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: 09/11/2024**
Section: CPE21s4	**Date Submitted:09/11/2024**
Name(s): Anna Marie Zolina	**Instructor: Ms. Maria Rizette Sayo**
6. Output	

Table 2-1. Initial Driver Program

Screenshot

main.cpp

```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;
}
```

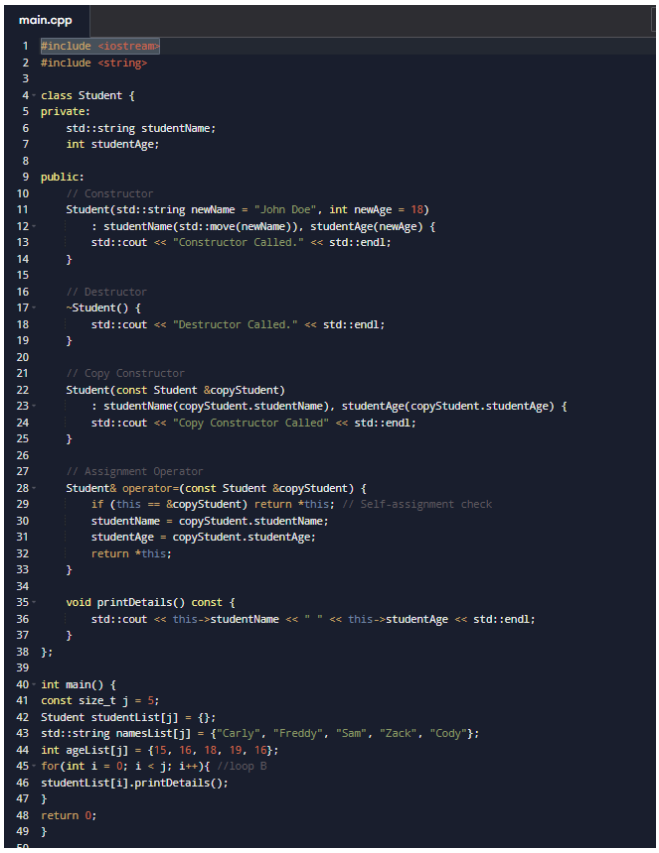
```
/tmp/yYVQTx8PXl.o
Constructor Called.
Copy Constructor Called
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

Observation	In this function, calls are made to the constructor and destructor.

Table 2-2. Modified Driver Program with Student Lists

Screenshot	main.cpp 1 #include <iostream> 2 #include <string> 3 4 class Student { 5 private: 6 std::string studentName; 7 int studentAge; 8 9 public: 10 // Constructor 11 Student(std::string newName = "John Doe", int newAge = 18) 12 : studentName(std::move(newName)), studentAge(newAge) { 13 std::cout << "Constructor Called." << std::endl; 14 } 15 16 // Destructor 17 ~Student() { 18 std::cout << "Destructor Called." << std::endl; 19 } 20 21 // Copy Constructor 22 Student(const Student &copyStudent) 23 : studentName(copyStudent.studentName), studentAge(copyStudent.studentAge) { 24 std::cout << "Copy Constructor Called" << std::endl; 25 } 26 27 // Assignment Operator 28 Student& operator=(const Student &copyStudent) { 29 if (this == &copyStudent) return *this; // Self-assignment check 30 studentName = copyStudent.studentName; 31 studentAge = copyStudent.studentAge; 32 return *this; 33 } 34 35 void printDetails() const { 36 std::cout << this->studentName << " " << this->studentAge << std::endl; 37 } 38 }; 39 40 int main() { 41 const size_t j = 5; 42 Student studentList[j] = {}; 43 std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"}; 44 int ageList[j] = {15, 16, 18, 19, 16}; 45 return 0; 46 } /tmp/U70dxRaaYz.o Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Destructor Called. Destructor Called. Destructor Called. Destructor Called. Destructor Called.
Observation	In this function, calls are made to the constructor and destructor.

Table 2-3. Final Driver Program

Loop A	main.cpp ```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; } ``` 
Observation	There is just one distinction between loops B and A: the lists describing the data are stored inside their routines.
Loop B	main.cpp ```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; } ``` 

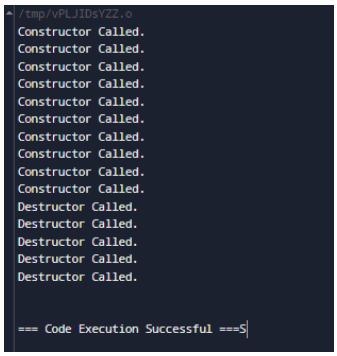
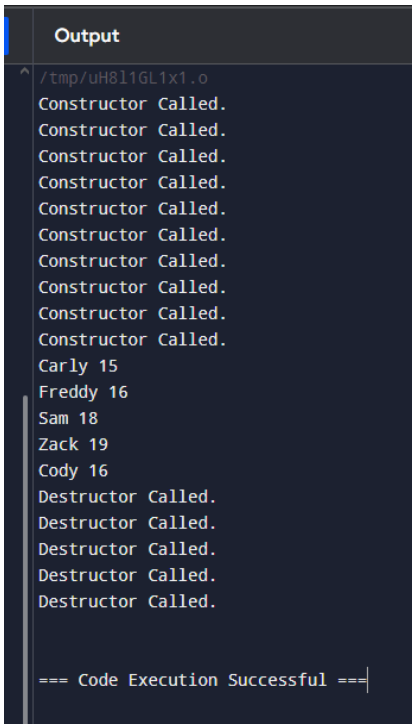
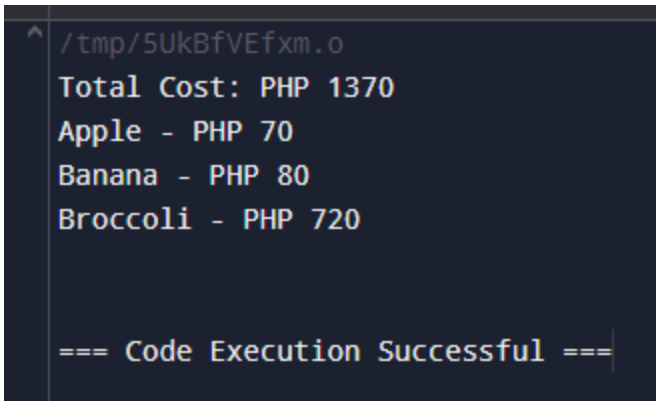
Observation	There is just one distinction between loops B and A: the lists describing the data are stored inside their routines.
Output	LOOP A:  /tmp/vPLJIDsYZZ.o Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Destructor Called. Destructor Called. Destructor Called. Destructor Called. Destructor Called. === Code Execution Successful ===S **LOOP B:**  /tmp/xm5BXRRABz.o 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 === **OVERALL OUTPUT:**  Output /tmp/uH8l1GL1x1.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	Although the given code shows how to allocate dynamic memory for the creation of Student objects, it does not properly deallocate memory, which could result in memory leaks.

Table 2-4. Modifications/Corrections Necessary	
Modifications	NONE
Observation	NONE

7. Supplementary Activity

```cpp
#include <iostream>
#include <cstring> // For strcpy and strlen
using namespace std;

class GroceryItem {
    char* itemName;
    double itemPrice;
    int itemQuantity;

public:
    // Constructor
    GroceryItem(const char* name = "", double price = 0, int quantity = 0) {
        itemName = new char[strlen(name) + 1];
        strcpy(itemName, name);
        itemPrice = price;
        itemQuantity = quantity;
    }

    // Destructor
    ~GroceryItem() {
        delete[] itemName; // Deallocate memory for the item name
    }

    // Calculate the total price for the item
    double calculateTotalPrice() const {
        return itemPrice * itemQuantity;
    }

    // Getter for the name
    const char* getItemName() const {
        return itemName;
    }
};

// Function to calculate the total sum of all items in the list
double calculateTotalCost(GroceryItem** groceryList, size_t listSize) {
    double totalCost = 0.0;
    for (size_t i = 0; i < listSize; ++i) {
        totalCost += groceryList[i]->calculateTotalPrice();
    }
    return totalCost;
}

// Function to delete an item from the list and shift remaining items
void removeItem(GroceryItem**& groceryList, size_t& listSize, const char* itemName) {
    for (size_t i = 0; i < listSize; ++i) {
        if (strcmp(groceryList[i]->getItemName(), itemName) == 0) {
            delete groceryList[i];

            // Shift remaining items left in the array
            for (size_t j = i; j < listSize - 1; ++j) {
                groceryList[j] = groceryList[j + 1];
            }
            --listSize;
            break;
        }
    }
}

int main() {
    size_t groceryListSize = 4;
    GroceryItem** groceryList = new GroceryItem*[groceryListSize];

    // Adding items to the groceryList array dynamically
    groceryList[0] = new GroceryItem("Apple", 10, 7);
    groceryList[1] = new GroceryItem("Banana", 10, 8);
    groceryList[2] = new GroceryItem("Broccoli", 60, 12);
    groceryList[3] = new GroceryItem("Lettuce", 50, 10);

    // Calculate the total cost of all items
    double totalCost = calculateTotalCost(groceryList, groceryListSize);
    cout << "Total Cost: PHP " << totalCost << endl;

    // Remove Lettuce from the list
    removeItem(groceryList, groceryListSize, "Lettuce");

    // Print the remaining items
    for (size_t i = 0; i < groceryListSize; ++i) {
        cout << groceryList[i]->getItemName() << " - PHP " << groceryList[i]->calculateTotalPrice() <<
            endl;
    }

    // Clean up remaining items
    for (size_t i = 0; i < groceryListSize; ++i) {
        delete groceryList[i];
    }

    delete[] groceryList; // Deallocate the array

    return 0;
}
```



8. Conclusion

Provide the following:
• Summary of lessons learned
• Analysis of the procedure
• Analysis of the supplementary activity
• Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

The lesson covered object creation, programming problem solving, static and dynamic memory allocation, and basic programming structures like loops, all of which were shown to have real-world applications in the use of student information. A simple grocery list management system is implemented using the C++ code, which makes use of dynamic memory allocation for the functions of item storage, cost estimation, and item removal. Although I now study Python for all of my programming classes, I believe that I performed rather well by finishing this exercise with some knowledge of C++. However, using C++ was challenging, and I still struggle to comprehend it. The assignment operator and operations like loops and void are the main areas where I still need to work on improving.

9. Assessment Rubric