Activity No. 2	
ARRAYS, POINTERS AND DYNAMIC MEMORY ALLOCATION	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:11/9/2024**
Section:CPE21S1	**Date Submitted: 11/9/2024**
Name(s): Jimenez, Christian Joros R.	**Instructor: Ms. Sayo**

6. Output

Table 2-1
Output:

```
Constructor Called.
Copy Constructor Called
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

Observation: Calls to Constructors and Destructors: As you can see, objects are created (either directly or by copying them), and objects are destroyed (either when they leave their scope or are destroyed).

Table 2-2
Output:

```
Name: Carly, Age: 15
Name: Freddy, Age: 16
Name: Sam, Age: 18
Name: Zack, Age: 19
Name: Cody, Age: 16
```

Observation:
The details of every student kept in the studentList array are displayed in the output. The name and age of each student in the studentList are printed after the code has successfully created a Student object for each name and age in the namesList and ageList arrays.

Table 2-3
Loop A

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

Observations: In loop A, it creates new Student objects dynamically and then copies these objects into the studentList array.

Loop B

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

Observations:
The loop B Iterates Over the Index: j is 5, and the loop iterates from i = 0 to i < j. The loop accesses the i-th element of the studentList array, which is a Student object, for each index i. This means that it will run five times, once for each Student object in the studentList array.

Output:

```
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.
Destructor Called.
Destructor Called.
Destructor Called.
 Constructor Called.
Copy Constructor Called
Constructor Called.
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.
Destructor Called.
Destructor Called.
Destructor Called.
```

Observation:

Constructor messages, copy constructor messages, printed student details, and destructor messages make up the output. The memory leak that dynamically allocated Student objects that are improperly managed or deleted are the main problem that needs to be fixed.

7. Supplementary Activity

Problem 1:

```
#include <iostream>
#include <string>
#include <vector>

class Product {
protected:
	std::string name;
	double price;
	int quantity;

public:
	Product(std::string n = "Unknown", double p = 0.0, int q = 0)
	: name(n), price(p), quantity(q) {}

	virtual ~Product() {}

	Product(const Product& other)
	: name(other.name), price(other.price), quantity(other.quantity) {}

	Product& operator=(const Product& other) {
	if (this != &other) {
	name = other.name;
	price = other.price;
	quantity = other.quantity;
	}
	return *this;
	}


	double calculateTotalCost() const {
	return price * quantity;
	}

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

class Fruit : public Product {
private:
	std::string variety;

public:
	Fruit(std::string n = "Unknown", double p = 0.0, int q = 0, std::string v = "Unknown")
```

```
        : Product(n, p, q), variety(v) {}

        ~Fruit() override {}

        Fruit(const Fruit& other)
        : Product(other), variety(other.variety) {}

        Fruit& operator=(const Fruit& other) {
        if (this != &other) {
        Product::operator=(other);
        variety = other.variety;
        }
        return *this;
        }

        void display() const override {
        Product::display();
        std::cout << "Variety: " << variety << std::endl;
        }
};

class Vegetable : public Product {
private:
        std::string color;

public:

        Vegetable(std::string n = "Unknown", double p = 0.0, int q = 0, std::string c = "Unknown")
        : Product(n, p, q), color(c) {}


        ~Vegetable() override {}


        Vegetable(const Vegetable& other)
        : Product(other), color(other.color) {}


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


        void display() const override {
        Product::display();
        std::cout << "Color: " << color << std::endl;
        }
```

```
};

int main() {
	std::vector<Product*> groceryList;

	groceryList.push_back(new Fruit("Apple", 10.0, 7, "Red"));
	groceryList.push_back(new Fruit("Banana", 10.0, 8, "Yellow"));


	groceryList.push_back(new Vegetable("Broccoli", 60.0, 12, "Green"));
	groceryList.push_back(new Vegetable("Lettuce", 50.0, 10, "Green"));


	double totalCost = 0.0;
	for (const auto& product : groceryList) {
	product->display();
	totalCost += product->calculateTotalCost();
	}

	std::cout << "Total Cost: PHP " << totalCost << std::endl;


	for (auto& product : groceryList) {
	delete product;
	}

	return 0;
}
```

Output:

```
Apple, Price: PHP 10, Quantity: 7, Total Cost: PHP 70
Variety: Red
Banana, Price: PHP 10, Quantity: 8, Total Cost: PHP 80
Variety: Yellow
Broccoli, Price: PHP 60, Quantity: 12, Total Cost: PHP 720
Color: Green
Lettuce, Price: PHP 50, Quantity: 10, Total Cost: PHP 500
Color: Green
Total Cost: PHP 1370
```

Problem 2:

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

class Product {
protected:
	std::string name;
	double price;
```

```
        int quantity;

public:

        Product(std::string n = "Unknown", double p = 0.0, int q = 0)
        : name(n), price(p), quantity(q) {}


        virtual ~Product() {}


        Product(const Product& other)
        : name(other.name), price(other.price), quantity(other.quantity) {}


        Product& operator=(const Product& other) {
        if (this != &other) {
        name = other.name;
        price = other.price;
        quantity = other.quantity;
        }
        return *this;
        }


        double calculateTotalCost() const {
        return price * quantity;
        }


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

class Fruit : public Product {
private:
        std::string variety;

public:

        Fruit(std::string n = "Unknown", double p = 0.0, int q = 0, std::string v = "Unknown")
        : Product(n, p, q), variety(v) {}


        ~Fruit() override {}


        Fruit(const Fruit& other)
```

```
        : Product(other), variety(other.variety) {}


        Fruit& operator=(const Fruit& other) {
        if (this != &other) {
        Product::operator=(other);
        variety = other.variety;
        }
        return *this;
        }


        void display() const override {
        Product::display();
        std::cout << "Variety: " << variety << std::endl;
        }
};

class Vegetable : public Product {
private:
        std::string color;

public:

        Vegetable(std::string n = "Unknown", double p = 0.0, int q = 0, std::string c = "Unknown")
        : Product(n, p, q), color(c) {}


        ~Vegetable() override {}


        Vegetable(const Vegetable& other)
        : Product(other), color(other.color) {}


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


        void display() const override {
        Product::display();
        std::cout << "Color: " << color << std::endl;
        }
};

int main() {
```

```
        const size_t groceryListSize = 4;
        Product* GroceryList[groceryListSize];

        GroceryList[0] = new Fruit("Apple", 10.0, 7, "Red");
        GroceryList[1] = new Fruit("Banana", 10.0, 8, "Yellow");
        GroceryList[2] = new Vegetable("Broccoli", 60.0, 12, "Green");
        GroceryList[3] = new Vegetable("Lettuce", 50.0, 10, "Green");

        for (size_t i = 0; i < groceryListSize; ++i) {
        GroceryList[i]->display();
        }

        for (size_t i = 0; i < groceryListSize; ++i) {
        delete GroceryList[i];
        }

        return 0;
}
```

Output:

```
Name: Apple, Price: PHP 10, Quantity: 7, Total Cost: PHP 70
Variety: Red
Name: Banana, Price: PHP 10, Quantity: 8, Total Cost: PHP 80
Variety: Yellow
Name: Broccoli, Price: PHP 60, Quantity: 12, Total Cost: PHP 720
Color: Green
Name: Lettuce, Price: PHP 50, Quantity: 10, Total Cost: PHP 500
Color: Green
```

Problem 3:

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

class Product {
protected:
        std::string name;
        double price;
        int quantity;

public:

        Product(std::string n = "Unknown", double p = 0.0, int q = 0)
        : name(n), price(p), quantity(q) {}


        virtual ~Product() {}


        Product(const Product& other)
        : name(other.name), price(other.price), quantity(other.quantity) {}
```

```
        Product& operator=(const Product& other) {
        if (this != &other) {
        name = other.name;
        price = other.price;
        quantity = other.quantity;
        }
        return *this;
        }


        double calculateTotalCost() const {
        return price * quantity;
        }


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

class Fruit : public Product {
private:
        std::string variety;

public:

        Fruit(std::string n = "Unknown", double p = 0.0, int q = 0, std::string v = "Unknown")
        : Product(n, p, q), variety(v) {}


        ~Fruit() override {}


        Fruit(const Fruit& other)
        : Product(other), variety(other.variety) {}


        Fruit& operator=(const Fruit& other) {
        if (this != &other) {
        Product::operator=(other);
        variety = other.variety;
        }
        return *this;
        }


        void display() const override {
        Product::display();
```

```
        std::cout << "Variety: " << variety << std::endl;
        }
};

class Vegetable : public Product {
private:
        std::string color;

public:

        Vegetable(std::string n = "Unknown", double p = 0.0, int q = 0, std::string c = "Unknown")
        : Product(n, p, q), color(c) {}


        ~Vegetable() override {}


        Vegetable(const Vegetable& other)
        : Product(other), color(other.color) {}


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


        void display() const override {
        Product::display();
        std::cout << "Color: " << color << std::endl;
        }
};


double TotalSum(Product* list[], size_t size) {
        double totalSum = 0.0;
        for (size_t i = 0; i < size; ++i) {
        totalSum += list[i]->calculateTotalCost();
        }
        return totalSum;
}

int main() {

        const size_t groceryListSize = 4;
        Product* GroceryList[groceryListSize];
```

```
        GroceryList[0] = new Fruit("Apple", 10.0, 7, "Red");
        GroceryList[1] = new Fruit("Banana", 10.0, 8, "Yellow");
        GroceryList[2] = new Vegetable("Broccoli", 60.0, 12, "Green");
        GroceryList[3] = new Vegetable("Lettuce", 50.0, 10, "Green");

        for (size_t i = 0; i < groceryListSize; ++i) {
        GroceryList[i]->display();
        }

        double totalCost = TotalSum(GroceryList, groceryListSize);
        std::cout << "Total price of all products: PHP " << totalCost << std::endl;

        for (size_t i = 0; i < groceryListSize; ++i) {
        delete GroceryList[i];
        }

        return 0;
}
```

Output:

```
Apple, Price: PHP 10, Quantity: 7, Total Cost: PHP 70
Variety: Red
Banana, Price: PHP 10, Quantity: 8, Total Cost: PHP 80
Variety: Yellow
Broccoli, Price: PHP 60, Quantity: 12, Total Cost: PHP 720
Color: Green
Lettuce, Price: PHP 50, Quantity: 10, Total Cost: PHP 500
Color: Green
Total price of all products: PHP 1370
```

Problem 4:

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


class Product {
protected:
        std::string name;
        double price;
        int quantity;

public:

        Product(std::string n = "Unknown", double p = 0.0, int q = 0)
        : name(n), price(p), quantity(q) {}


        virtual ~Product() {}


        Product(const Product& other)
```

```
        : name(other.name), price(other.price), quantity(other.quantity) {}


        Product& operator=(const Product& other) {
        if (this != &other) {
        name = other.name;
        price = other.price;
        quantity = other.quantity;
        }
        return *this;
        }


        double calculateTotalCost() const {
        return price * quantity;
        }


        std::string getName() const {
        return name;
        }


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

class Fruit : public Product {
private:
        std::string variety;

public:

        Fruit(std::string n = "Unknown", double p = 0.0, int q = 0, std::string v = "Unknown")
        : Product(n, p, q), variety(v) {}


        ~Fruit() override {}

        Fruit(const Fruit& other)
        : Product(other), variety(other.variety) {}


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

```
        return *this;
        }


        void display() const override {
        Product::display();
        std::cout << "Variety: " << variety << std::endl;
        }
};

class Vegetable : public Product {
private:
        std::string color;

public:

        Vegetable(std::string n = "Unknown", double p = 0.0, int q = 0, std::string c = "Unknown")
        : Product(n, p, q), color(c) {}


        ~Vegetable() override {}


        Vegetable(const Vegetable& other)
        : Product(other), color(other.color) {}


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


        void display() const override {
        Product::display();
        std::cout << "Color: " << color << std::endl;
        }
};


double TotalSum(Product* list[], size_t size) {
        double totalSum = 0.0;
        for (size_t i = 0; i < size; ++i) {
        totalSum += list[i]->calculateTotalCost();
        }
        return totalSum;
}
```

```
int main() {

	const size_t initialSize = 4;
	size_t groceryListSize = initialSize;
	Product* GroceryList[initialSize];


	GroceryList[0] = new Fruit("Apple", 10.0, 7, "Red");
	GroceryList[1] = new Fruit("Banana", 10.0, 8, "Yellow");
	GroceryList[2] = new Vegetable("Broccoli", 60.0, 12, "Green");
	GroceryList[3] = new Vegetable("Lettuce", 50.0, 10, "Green");


	std::cout << "Grocery List before removing Lettuce:" << std::endl;
	for (size_t i = 0; i < groceryListSize; ++i) {
	GroceryList[i]->display();
	}


	for (size_t i = 0; i < groceryListSize; ++i) {
	if (GroceryList[i]->getName() == "Lettuce") {
	delete GroceryList[i];

	for (size_t j = i; j < groceryListSize - 1; ++j) {
	GroceryList[j] = GroceryList[j + 1];
	}

	GroceryList[groceryListSize - 1] = nullptr;

	--groceryListSize;
	break;
	}
	}


	std::cout << "Grocery List after removing Lettuce:" << std::endl;
	for (size_t i = 0; i < groceryListSize; ++i) {
	GroceryList[i]->display();
	}


	double totalCost = TotalSum(GroceryList, groceryListSize);
	std::cout << "Total Sum of Remaining Products: PHP " << totalCost << std::endl;


	for (size_t i = 0; i < groceryListSize; ++i) {
	delete GroceryList[i];
	}

	return 0;
}
```

Output:

```
Grocery List before removing Lettuce:
Name: Apple, Price: PHP 10, Quantity: 7, Total Cost: PHP 70
Variety: Red
Name: Banana, Price: PHP 10, Quantity: 8, Total Cost: PHP 80
Variety: Yellow
Name: Broccoli, Price: PHP 60, Quantity: 12, Total Cost: PHP 720
Color: Green
Name: Lettuce, Price: PHP 50, Quantity: 10, Total Cost: PHP 500
Color: Green
Grocery List after removing Lettuce:
Name: Apple, Price: PHP 10, Quantity: 7, Total Cost: PHP 70
Variety: Red
Name: Banana, Price: PHP 10, Quantity: 8, Total Cost: PHP 80
Variety: Yellow
Name: Broccoli, Price: PHP 60, Quantity: 12, Total Cost: PHP 720
Color: Green
Total Sum of Remaining Products: PHP 870
```

8. Conclusion

In this activity, I learned how to implement static and dynamic memory allocation in my code, I was also able to make dynamically allocated objects using pointers and arrays. This is one of the hardest activities i had encountered, mostly because I was still getting refreshed on my C++ skills so i took longer in designing and understanding pieces of code, but this activity was kind of hard to do, but I was able to accomplish it in the end. We made a code that uses arrays, memory allocation, pointers and arrays. I think I did quite good on this activity, at least a bit better than the previous one.

9. Assessment Rubric