UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 4

Arrays

Submitted by:
Catahan, Joshua A.

Instructor:
Engr. Maria Rizette H. Sayo

08, 16, 2025

I. Objectives

Introduction

Array, in general, refers to an orderly arrangement of data elements. Array is a type of data structure that stores data elements in adjacent locations. Array is considered as linear data structure that stores elements of same data types. Hence, it is also called as a linear homogenous data structure.

This laboratory activity aims to implement the principles and techniques in:

- Writing algorithms using Array data structure
- Solve programming problems using dynamic memory allocation, arrays and pointers

II. Methods

Jenna's Grocery

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.

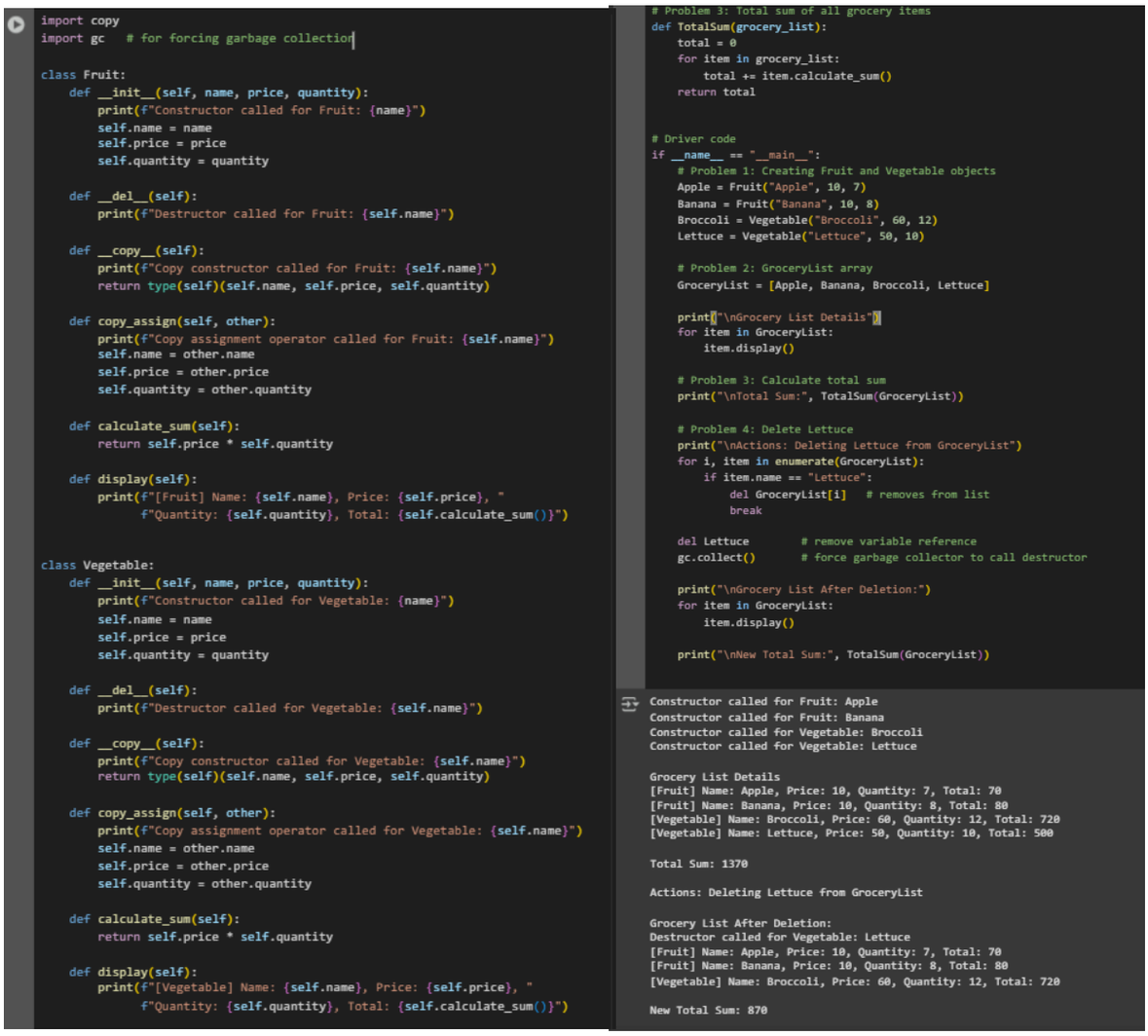
Problem 1: Create a class for the fruit and the vegetable classes. Each class must have a constructor, deconstructor, copy constructor and copy assignment operator. They must also have all relevant attributes (such as name, price and quantity) and functions (such as calculate sum) as presented in the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

III. Results



```python
import copy
import gc   # for forcing garbage collection

class Fruit:
    def __init__(self, name, price, quantity):
        print(f"Constructor called for Fruit: {name}")
        self.name = name
        self.price = price
        self.quantity = quantity

    def __del__(self):
        print(f"Destructor called for Fruit: {self.name}")

    def __copy__(self):
        print(f"Copy constructor called for Fruit: {self.name}")
        return type(self)(self.name, self.price, self.quantity)

    def copy_assign(self, other):
        print(f"Copy assignment operator called for Fruit: {self.name}")
        self.name = other.name
        self.price = other.price
        self.quantity = other.quantity

    def calculate_sum(self):
        return self.price * self.quantity

    def display(self):
        print(f"[Fruit] Name: {self.name}, Price: {self.price}, "
              f"Quantity: {self.quantity}, Total: {self.calculate_sum()}")


class Vegetable:
    def __init__(self, name, price, quantity):
        print(f"Constructor called for Vegetable: {name}")
        self.name = name
        self.price = price
        self.quantity = quantity

    def __del__(self):
        print(f"Destructor called for Vegetable: {self.name}")

    def __copy__(self):
        print(f"Copy constructor called for Vegetable: {self.name}")
        return type(self)(self.name, self.price, self.quantity)

    def copy_assign(self, other):
        print(f"Copy assignment operator called for Vegetable: {self.name}")
        self.name = other.name
        self.price = other.price
        self.quantity = other.quantity

    def calculate_sum(self):
        return self.price * self.quantity

    def display(self):
        print(f"[Vegetable] Name: {self.name}, Price: {self.price}, "
              f"Quantity: {self.quantity}, Total: {self.calculate_sum()}")
```

```python
# Problem 3: Total sum of all grocery items
def TotalSum(grocery_list):
    total = 0
    for item in grocery_list:
        total += item.calculate_sum()
    return total


# Driver code
if __name__ == "__main__":
    # Problem 1: Creating Fruit and Vegetable objects
    Apple = Fruit("Apple", 10, 7)
    Banana = Fruit("Banana", 10, 8)
    Broccoli = Vegetable("Broccoli", 60, 12)
    Lettuce = Vegetable("Lettuce", 50, 10)

    # Problem 2: GroceryList array
    GroceryList = [Apple, Banana, Broccoli, Lettuce]

    print("\nGrocery List Details")
    for item in GroceryList:
        item.display()

    # Problem 3: Calculate total sum
    print("\nTotal Sum:", TotalSum(GroceryList))

    # Problem 4: Delete Lettuce
    print("\nActions: Deleting Lettuce from GroceryList")
    for i, item in enumerate(GroceryList):
        if item.name == "Lettuce":
            del GroceryList[i]   # removes from list
            break

    del Lettuce        # remove variable reference
    gc.collect()       # force garbage collector to call destructor

    print("\nGrocery List After Deletion:")
    for item in GroceryList:
        item.display()

    print("\nNew Total Sum:", TotalSum(GroceryList))
```

```
Constructor called for Fruit: Apple
Constructor called for Fruit: Banana
Constructor called for Vegetable: Broccoli
Constructor called for Vegetable: Lettuce

Grocery List Details
[Fruit] Name: Apple, Price: 10, Quantity: 7, Total: 70
[Fruit] Name: Banana, Price: 10, Quantity: 8, Total: 80
[Vegetable] Name: Broccoli, Price: 60, Quantity: 12, Total: 720
[Vegetable] Name: Lettuce, Price: 50, Quantity: 10, Total: 500

Total Sum: 1370

Actions: Deleting Lettuce from GroceryList

Grocery List After Deletion:
Destructor called for Vegetable: Lettuce
[Fruit] Name: Apple, Price: 10, Quantity: 7, Total: 70
[Fruit] Name: Banana, Price: 10, Quantity: 8, Total: 80
[Vegetable] Name: Broccoli, Price: 60, Quantity: 12, Total: 720

New Total Sum: 870
```

Figure 1: Screenshot of program

As seen in figure 1, my code works by defining two separate classes, one for fruits and one for vegetables, each with its own constructor, copy constructor, assignment, and destructor so I could clearly see what happens with the objects when running it. I gave each object attributes like name, price, and quantity, along with functions to calculate totals. All the items are stored in a grocery list, and I loop through it to display the details and get the overall sum. I also used gc.collect() after deleting items to make sure destructors get called properly. On the first run, everything works as expected, with constructors showing when items are created and destructors when they are deleted. The unusual part comes on the second run, where destructors from the previous objects appear before the new output. This happens because Colab keeps old objects in memory until the next execution, so when I rerun the cell those leftover objects are finally destroyed. It looks strange at first, but it's simply how Colab handles memory across runs.

IV. Conclusion

In conclusion, this lab gave us hands-on practice with object-oriented programming by building separate fruit and vegetable classes, handling constructors, destructors, and using garbage collection to see how memory is managed. It also showed how the coding environment behaves differently between runs, which is a good reminder that system-level behavior can affect our results. Overall, it tied the concepts together in a way that feels practical for me as a computer engineering student.

References

[1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.