Activity No. 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: 9/11/24**
Name(s): Solis, Paul Vincent M.	**Instructor: Rizette Sayo**
6. Output	

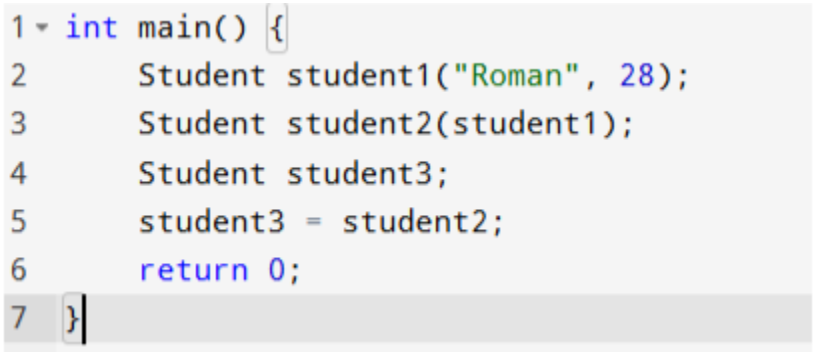
Screenshots	
Observation	Upon creation of student1, the constructor is invoked. When student2 is produced from student1, the copy constructor is called. For student3, the default constructor is invoked. Student2 is copied to Student3 using the copy assignment operator. At the conclusion of main(), when a student, two, or three goes out of scope, their destructors are called.

Table 2-1. Initial Driver Program

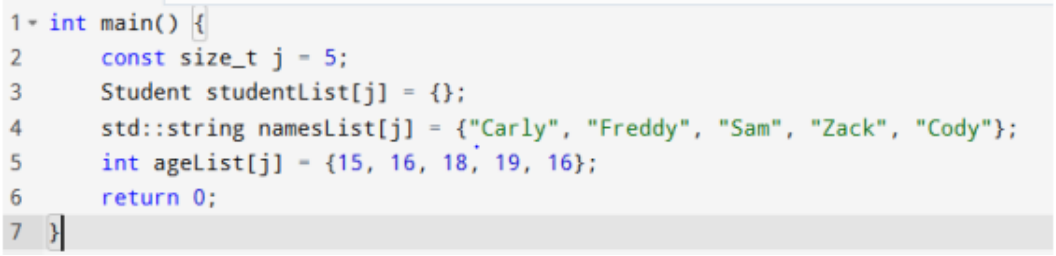
Screenshots	
Observations	The `studentList` array is initialized with 5 elements, each created using the default constructor of the `Student` class, which means no custom initialization occurs. As a result, if the `printDetails` function were called, it would print the default values of the `Student`

Table 2-2. Modified Driver Program with Student Lists

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

Observation	It is possible to fix the memory leak, shallow copy, undefined behavior, and pointer overuse in the code by either utilizing direct object creation and assignment or by employing appropriate pointer management.

Loop B	<pre>for(int i = 0; i < j; i++){ //loop B
	    studentList[i].printDetails();</pre>

Observation	Assuming that printDetails() is a legitimate Student class method and studentList is a valid array or container, the code is generally correct.

Output	<pre>Output
	/tmp/VbDIMApP7w.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.</pre>

Observation	loop a - Using new Student(namesList[i], ageList[i]), you dynamically allocate memory for a new Student object in this loop. The Student object that ptr refers to is subsequently assigned to the studentList[i] element. The copy constructor is called in this assignment since studentList[i] is an object and not a pointer. loop b: This loop executes the printDetails() function on each Student object as iterates through the studentList array.

Table 2-3. Final Driver Program

Modifications	n
Observation	

7. Supplementary Activity

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

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

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

    // Destructor
    ~Item() {}

    // Copy Constructor
    Item(const Item& other) : price(other.price), quantity(other.quantity) {
        std::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) {
            std::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;
```

```
        }
        return *this;
    }

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


    void display() const {
        std::cout << "Name: " << name << ", Price: PHP " << price
                  << ", Quantity: " << quantity
                  << ", Total: PHP " << calculateTotal() << std::endl;
    }


    bool isName(const char* nameToCompare) const {
        return std::strcmp(name, nameToCompare) == 0;
    }
};

class Fruit : public Item {
public:

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


    Fruit(const Fruit& other) : Item(other) {}


    Fruit& operator=(const Fruit& other) {
        if (this != &other) {
            Item::operator=(other);
        }
```

```
        return *this;
    }
};

class Vegetable : public Item {
public:

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


    Vegetable(const Vegetable& other) : Item(other) {}


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


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() {

    Item* GroceryList[] = {
        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 = sizeof(GroceryList) / sizeof(GroceryList[0]);


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


    double totalSum = TotalSum(GroceryList, listSize);
    std::cout << "Total Sum: PHP " << totalSum << std::endl;


    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];
        }
        --listSize;
    }


```

```
135
136
137     std::cout << "\nAfter removing Lettuce:\n";
138     for (int i = 0; i < listSize; ++i) {
139         GroceryList[i]->display();
140     }
141
142
143     totalSum = TotalSum(GroceryList, listSize);
144     std::cout << "Total Sum after removal: PHP " << totalSum << std::endl;
145
146
147     for (int i = 0; i < listSize; ++i) {
148         delete GroceryList[i];
149     }
150
151     return 0;
152 }
153
```

```
/tmp/T17Kep9IIU.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

Arrays, pointers, and dynamic memory allocation are fundamental programming concepts that enable developers to efficiently manage memory and create flexible, scalable, and effective software applications.

9. Assessment Rubric