Activity No. 2.1	
Hands-on Activity 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: 9/12/24**
Name(s): Santos, Andrei R.	**Instructor: Professor Maria Rizette Sayo**
6. Output	

Table 2-1. Initial Driver Program

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

```
/tmp/8y6BHpDDpa.o
Constructor Called.
Copy Constructor Called
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.


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

Observation:	The code creates functions that show creating, and copying, and also deleting student objects. We can see the obstructor and as well as the destructor. In the main function, three students are created, one with the first one creating, one by copying the first one which is the constructor, and one by assigning the copied details, the copy constructor and indeed and shows the attributes.

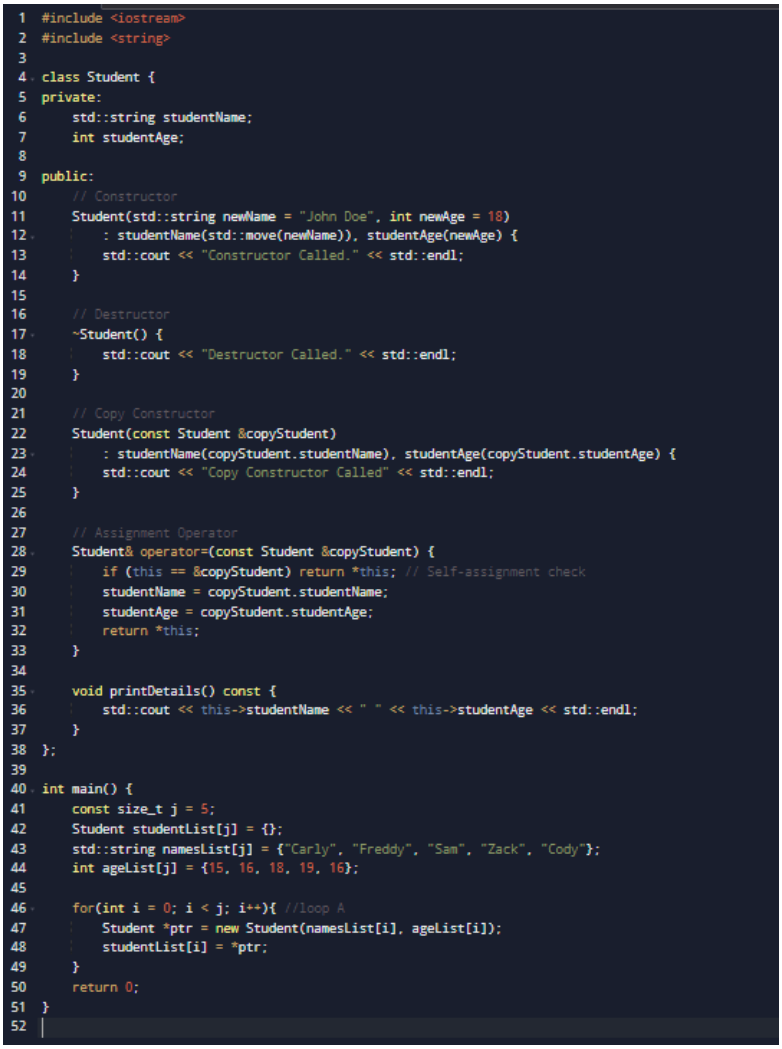
Table 2-2. Modified Driver Program with Student Lists

Screenshot:

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

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;
    }

    // Destructor
    ~Student() {
        std::cout << "Destructor Called." << std::endl;
    }

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

    // Assignment Operator
    Student& operator=(const Student &copyStudent) {
        if (this == &copyStudent) return *this; // Self-assignment check
        studentName = copyStudent.studentName;
        studentAge = copyStudent.studentAge;
        return *this;
    }

    void printDetails() const {
        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/AzLRODyzaw.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 also shows a constructor, destructor, copy constructor, and assignment operator to manage student details which is already in code 2.1, but we just changed and added some code. The constructor initializes the information of the student then a message is printed after. In the main function, the Student objects and also the names and ages are already initialized but not yet shown, but shows constructor called and obstructor called.

Table 2-3. Final Driver Program Loop A	
Observation	In loop A, we can see that we created the nameList and agelist in the code, pertaining to their corresponding position which is the student list. After that it is not interpreted after the assigning is complete.

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

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;
    }

    // Destructor
    ~Student() {
        std::cout << "Destructor Called." << std::endl;
    }

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

    // Assignment Operator
    Student& operator=(const Student &copyStudent) {
        if (this == &copyStudent) return *this; // Self-assignment check
        studentName = copyStudent.studentName;
        studentAge = copyStudent.studentAge;
        return *this;
    }

    void printDetails() const {
        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	
Observation	In loop B, the program already shows the studentlist and now calls the details. This method now shows and prints the name and the age of the student that we can after we run in the interpreter.
Output	

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

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;
    }

    // Destructor
    ~Student() {
        std::cout << "Destructor Called." << std::endl;
    }

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

    // Assignment Operator
    Student& operator=(const Student &copyStudent) {
        if (this == &copyStudent) return *this; // Self-assignment check
        studentName = copyStudent.studentName;
        studentAge = copyStudent.studentAge;
        return *this;
    }

    void printDetails() const {
        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;
}
```

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

class Student {
private:
    string studentName;
    int studentAge;

public:
    // Constructor
    Student(string newName = "John Doe", int newAge = 18)
        : studentName(move(newName)), studentAge(newAge) {
        cout << "Constructor Called." << endl;
    }

    // Destructor
    ~Student() {
        cout << "Destructor Called." << endl;
    }

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

    // Assignment Operator
    Student& operator=(const Student &copyStudent) {
        if (this == &copyStudent) return *this; // Self-assignment check
        studentName = copyStudent.studentName;
        studentAge = copyStudent.studentAge;
        return *this;
    }

    void printDetails() const {
        cout << this->studentName << " " << this->studentAge << endl;
    }
};

int main() {
    const size_t j = 5;
    Student studentList[j] = {};
    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/uqvms3kWnM.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 ===
```

Observation	After we combined and ran the program, we can see that it interpreted the constructors, and as well as the destructor and lastly it shows the name and the age of the student.

7. Supplementary Activity

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

using namespace std;

// Fruit class definition
class Fruit {
public:
    string name;
    double price;
    int quantity;
    // Constructor
    Fruit(string n, double p, int q) : name(n), price(p), quantity(q) {}
    // Destructor
    ~Fruit() {}
    // Copy Constructor
    Fruit(const Fruit &other) : name(other.name), price(other.price), quantity(other.quantity) {}
    // Copy Assignment Operator
    Fruit& operator=(const Fruit &other) {
        if (this == &other) return *this;
        name = other.name;
        price = other.price;
        quantity = other.quantity;
        return *this;
    }
    // Calculate sum
    double calculateSum() const {
        return price * quantity;
    }
};

// Vegetable class definition
class Vegetable {
public:
    string name;
    double price;
    int quantity;
    // Constructor
    Vegetable(string n, double p, int q) : name(n), price(p), quantity(q) {}
    // Destructor
    ~Vegetable() {}
    // Copy Constructor
    Vegetable(const Vegetable &other) : name(other.name), price(other.price), quantity(other.quantity) {}
    // Copy Assignment Operator
    Vegetable& operator=(const Vegetable &other) {
        if (this == &other) return *this;
        name = other.name;
        price = other.price;
        quantity = other.quantity;
        return *this;
    }
    // Calculate sum
    double calculateSum() const {
        return price * quantity;
    }
};

```

```cpp
// Function to calculate total sum
double TotalSum(Fruit fruits[], int fruitCount, Vegetable vegetables[], int vegetableCount) {
    double total = 0;
    for (int i = 0; i < fruitCount; ++i) {
        total += fruits[i].calculateSum();
    }
    for (int i = 0; i < vegetableCount; ++i) {
        total += vegetables[i].calculateSum();
    }
    return total;
}
// Function to remove a vegetable by name
int removeVegetable(Vegetable vegetables[], int vegetableCount, string nameToRemove) {
    for (int i = 0; i < vegetableCount; ++i) {
        if (vegetables[i].name == nameToRemove) {
            // Shift elements to the left
            for (int j = i; j < vegetableCount - 1; ++j) {
                vegetables[j] = vegetables[j + 1];
            }
            return vegetableCount - 1; // Reduce the size of the array
        }
    }
    return vegetableCount; // No removal if item not found
}
int main() {
    // Creating GroceryList
    Fruit fruits[] = {
        Fruit("Apple", 10, 7),
        Fruit("Banana", 10, 8)
    };
    int fruitCount = sizeof(fruits) / sizeof(fruits[0]);

    Vegetable vegetables[] = {
        Vegetable("Broccoli", 60, 12),
        Vegetable("Lettuce", 50, 10)
    };
    int vegetableCount = sizeof(vegetables) / sizeof(vegetables[0]);

    cout << "This is the original Grocery List:" << endl;
    for (int i = 0; i < fruitCount; ++i) {
        cout << fruits[i].name << " PHP " << fruits[i].price << " (x" << fruits[i].quantity << ")" << endl;
    }
    for (int i = 0; i < vegetableCount; ++i) {
        cout << vegetables[i].name << " PHP " << vegetables[i].price << " (x" << vegetables[i].quantity << ")" << endl;
    }

    // Remove Lettuce from the vegetable list
    vegetableCount = removeVegetable(vegetables, vegetableCount, "Lettuce");

    cout << "\nThe updated Grocery List:" << endl;
    for (int i = 0; i < fruitCount; ++i) {
        cout << fruits[i].name << " PHP " << fruits[i].price << " (x" << fruits[i].quantity << ")" << endl;
    }
    for (int i = 0; i < vegetableCount; ++i) {
        cout << vegetables[i].name << " PHP " << vegetables[i].price << " (x" << vegetables[i].quantity << ")" << endl;
    }

    double total = TotalSum(fruits, fruitCount, vegetables, vegetableCount);
    cout << "\nThe total Sum: PHP " << total << endl;

    return 0;
}
```

```
/tmp/QZdF4AQjza.o
This is the original Grocery List:
Apple PHP 10 (x7)
Banana PHP 10 (x8)
Broccoli PHP 60 (x12)
Lettuce PHP 50 (x10)

The updated Grocery List:
Apple PHP 10 (x7)
Banana PHP 10 (x8)
Broccoli PHP 60 (x12)

The total Sum: PHP 870
```

8. Conclusion

In conclusion to this activity, still, I am adjusting since we are learning python programming language and switched back to c++ programming language. I have learned the pointers which are the constructor, its destructor, and also copying its constructor, the assignment of variables in arrays by storing such as the vegetables and fruits. I am a little confused with some parts so I need to review it again, that is why I always do trial and error. Overall there are still learnings I collected for more future code programs.

9. Assessment Rubric