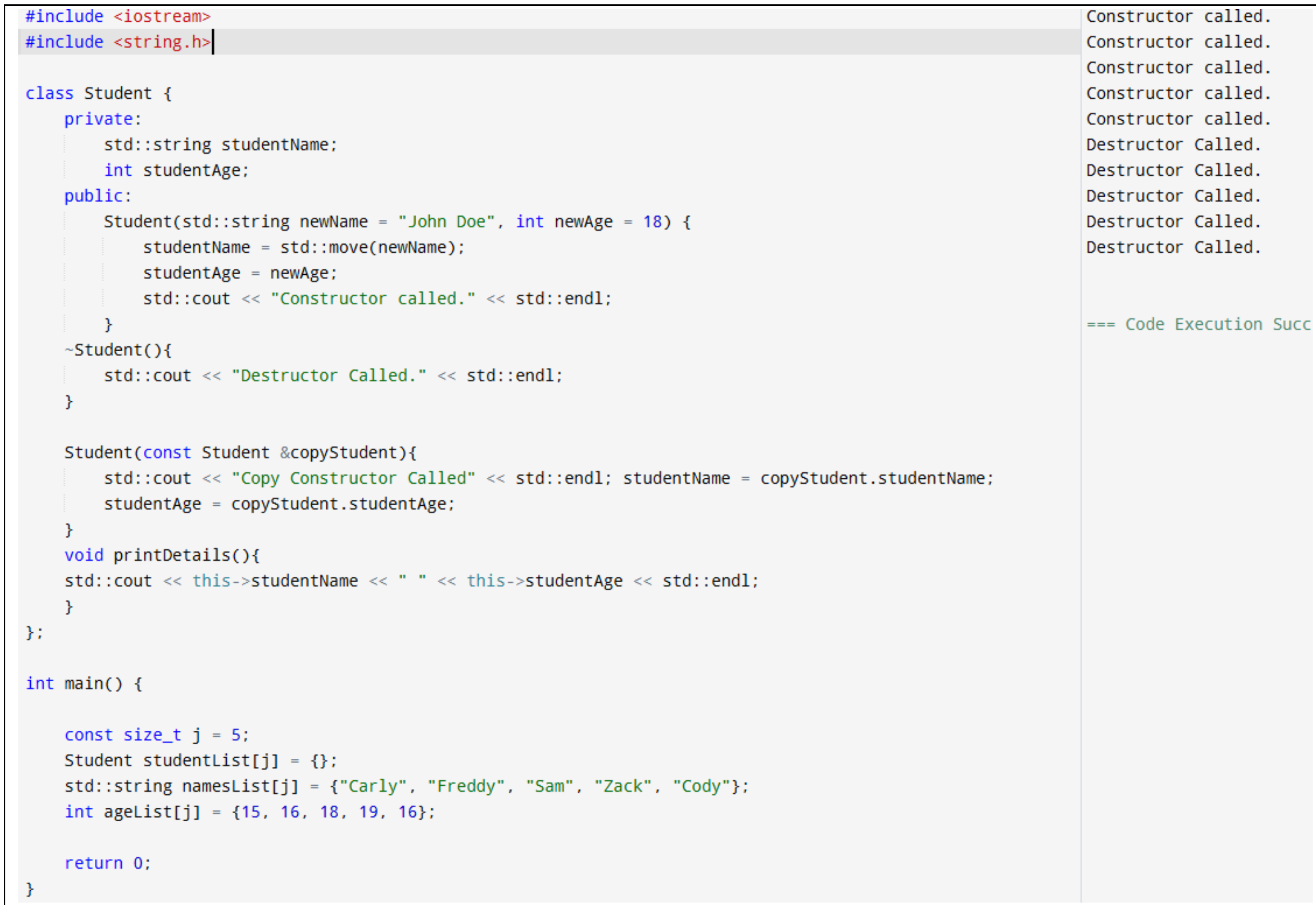
Procedure:



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

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

    Student(const Student &copyStudent){
        std::cout << "Copy Constructor Called" << std::endl; studentName = copyStudent
            .studentName;
        studentAge = copyStudent.studentAge;
    }
    void printDetails(){
    std::cout << this->studentName << " " << this->studentAge << std::endl;
    }
};

int main() {
    Student student1("Roman", 28);
    Student student2(student1);
    Student student3;
    student3 = student2;
    return 0;
}
```

```
Constructor called.
Copy Constructor Called
Constructor called.
Destructor Called.
Destructor Called.
Destructor Called.


=== Code Execution Success
```

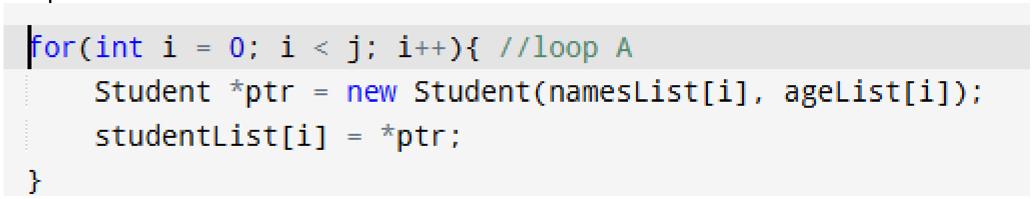
Observation table 2-1:
Based on the program, we have allocated the memory with creating an object. The program creates a constructor so that it will call to the driver function and will print the constructor called. The next thing that was called in the main function is the copy constructor, which it copies the existing constructor which is the student 1. In the class also, it is also initialized to call the destructor so that it will delete the memory after running the program.



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

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

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

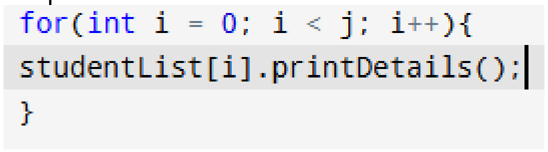
```
Constructor called.
Constructor called.
Constructor called.
Constructor called.
Constructor called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.


=== Code Execution Succ
```

Observation table 2-2:
Here in this program, the driver function calls the 5 students in the list which is Carly, Freddy, etc. The constructor was called 5 times since the array contains 5 elements. As well as the destructor, to delete the memory that was exist during running the code.
Loop A:



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

Observation:
First it creates a new student on the heap using the new to allocate the memory during run time. This will result in calling the constructor. Then, it copies the objected pointed to the ptr into student list. Hence, it will result in using the default copy assignment operator. However, delete was not called so the memory in the heap of the student has not destroyed.
Loop B:



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

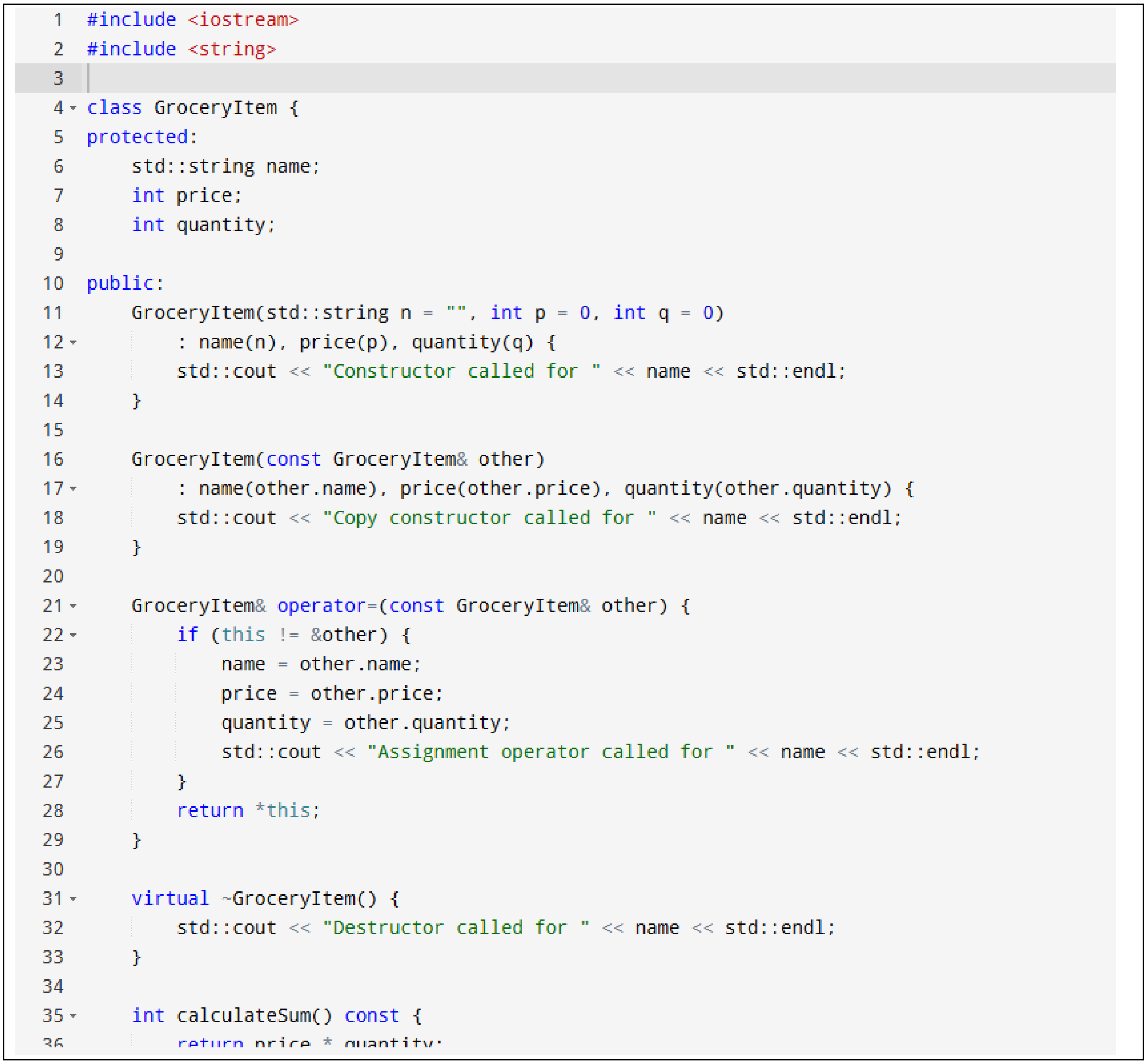
Observation:
The loop calls the printDetails for each student in the studentlist. This resulted in printing Carl 15, Freddy 16, Sam 18, etc.

```
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:
We can observe in the result that the constructor was printed 10 times. The first 5 constructors is the student list from 0 to 4 and the rest was the new students, Carly 15 and the other students, that was allocated to the heap.
Modification:

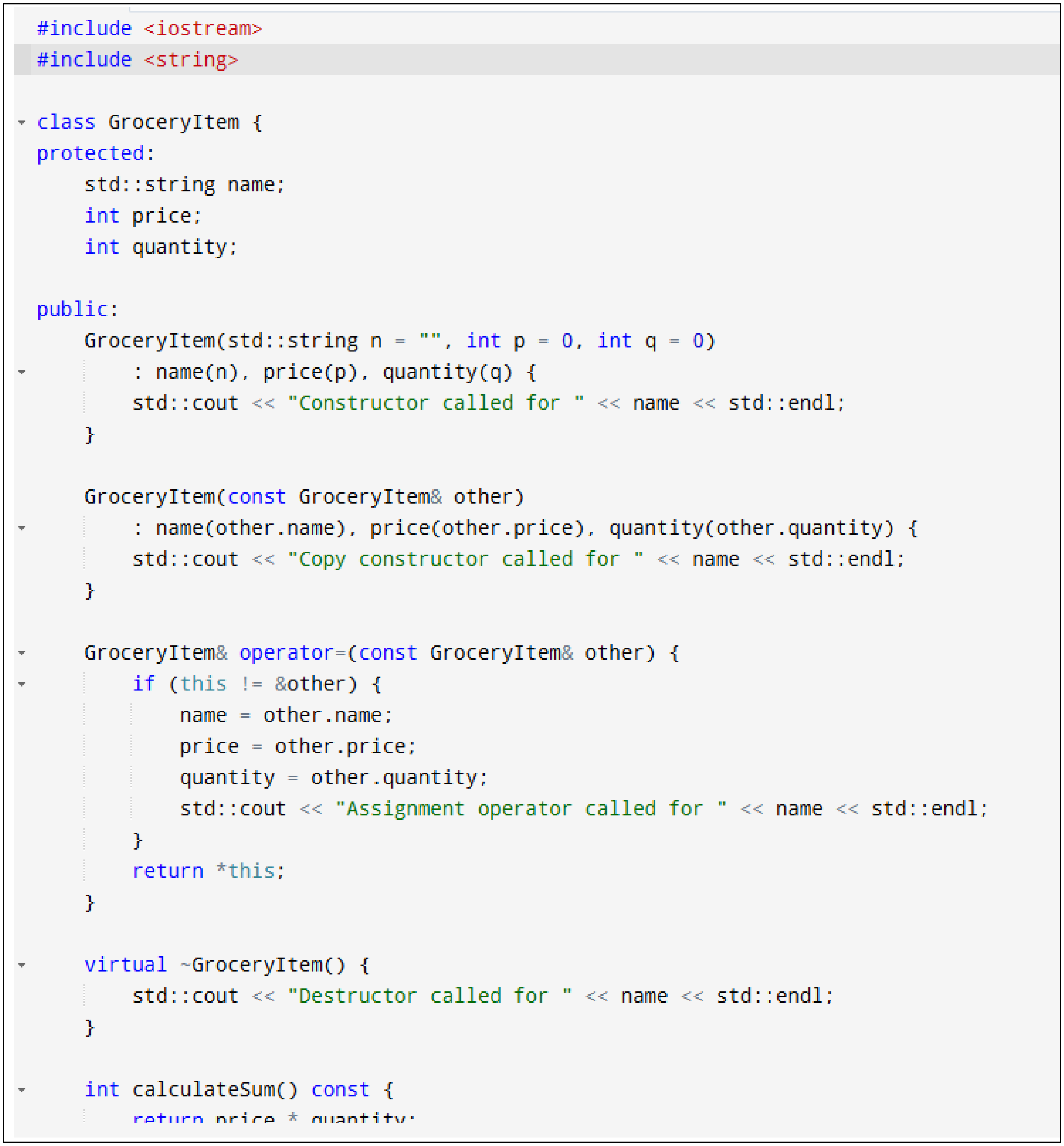
```
for(int i = 0; i < j; i++){
    Student *ptr = new Student(namesList[i], ageList[i]);
    studentList[i] = *ptr;
    delete ptr;
}
```

Observation:
Since the new memory allocation was not deleted in the original, I added delete ptr so that it will avoid memory leak.
ILO C:

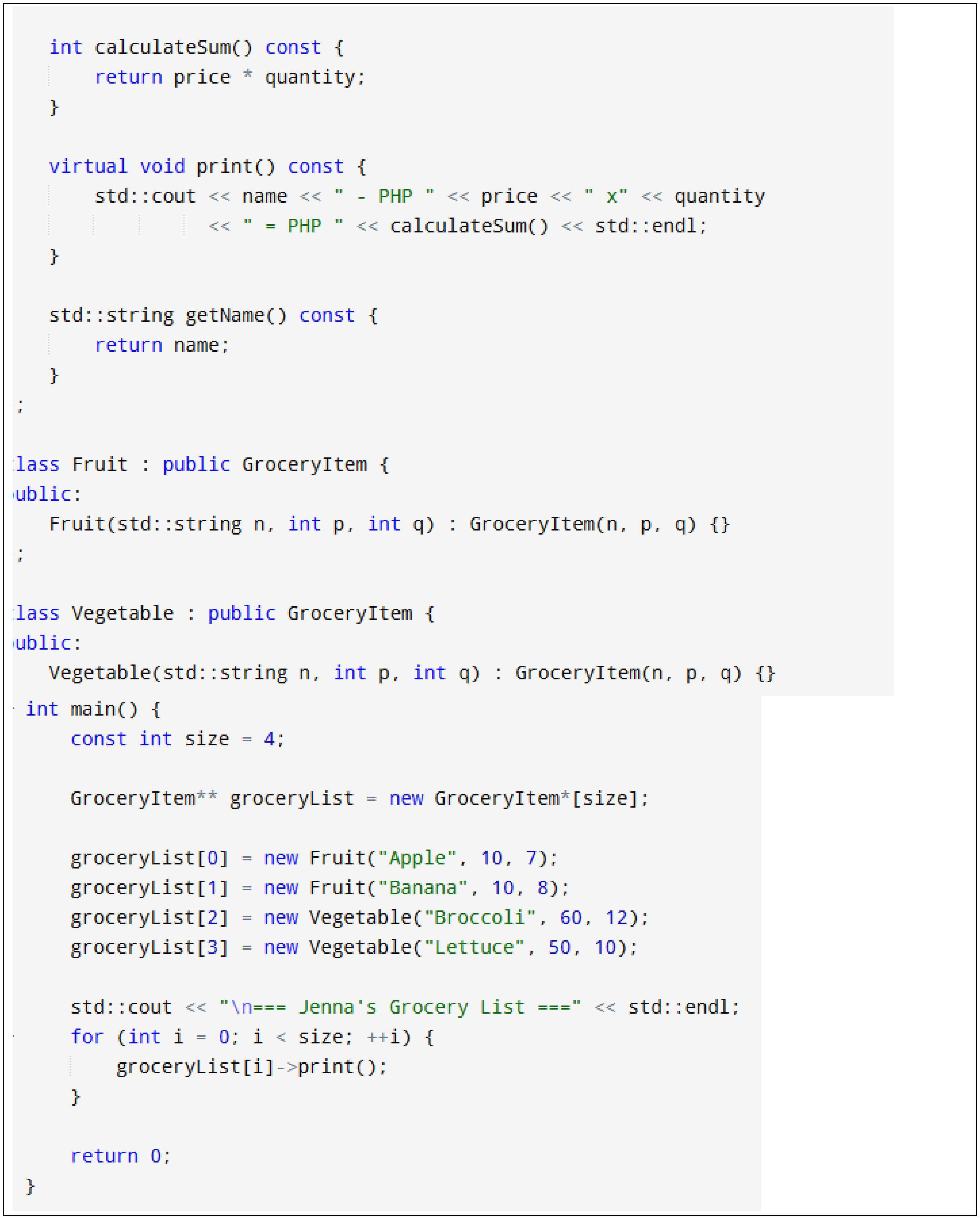


```cpp
#include <iostream>
#include <string>

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

public:
    GroceryItem(std::string n = "", int p = 0, int q = 0)
        : name(n), price(p), quantity(q) {
        std::cout << "Constructor called for " << name << std::endl;
    }

    GroceryItem(const GroceryItem& other)
        : name(other.name), price(other.price), quantity(other.quantity) {
        std::cout << "Copy constructor called for " << name << std::endl;
    }

    GroceryItem& operator=(const GroceryItem& other) {
        if (this != &other) {
            name = other.name;
            price = other.price;
            quantity = other.quantity;
            std::cout << "Assignment operator called for " << name << std::endl;
        }
        return *this;
    }

    virtual ~GroceryItem() {
        std::cout << "Destructor called for " << name << std::endl;
    }

    int calculateSum() const {
        return price * quantity;
    int calculateSum() const {
        return price * quantity;
    }

    virtual void print() const {
        std::cout << name << " - PHP " << price << " x" << quantity
                  << " = PHP " << calculateSum() << std::endl;
    }

    std::string getName() const {
        return name;
    }
};

class Fruit : public GroceryItem {
public:
    Fruit(std::string n, int p, int q) : GroceryItem(n, p, q) {}
};

class Vegetable : public GroceryItem {
public:
    Vegetable(std::string n, int p, int q) : GroceryItem(n, p, q) {}
};
```

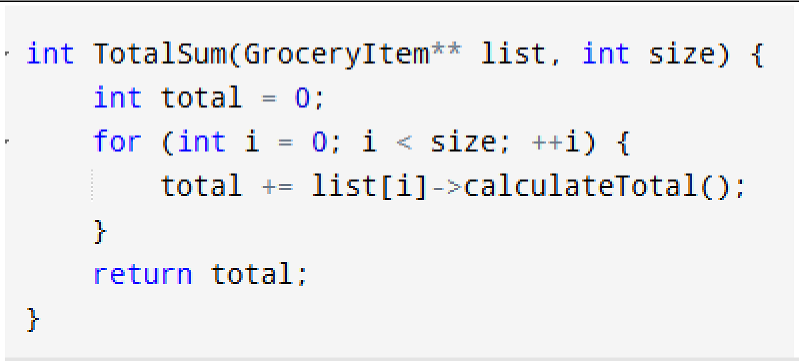
7. Supplementary Activity



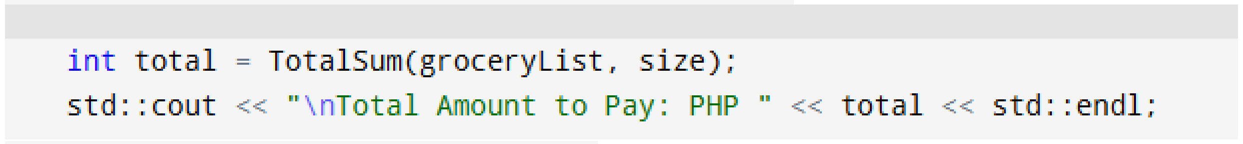
```cpp
#include <iostream>
#include <string>

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

public:
    GroceryItem(std::string n = "", int p = 0, int q = 0)
        : name(n), price(p), quantity(q) {
        std::cout << "Constructor called for " << name << std::endl;
    }

    GroceryItem(const GroceryItem& other)
        : name(other.name), price(other.price), quantity(other.quantity) {
        std::cout << "Copy constructor called for " << name << std::endl;
    }

    GroceryItem& operator=(const GroceryItem& other) {
        if (this != &other) {
            name = other.name;
            price = other.price;
            quantity = other.quantity;
            std::cout << "Assignment operator called for " << name << std::endl;
        }
        return *this;
    }

    virtual ~GroceryItem() {
        std::cout << "Destructor called for " << name << std::endl;
    }

    int calculateSum() const {
        return price * quantity;
```



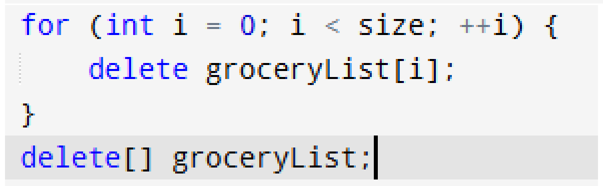
```
    int calculateSum() const {
        return price * quantity;
    }

    virtual void print() const {
        std::cout << name << " - PHP " << price << " x" << quantity
                  << " = PHP " << calculateSum() << std::endl;
    }

    std::string getName() const {
        return name;
    }
;

lass Fruit : public GroceryItem {
ublic:
    Fruit(std::string n, int p, int q) : GroceryItem(n, p, q) {}
;

lass Vegetable : public GroceryItem {
ublic:
    Vegetable(std::string n, int p, int q) : GroceryItem(n, p, q) {}
 int main() {
     const int size = 4;

     GroceryItem** groceryList = new GroceryItem*[size];

     groceryList[0] = new Fruit("Apple", 10, 7);
     groceryList[1] = new Fruit("Banana", 10, 8);
     groceryList[2] = new Vegetable("Broccoli", 60, 12);
     groceryList[3] = new Vegetable("Lettuce", 50, 10);

     std::cout << "\n=== Jenna's Grocery List ===" << std::endl;
     for (int i = 0; i < size; ++i) {
         groceryList[i]->print();
     }

     return 0;
 }
```



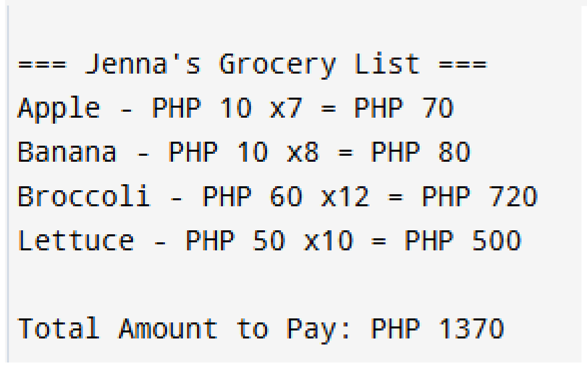
```
int TotalSum(GroceryItem** list, int size) {
    int total = 0;
    for (int i = 0; i < size; ++i) {
        total += list[i]->calculateTotal();
    }
    return total;
}
```



```
    int total = TotalSum(groceryList, size);
    std::cout << "\nTotal Amount to Pay: PHP " << total << std::endl;
```



```
for (int i = 0; i < size; ++i) {
    delete groceryList[i];
}
delete[] groceryList;
```



```

=== Jenna's Grocery List ===
Apple - PHP 10 x7 = PHP 70
Banana - PHP 10 x8 = PHP 80
Broccoli - PHP 60 x12 = PHP 720
Lettuce - PHP 50 x10 = PHP 500

Total Amount to Pay: PHP 1370
```

8. Conclusion

To conclude, this activity discussed about memory and how to allocate, and store by pointer. It also discusses how to use this memory allocation in the procedure by different scenarios like arrays and classes. In this activity, I think it was very challenging for me especially some parts are very new to me so I wasn't able to understand it easily. Yet, it was an interesting topic and I have gained knowledge and also experience in accessing memories and how to use them in arrays as well as the class

9. Assessment Rubric