Activity No. <2>	
<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:09/13/2024**
Name(s): Danica T. Guariño	**Instructor: Maria Rizette Sayo**

6. Output

Screenshot

main.cpp

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

class Student {
private:
    std::string name;
    int age;
public:
    Student(const std::string& n, int a) : name(n), age(a) {}
    void printDetails() const {
        std::cout << "Name: " << name << ", Age: " << age << std::endl;
    }
};

int main() {
    const size_t j = 5;
    std::vector<Student> studentList;
    std::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.emplace_back(namesList[i], ageList[i]);
    }

    for (const auto& student : studentList) {
        student.printDetails();
    }

    return 0;
}
```

Console

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

Observation

By just modifying an array is quite easy as it seems.

7. Supplementary Activity

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.

main.cpp

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

class Fruit {
private:
    std::string name;
    double price;
    int quantity;

public:

    Fruit(std::string newName = "Unknown", double newPrice = 0.0, int newQuantity = 0)
        : name(std::move(newName)), price(newPrice), quantity(newQuantity) {
        std::cout << "Fruit Constructor Called." << std::endl;
    }


    ~Fruit() {
        std::cout << "Fruit Destructor Called." << std::endl;
    }


    Fruit(const Fruit &copyFruit)
        : name(copyFruit.name), price(copyFruit.price), quantity(copyFruit.quantity) {
        std::cout << "Fruit Copy Constructor Called." << std::endl;
    }


    Fruit& operator=(const Fruit &copyFruit) {
        if (this == &copyFruit) {
            return *this;
        }
        std::cout << "Fruit Copy Assignment Operator Called." << std::endl;
        name = copyFruit.name;
        price = copyFruit.price;
        quantity = copyFruit.quantity;
        return *this;
```

Console

```
Fruit Constructor Called.
Fruit Copy Constructor Called.
Fruit Constructor Called.
Fruit: Apple, Price: 2.5, Quantity: 10, Total Cost: 25
Fruit: Apple, Price: 2.5, Quantity: 10, Total Cost: 25
Fruit: Unknown, Price: 0, Quantity: 0, Total Cost: 0
Vegetable Constructor Called.
Vegetable Copy Constructor Called.
Vegetable Constructor Called.
Vegetable Copy Assignment Operator Called.
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable Destructor Called.
Vegetable Destructor Called.
Vegetable Destructor Called.
Fruit Destructor Called.
Fruit Destructor Called.
Fruit Destructor Called.
```

main.cpp

main.cpp > Fruit > Vegetable > ~Vegetable

```cpp
        return *this;
    }


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


    void printDetails() const {
        std::cout << "Fruit: " << name << ", Price: " << price << ", Quantity: " << quantity << ", Total Cost: " << calculateSum() << std::endl;
    }
};

class Vegetable {
private:
    std::string name;
    double price;
    int quantity;

public:

    Vegetable(std::string newName = "Unknown", double newPrice = 0.0, int newQuantity = 0)
        : name(std::move(newName)), price(newPrice), quantity(newQuantity) {
        std::cout << "Vegetable Constructor Called." << std::endl;
    }


    ~Vegetable() {
        std::cout << "Vegetable Destructor Called." << std::endl;
    }


    Vegetable(const Vegetable &copyVegetable)
        : name(copyVegetable.name), price(copyVegetable.price), quantity(copyVegetable.quantity) {
```

Console | Shell | AI

Run | Ask AI | 1s

```
Fruit Constructor Called.
Fruit Copy Constructor Called.
Fruit Constructor Called.
Fruit: Apple, Price: 2.5, Quantity: 10, Total Cost: 25
Fruit: Apple, Price: 2.5, Quantity: 10, Total Cost: 25
Fruit: Unknown, Price: 0, Quantity: 0, Total Cost: 0
Vegetable Constructor Called.
Vegetable Copy Constructor Called.
Vegetable Constructor Called.
Vegetable Copy Assignment Operator Called.
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable Destructor Called.
Vegetable Destructor Called.
Vegetable Destructor Called.
Fruit Destructor Called.
Fruit Destructor Called.
Fruit Destructor Called.
```

main.cpp

main.cpp > Fruit > Vegetable > ~Vegetable

```cpp
    quantity(copyVegetable.quantity) {
        std::cout << "Vegetable Copy Constructor Called." << std::endl;
    }


    Vegetable& operator=(const Vegetable &copyVegetable) {
        if (this == &copyVegetable) {
            return *this;
        }
        std::cout << "Vegetable Copy Assignment Operator Called." << std::endl;
        name = copyVegetable.name;
        price = copyVegetable.price;
        quantity = copyVegetable.quantity;
        return *this;
    }


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

    void printDetails() const {
        std::cout << "Vegetable: " << name << ", Price: " << price << ", Quantity: " << quantity << ", Total Cost: " << calculateSum() << std::endl;
    }
};

int main() {
    Fruit apple("Apple", 2.5, 10);
    Fruit banana(apple);
    Fruit orange;

    apple.printDetails();
    banana.printDetails();
    orange.printDetails();

    Vegetable carrot("Carrot", 1.2, 15);
```

Console | Shell | AI

Run

```
Fruit Constructor Called.
Fruit Copy Constructor Called.
Fruit Constructor Called.
Fruit: Apple, Price: 2.5, Quantity: 10, Total Cost: 25
Fruit: Apple, Price: 2.5, Quantity: 10, Total Cost: 25
Fruit: Unknown, Price: 0, Quantity: 0, Total Cost: 0
Vegetable Constructor Called.
Vegetable Copy Constructor Called.
Vegetable Constructor Called.
Vegetable Copy Assignment Operator Called.
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost
Vegetable Destructor Called.
Vegetable Destructor Called.
Vegetable Destructor Called.
Fruit Destructor Called.
Fruit Destructor Called.
Fruit Destructor Called.
```

```cpp
        quantity = copyVegetable.quantity;
        return *this;
    }


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

    void printDetails() const {
        std::cout << "Vegetable: " << name << ", Price: " << price << ", Quantity: " << quantity << ", Total Cost: " << calculateSum() << std::endl;
    }
};

int main() {
    Fruit apple("Apple", 2.5, 10);
    Fruit banana(apple);
    Fruit orange;

    apple.printDetails();
    banana.printDetails();
    orange.printDetails();

    Vegetable carrot("Carrot", 1.2, 15);
    Vegetable potato(carrot);
    Vegetable tomato;
    tomato = potato;

    carrot.printDetails();
    potato.printDetails();
    tomato.printDetails();

    return 0;
}
```

```
Fruit Constructor Called.
Fruit Copy Constructor Called.
Fruit Constructor Called.
Fruit: Apple, Price: 2.5, Quantity: 10, Total Cost: 25
Fruit: Apple, Price: 2.5, Quantity: 10, Total Cost: 25
Fruit: Unknown, Price: 0, Quantity: 0, Total Cost: 0
Vegetable Constructor Called.
Vegetable Copy Constructor Called.
Vegetable Constructor Called.
Vegetable Copy Assignment Operator Called.
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable: Carrot, Price: 1.2, Quantity: 15, Total Cost: 18
Vegetable Destructor Called.
Vegetable Destructor Called.
Vegetable Destructor Called.
Fruit Destructor Called.
Fruit Destructor Called.
Fruit Destructor Called.
```

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.

```cpp
#include <iostream>
#include <string>
int main() {
std::string groceryList[5][3] = {
{"Jennas_Grocery_List", "", ""},
{"Apple", "PHP 10", "7x"},
{"Banana", "PHP 10", "8x"},
{"Broccoli", "PHP 60", "12x"},
{"Lettuce", "PHP 50", "10x"}
};
for (int i = 0; i < 5; i++) {
for (int j = 0; j < 3; j++) {
std::cout << groceryList[i][j] << " ";
}
std::cout << std::endl;
}
return 0;
}
```

```
Jennas_Grocery_List
Apple PHP 10 7x
Banana PHP 10 8x
Broccoli PHP 60 12x
Lettuce PHP 50 10x
```

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

main.cpp

```cpp
#include <iostream>
#include <string>
class Produce {
protected:
std::string name;
double price;
int quantity;
public:
Produce(const std::string& n, double p, int q) : name(n), price(p), quantity(q) {}
virtual ~Produce() {}
double calculateSum() const {
return price * quantity;
}
std::string getName() const { return name; }
double getPrice() const { return price; }
int getQuantity() const { return quantity; }
void setName(const std::string& n) { name = n; }
void setPrice(double p) { price = p; }
void setQuantity(int q) { quantity = q; }
};
class Fruit : public Produce {
public:
Fruit(const std::string& n, double p, int q) : Produce(n, p, q) {}
Fruit(const Fruit& other) : Produce(other) {}
Fruit& operator=(const Fruit& other) {
if (this != &other) {
Produce::operator=(other);
}
return *this;
}
~Fruit() {}
};
class Vegetable : public Produce {
public:
Vegetable(const std::string& n, double p, int q) : Produce(n, p, q) {}
Vegetable(const Vegetable& other) : Produce(other) {}
Vegetable& operator=(const Vegetable& other) {
```

Console

```
Apple cost: PHP 70 (Quantity: 7)
Banana cost: PHP 80 (Quantity: 8)
Broccoli cost: PHP 720 (Quantity: 12)
Lettuce cost: PHP 500 (Quantity: 10)
Total cost: PHP 1370
```

main.cpp

```cpp
~Fruit() {}
};
class Vegetable : public Produce {
public:
Vegetable(const std::string& n, double p, int q) : Produce(n, p, q) {}
Vegetable(const Vegetable& other) : Produce(other) {}
Vegetable& operator=(const Vegetable& other) {
if (this != &other) {
Produce::operator=(other);

}
return *this;
}
~Vegetable() {}
};
int main() {
Fruit apple("Apple", 10, 7);
Fruit banana("Banana", 10, 8);
Vegetable broccoli("Broccoli", 60, 12);
Vegetable lettuce("Lettuce", 50, 10);
std::cout << "Apple cost: PHP " << apple.calculateSum() << " (Quantity: " << apple.getQuantity()
<< ")" << std::endl;
std::cout << "Banana cost: PHP " << banana.calculateSum() << " (Quantity: " <<
banana.getQuantity() << ")" << std::endl;
std::cout << "Broccoli cost: PHP " << broccoli.calculateSum() << " (Quantity: " <<
broccoli.getQuantity() << ")" << std::endl;
std::cout << "Lettuce cost: PHP " << lettuce.calculateSum() << " (Quantity: " <<
lettuce.getQuantity() << ")" << std::endl;
double totalSum = apple.calculateSum() + banana.calculateSum() + broccoli.calculateSum() +
lettuce.calculateSum();
std::cout << "Total cost: PHP " << totalSum << std::endl;
return 0;
return 0;
}
```

Console

```
Apple cost: PHP 70 (Quantity: 7)
Banana cost: PHP 80 (Quantity: 8)
Broccoli cost: PHP 720 (Quantity: 12)
Lettuce cost: PHP 500 (Quantity: 10)
Total cost: PHP 1370
```

Problem 4: Delete the Lettuce from Jenna's GroceryList list and de-allocate the memory assigned

main.cpp

main.cpp > Fruit > Fruit > ...

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


class Produce {
protected:
std::string name;
double price;
int quantity;
public:
Produce(const std::string& n, double p, int q) : name(n), price(p), quantity(q) {}
virtual ~Produce() {}
double calculateSum() const {
return price * quantity;
}

std::string getName() const { return name; }
double getPrice() const { return price; }
int getQuantity() const { return quantity; }
void setName(const std::string& n) { name = n; }
void setPrice(double p) { price = p; }
void setQuantity(int q) { quantity = q; }
};

class Fruit : public Produce {
public:
Fruit(const std::string& n, double p, int q) : Produce(n, p, q) {}
Fruit(const Fruit& other) : Produce(other) {}
Fruit& operator=(const Fruit& other) {
if (this != &other) {
Produce::operator=(other);
}

return *this;
}
};
~Fruit() {}
```

Console | Shell | AI

Run

```
Apple cost: PHP 70 (Quantity: 7)
Banana cost: PHP 80 (Quantity: 8)
Lettuce cost: PHP 500 (Quantity: 10)
Total cost: PHP 650
```

main.cpp

```cpp
if (this != &other) {
Produce::operator=(other);
}

return *this;
}

~Fruit() {}
};
class Vegetable : public Produce {
public:
Vegetable(const std::string& n, double p, int q) : Produce(n, p, q) {}
Vegetable(const Vegetable& other) : Produce(other) {}
Vegetable& operator=(const Vegetable& other) {
if (this != &other) {

Produce::operator=(other);
}
return *this;
}
~Vegetable() {}
};
int main() {
Fruit apple("Apple", 10, 7);
Fruit banana("Banana", 10, 8);
Vegetable lettuce("Lettuce", 50, 10);
std::cout << "Apple cost: PHP " << apple.calculateSum() << " (Quantity: " <<
apple.getQuantity() << ")" << std::endl;
std::cout << "Banana cost: PHP " << banana.calculateSum() << " (Quantity: " <<
banana.getQuantity() << ")" << std::endl;
std::cout << "Lettuce cost: PHP " << lettuce.calculateSum() << " (Quantity: " <<
lettuce.getQuantity() << ")" << std::endl;
double totalSum = apple.calculateSum() + banana.calculateSum() + lettuce.calculateSum();
std::cout << "Total cost: PHP " << totalSum << std::endl;
return 0;
}
```

Console

```
Apple cost: PHP 70 (Quantity: 7)
Banana cost: PHP 80 (Quantity: 8)
Lettuce cost: PHP 500 (Quantity: 10)
Total cost: PHP 650
```

8. Conclusion

In doing this activity, we learned and practiced our skills in programming arrays. The procedure of creating an array is easy to follow since there is a step by step procedure that we can follow in order to keep up with the process. This supplementary activity was easy to create array and add some variables to it. All in all, I had fun doing this activity but had a hard time since this task requires too much time to create.

9. Assessment Rubric