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):BONIFACIO, NYKO ADREIN L.	**Instructor: Dr. Sayo**
6. Output	

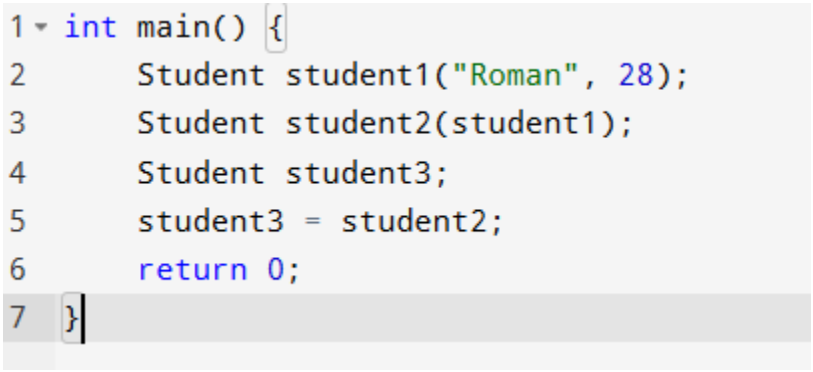
SCREENSHOT	
OBSERVATION	The constructor is called when student1 is created. The copy constructor is called when student2 is created from student1. The default constructor is called for student3. The copy assignment operator is used to copy student2 to student3. The destructors for student1, student2, and student3 are called when they go out of scope at the end of main().

Table 2-1. Initial Driver Program

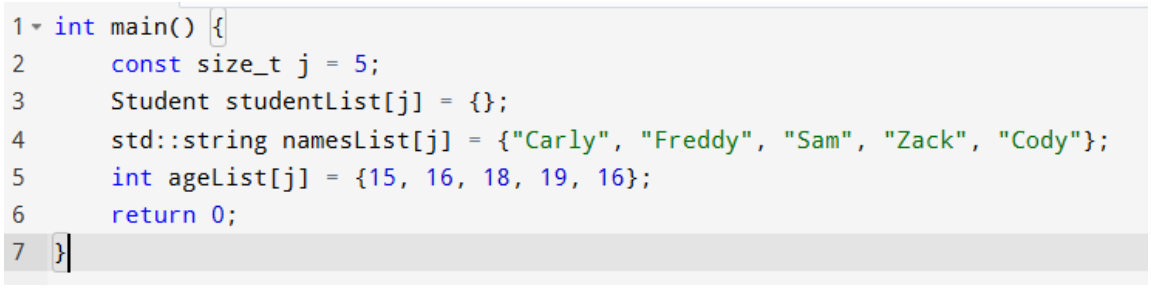
SCREENSHOT	
OBSERVATION	No dynamic memory allocation occurs here; everything is statically allocated. Each Student object in studentList is created using the default constructor. No output for the constructors since no custom initialization happens (default constructor). If the printDetails function were called, it would print "John Doe 18" five times, as that's the default value for the Student class.

Table 2-2. Modified Driver Program with Student Lists

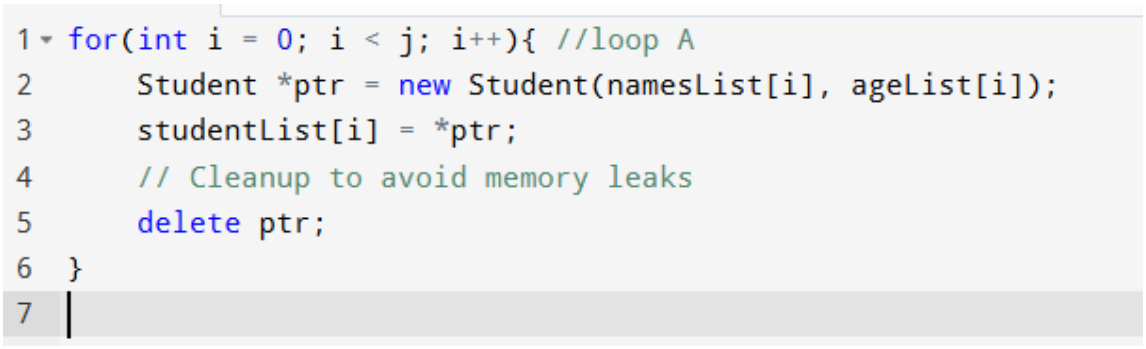
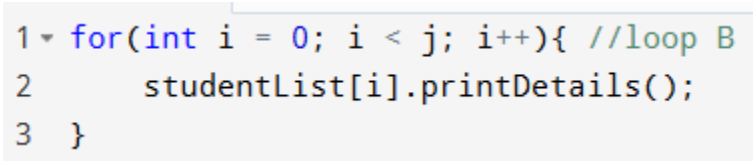
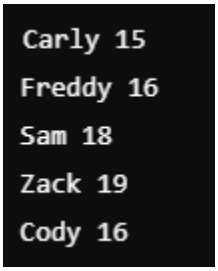
LOOP A	
OBSERVATION	Loop A dynamically allocates a new Student object for each entry in namesList and ageList. Each dynamically allocated object is assigned to the corresponding index in the studentList array.
LOOP B	
OBSERVATION	Loop B iterates through the studentList array and calls the printDetails() method for each Student object, which prints the name and age of each student.
OUTPUT	
OBSERVATION	This indicates that the student objects are correctly created and stored in the studentList array, and their details are printed correctly.

Table 2-3. Final Driver Program

MODIFICATIONS	```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;
	        delete ptr; // Free dynamically allocated memory
	    }
	
	    for(int i = 0; i < j; i++){ //loop B
	        studentList[i].printDetails();
	    }
	
	    return 0;
	}```

OBSERVATION	The code now correctly deallocates memory for each dynamically allocated Student object, preventing memory leaks. The delete ptr; statement in Loop A ensures that the memory used by each Student object created with new is freed after being copied to the studentList array.

Table 2-4. Modifications/Corrections Necessary

7. Supplementary Activity

main.cpp

Share Run

```cpp
#include <iostream>
#include <cstring>

using namespace std;

class Item {
protected:
    char name[50];
    double price;
    int quantity;

public:
    // Constructor
    Item(const char* name, double price, int quantity) : price(price), quantity(quantity) {
        strncpy(this->name, name, sizeof(this->name) - 1);
        this->name[sizeof(this->name) - 1] = '\0'; // Ensure null termination
    }

    // Virtual Destructor
    virtual ~Item() {}

    // Copy Constructor
    Item(const Item& other) : price(other.price), quantity(other.quantity) {
        strncpy(this->name, other.name, sizeof(this->name) - 1);
        this->name[sizeof(this->name) - 1] = '\0'; // Ensure null termination
    }

    // Copy Assignment Operator
    Item& operator=(const Item& other) {
        if (this != &other) {
            strncpy(this->name, other.name, sizeof(this->name) - 1);
            this->name[sizeof(this->name) - 1] = '\0'; // Ensure null termination
            this->price = other.price;
            this->quantity = other.quantity;
        }
        return *this;
    }

    // Calculate total price
    double calculateTotal() const {
        return price * quantity;
    }

    // Display item details
    virtual void display() const {
        cout << "Name: " << name << ", Price: PHP " << price
             << ", Quantity: " << quantity
             << ", Total: PHP " << calculateTotal() << endl;
    }

    // Check if name matches
    bool isName(const char* nameToCompare) const {
        return strcmp(name, nameToCompare) == 0;
    }
};

class Fruit : public Item {
public:
```

```
    // Constructor
    Fruit(const char* name, double price, int quantity) : Item(name, price, quantity) {}

    // Copy Constructor
    Fruit(const Fruit& other) : Item(other) {}

    // Copy Assignment Operator
    Fruit& operator=(const Fruit& other) {
        if (this != &other) {
            Item::operator=(other);
        }
        return *this;
    }
};

class Vegetable : public Item {
public:
    // Constructor
    Vegetable(const char* name, double price, int quantity) : Item(name, price, quantity) {}

    // Copy Constructor
    Vegetable(const Vegetable& other) : Item(other) {}

    // Copy Assignment Operator
    Vegetable& operator=(const Vegetable& other) {
        if (this != &other) {
            Item::operator=(other);
        }
        return *this;
    }
};

// Function to calculate the total sum of all items
double TotalSum(Item* list[], int size) {
    double sum = 0.0;
    for (int i = 0; i < size; ++i) {
        sum += list[i]->calculateTotal();
    }
    return sum;
}

int main() {
    // Create grocery list
    const int initialSize = 4;
    Item* GroceryList[initialSize] = {
        new Fruit("Apple", 10.0, 7),
        new Fruit("Banana", 10.0, 8),
        new Vegetable("Broccoli", 60.0, 12),
        new Vegetable("Lettuce", 50.0, 10)
    };

    int listSize = initialSize;

    // Display all items
    cout << "Initial Grocery List:\n";
    for (int i = 0; i < listSize; ++i) {
        GroceryList[i]->display();
    }

```

```
    // Calculate and display total sum
    double totalSum = TotalSum(GroceryList, listSize);
    cout << "Total Sum: PHP " << totalSum << endl;

    // Remove Lettuce
    int indexToRemove = -1;
    for (int i = 0; i < listSize; ++i) {
        if (GroceryList[i]->isName("Lettuce")) {
            indexToRemove = i;
            break;
        }
    }

    if (indexToRemove != -1) {
        delete GroceryList[indexToRemove];
        for (int i = indexToRemove; i < listSize - 1; ++i) {
            GroceryList[i] = GroceryList[i + 1];
        }
        GroceryList[listSize - 1] = nullptr; // Nullify the last pointer
        --listSize;
    }

    // Display remaining items
    cout << "\nAfter removing Lettuce:\n";
    for (int i = 0; i < listSize; ++i) {
        GroceryList[i]->display();
    }

    // Calculate and display new total sum
    totalSum = TotalSum(GroceryList, listSize);
    cout << "Total Sum after removal: PHP " << totalSum << endl;

    // Cleanup remaining items
    for (int i = 0; i < listSize; ++i) {
        delete GroceryList[i];
    }

    return 0;
}
```

Output

```
/tmp/qumsJrNbt9.o
Initial Grocery List:
Name: Apple, Price: PHP 10, Quantity: 7, Total: PHP 70
Name: Banana, Price: PHP 10, Quantity: 8, Total: PHP 80
Name: Broccoli, Price: PHP 60, Quantity: 12, Total: PHP 720
Name: Lettuce, Price: PHP 50, Quantity: 10, Total: PHP 500
Total Sum: PHP 1370

After removing Lettuce:
Name: Apple, Price: PHP 10, Quantity: 7, Total: PHP 70
Name: Banana, Price: PHP 10, Quantity: 8, Total: PHP 80
Name: Broccoli, Price: PHP 60, Quantity: 12, Total: PHP 720
Total Sum after removal: PHP 870


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

8. Conclusion

From this activity, I have gained more understanding on the use of pointers and arrays by implementing static and dynamic memory in C++. I constructed a class called `Student` and observed how constructors and destructors and other crucial functions work. I was able to make an array of students and provide dynamic memory allocation for any student. I also learned that memory is something that needs to be cleared after it has been used in order to avoid issues. In conclusion, I did decent, but I can do better to refresh knowledge on memory management and optimization of the code I wrote.

9. Assessment Rubric