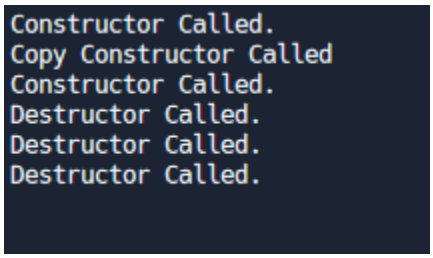
Activity No. 2	
ARRAYS, POINTERS AND DYNAMIC MEMORY ALLOCATION	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 09/11/2024
Section: CPE21S4	**Date Submitted:** 09/11/2024
Name(s): Sanchez, Christan Ray R.	**Instructor:** Ma'am Ma. Rizzete Sayo

6. Output

TABLE 2-1

SCREENSHOT

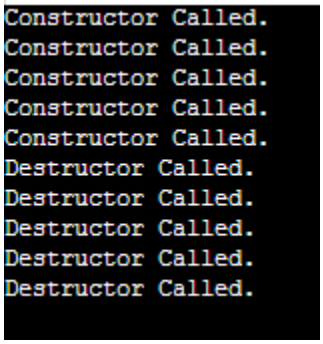


OBSERVATION

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

//Super class
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() {
  //these are the objects that are recalled in the main function
Student student1("Roman", 28);
Student student2(student1);
Student student3;
student3 = student2;
return 0;
}
```

TABLE 2-2

SCREENSHOT



OBSERVATION

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

//Super class
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};
}
```

TABLE 2-3

LOOP A

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


...Program finished with exit code 0
Press ENTER to exit console.
```

OBSERVATION

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

//Super class
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;
}
return 0;
}
```

LOOP B

```
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

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

//Super class
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 B
    studentList[i].printDetails();
    }
    return 0;
}
```

OUTPUT

```
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

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

//Super class
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;
}
```

7. Supplementary Activity

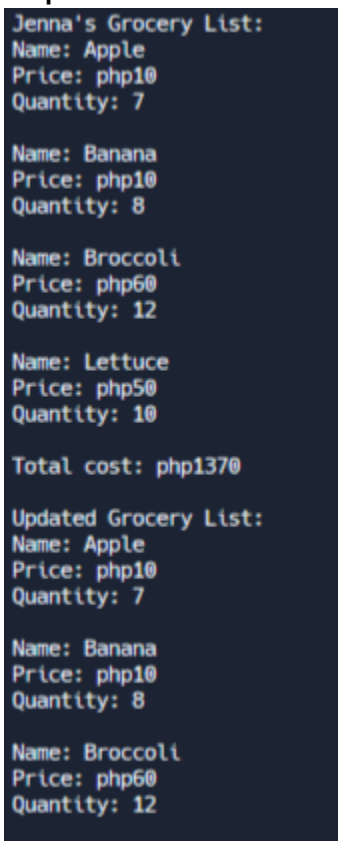
Observation:

```cpp
#include <string>

// Base class for food items
class FoodItem {
protected:
    std::string name;
    double price;
    int quantity;

public:
    // Constructor
    FoodItem(const std::string& newName, double newPrice, int newQuantity) :
        name(newName), price(newPrice), quantity(newQuantity) {}

    // Destructor
    virtual ~FoodItem() {}

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

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

    // Getters
    std::string getName() const { return name; }
    double getPrice() const { return price; }
    int getQuantity() const { return quantity; }

    // Calculate sum for a single item
    double calculateSum() const { return price * quantity; }
};

// Derived class for fruits
class Fruit : public FoodItem {
public:
    // Constructor
    Fruit(const std::string& newName, double newPrice, int newQuantity) :
        FoodItem(newName, newPrice, newQuantity) {}
```

```
    // Destructor
    ~Fruit() {}

    // Copy constructor
    Fruit(const Fruit& other) : FoodItem(other) {}

    // Copy assignment operator
    Fruit& operator=(const Fruit& other) {
        if (this != &other) {
            FoodItem::operator=(other);
        }
        return *this;
    }
};

// Derived class for vegetables
class Vegetable : public FoodItem {
public:
    // Constructor
    Vegetable(const std::string& newName, double newPrice, int newQuantity) :
        FoodItem(newName, newPrice, newQuantity) {}

    // Destructor
    ~Vegetable() {}

    // Copy constructor
    Vegetable(const Vegetable& other) : FoodItem(other) {}

    // Copy assignment operator
    Vegetable& operator=(const Vegetable& other) {
        if (this != &other) {
            FoodItem::operator=(other);
        }
        return *this;
    }
};

// Function to calculate the total sum of all items (Problem 3)
double TotalSum(FoodItem* groceryList[], int size) {
    double totalSum = 0;
    for (int i = 0; i < size; ++i) {
        totalSum += groceryList[i]->calculateSum();
    }
    return totalSum;
}
```

```cpp
int main() {
    // Create Jenna's grocery list (Problem 2)
    FoodItem* groceryList[5];
    groceryList[0] = new Fruit("Apple", 10.0, 7); // Apple 10php x7
    groceryList[1] = new Fruit("Banana", 10.0, 8); // Banana 10 php x8
    groceryList[2] = new Vegetable("Broccoli", 60.0, 12); // Broccoli 60 php x12
    groceryList[3] = new Vegetable("Lettuce", 50.0, 10); // Lettuce 50 php x10
    // (No need to add the 5th item as the list only has 4 items)

    // Display all details of the items (Problem 2)
    std::cout << "Jenna's Grocery List:\n";
    for (int i = 0; i < 4; ++i) {
        std::cout << "Name: " << groceryList[i]->getName() << "\n";
        std::cout << "Price: php" << groceryList[i]->getPrice() << "\n";
        std::cout << "Quantity: " << groceryList[i]->getQuantity() << "\n\n";
    }

    // Calculate the total sum (Problem 3)
    double totalCost = TotalSum(groceryList, 4);
    std::cout << "Total cost: php" << totalCost << std::endl;

    // Delete Lettuce from the grocery list (Problem 4)
    delete groceryList[3];
    groceryList[3] = nullptr; // Set the pointer to null to avoid dangling pointers

    // Display the updated grocery list (Problem 2)
    std::cout << "\nUpdated Grocery List:\n";
    for (int i = 0; i < 4; ++i) {
        if (groceryList[i] != nullptr) {
            std::cout << "Name: " << groceryList[i]->getName() << "\n";
            std::cout << "Price: php" << groceryList[i]->getPrice() << "\n";
            std::cout << "Quantity: " << groceryList[i]->getQuantity() << "\n\n";
        }
    }

    // Clean up the remaining dynamically allocated objects
    for (int i = 0; i < 4; ++i) {
        if (groceryList[i] != nullptr) {
            delete groceryList[i];
        }
    }

    return 0;
}
```

Output:



8. Conclusion

With an emphasis on object-oriented programming concepts like inheritance, constructors, destructors, copy constructors, and copy assignment operators, we learnt how to design and manage classes in C++ with this exercise. We implemented and processed arrays of objects, computed totals while managing lists, and handled various grocery item kinds consistently using polymorphism and dynamic memory management. Creating classes, making sure that memory was managed properly, creating functions for computation and display, testing, and debugging the code were all part of the process. Making a grocery list, figuring out how much it would cost, and editing the list by taking things off were the extra tasks. In general, the task proved to be a beneficial exercise in utilizing OOP principles, and I think I executed the task effectively, attaining the intended results. On the other hand, documentation, code optimization, and error handling could all need some work.

9. Assessment Rubric