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:** 11/09 24
Section:CPE21S4	**Date Submitted:** 11/09/24
Name(s): Pornobe, Reuel Christian, M.	**Instructor:** Prof. Maria Rizette Sayo

6. Output

Screenshot	Output /tmp/SCk5z3Q8ch.o Constructor Called. Copy Constructor Called Constructor Called. Destructor Called. Destructor Called. Destructor Called. === Code Execution Successful ===
Observation	In the first example of code, the initialization of the value of the variables were only intialized. It was not outputted by the program.

Table 2-1. Initial Driver Program

Screenshot	Output /tmp/0bRFSUwO9x.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	In this new modified driver, the student that was initialized has been printed. It utilized arrays to print all the initialized values in the arrays.

Table 2-2. Modified Driver Program with Student Lists

Loop A	

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

Loop B	

Screenshot	`for(int i = 0; i < j; i++){`
	`  //loop B`
	`studentList[i].printDetails();`
	`}`

Output	Constructor Called. Carly 15 Freddy 16 Sam 18 Zack 19 Cody 16 Destructor Called.
Observation	The loop dynamically creates Student objects with names and ages from the namesList and ageList arrays, then assigns these objects to the studentList array. This operation causes the dynamically allocated memory to be lost (leaked) because no deletion of the heap-allocated objects is performed. After the loop, studentList contains copies of the Student objects that were originally allocated dynamically, but the original heap-allocated objects are not properly cleaned up.

Table 2-3. Final Driver Program

Modifications	

```
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 (size_t i = 0; i < j; i++) {
        studentList[i] = new Student(namesList[i], ageList[i]);
    }

    for (size_t i = 0; i < j; i++) {
        studentList[i]->printDetails();
    }

    // Cleanup: delete dynamically allocated Student objects
    for (size_t i = 0; i < j; i++) {
        delete studentList[i];
    }

    return 0;
}
```

Observation	Now the loop that Dynamically allocates objects are now managed properly and correctly because all memory that was allocated is properly deallocated after using it.

7. Supplementary Activity

Jenna's Grocery List		
Apple	PHP 10	x7
Banana	PHP 10	x8
Broccoli	PHP 60	x12
Lettuce	PHP 50	x10

Jenna wants to buy the following fruits and vegetables for her daily consumption. However, she needs to distinguish between fruit and vegetable, as well as calculate the sum of prices that she has to pay in total.

Problem 1: Create a class for the fruit and the vegetable classes. Each class must have a constructor, deconstructor, copy constructor and copy assignment operator. They must also have all relevant attributes
(such as name, price and quantity) and functions (such as calculate sum) as presented in the problem description above.

Problem 2: Create an array GroceryList in the driver code that will contain all items in Jenna's Grocery List. You must then access each saved instance and display all details about the items.

Problem 3: Create a function TotalSum that will calculate the sum of all objects listed in Jenna's Grocery List.

Problem 4: Delete the Lettuce from Jenna's GroceryList list and deallocate the memory assigned.

main.cpp

Share

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

using namespace std;

// Problem 1: Create a class for the fruit and the vegetable classes.
class GroceryItem {
public:
    virtual ~GroceryItem() {}  // Virtual destructor
    virtual double calculateTotal() const = 0;
    virtual void display() const = 0;
};

// Problem 1: Create a class for the fruit and the vegetable classes.
class Fruit : public GroceryItem {
private:
    string itemName;
    double itemPrice;
    int itemQuantity;

public:
    // Constructor
    Fruit(const string& name, double price, int quantity)
        : itemName(name), itemPrice(price), itemQuantity(quantity) {}
    Fruit(const Fruit& other)
        : itemName(other.itemName), itemPrice(other.itemPrice), itemQuantity(other
            .itemQuantity) {}
    Fruit& operator=(const Fruit& other) {
        if (this != &other) {
            itemName = other.itemName;
            itemPrice = other.itemPrice;
            itemQuantity = other.itemQuantity;
        }
        return *this;
    }

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

main.cpp

Share

```cpp

    // Problem 3: Create a function TotalSum that will calculate the sum of all objects listed in Jenna's Grocery
    double calculateTotal() const override {
        return itemPrice * itemQuantity;
    }
    void display() const override {
        cout << "Fruit: " << itemName << ", Price: $" << itemPrice << ", Quantity: " << itemQuantity << endl;
    }
};

// Problem 1: Create a class for the fruit and the vegetable classes.
class Vegetable : public GroceryItem {
private:
    string itemName;
    double itemPrice;
    int itemQuantity;
public:
    // Constructor
    Vegetable(const string& name, double price, int quantity)
        : itemName(name), itemPrice(price), itemQuantity(quantity) {}
    Vegetable(const Vegetable& other)
        : itemName(other.itemName), itemPrice(other.itemPrice), itemQuantity(other.itemQuantity) {}
    Vegetable& operator=(const Vegetable& other) {
        if (this != &other) {
            itemName = other.itemName;
            itemPrice = other.itemPrice;
            itemQuantity = other.itemQuantity;
        }
        return *this;
    }

    // Destructor
    ~Vegetable() override {}

    // Problem 3: Create a function TotalSum that will calculate the sum of all objects listed in Jenna's Grocery
```

main.cpp

Share

```cpp

    // Problem 3: Create a function TotalSum that will calculate the sum of all object listed in Jenna's Grocery
    double calculateTotal() const override {
        return itemPrice * itemQuantity;
    }

    // Function to display details
    void display() const override {
        cout << "Vegetable: " << itemName << ", Price: $" << itemPrice << ", Quantity: " << itemQuantity << endl;
    }
};

int main() {
    const int LIST_SIZE = 5; // Example size

    // Problem 2: Create an array GroceryList in the driver code
    GroceryItem* groceryList[LIST_SIZE];
    groceryList[0] = new Fruit("Apple", 1.50, 10);
    groceryList[1] = new Vegetable("Carrot", 0.75, 5);
    groceryList[2] = new Fruit("Banana", 1.20, 6);
    groceryList[3] = new Vegetable("Lettuce", 2.00, 1);
    groceryList[4] = new Fruit("Orange", 1.80, 4);

    // Display details about all items
    cout << "Original Grocery List:" << endl;
    for (int i = 0; i < LIST_SIZE; ++i) {
        groceryList[i]->display();
    }


    double totalCost = 0.0;
    for (int i = 0; i < LIST_SIZE; ++i) {
        totalCost += groceryList[i]->calculateTotal();
    }
    cout << "Total Cost: $" << totalCost << endl;

```

```
    // Problem 4: Delete the Lettuce from Jenna's GroceryList list and de-allocate th
        memory assigned.
    delete groceryList[3]; // Lettuce
    groceryList[3] = nullptr;

    cout << "Updated Grocery List:" << endl;
    for (int i = 0; i < LIST_SIZE; ++i) {
        if (groceryList[i] != nullptr) {
            groceryList[i]->display();
        }
    }
    for (int i = 0; i < LIST_SIZE; ++i) {
        if (groceryList[i] != nullptr) {
            delete groceryList[i];
        }
    }

    return 0;
}
```

Output

```
/tmp/4aUcZy6cuj.o
Original Grocery List:
Fruit: Apple, Price: $1.5, Quantity: 10
Vegetable: Carrot, Price: $0.75, Quantity: 5
Fruit: Banana, Price: $1.2, Quantity: 6
Vegetable: Lettuce, Price: $2, Quantity: 1
Fruit: Orange, Price: $1.8, Quantity: 4
Total Cost: $35.15
Updated Grocery List:
Fruit: Apple, Price: $1.5, Quantity: 10
Vegetable: Carrot, Price: $0.75, Quantity: 5
Fruit: Banana, Price: $1.2, Quantity: 6
Fruit: Orange, Price: $1.8, Quantity: 4


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

8. Conclusion

For this lab activity, I learned how to use class in C++. I was able to learn the constructors and deconstructors and their uses. Constructors are used for initializing values in an object. Destructors are used for deleting to save memory and prevent memory leakage. Further, I learned that arrays are also pointers. Arrays are like shortcuts for a group of pointers.

For the analysis of the procedure, In the foremost part of the program, the necessary header files were added. Below that is the class Student which will have the initialization of the variables and functions inside the class. After that are the constructors and destructors. In the main driver is the calling of the class and its initialized variables and functions.

For the supplementary activity, I did a similar procedure to the earlier activity. I used class, constructors, and destructors. However, I used a subclass under the main class to create an organized and understandable code. Under superclass is where I initialized the variables and functions needed. Then, I made the subclass inherit after the Superlass.

In this lab activity, I fell short on understanding C++ since I am only familiar with python when it comes to coding OOP. It was a hard activity for me since I had to self-study in a short time all the syntax and meaning of OOP in C++.

9. Assessment Rubric