

Activity No. 2.1	
ARRAYS, POINTERS AND DYNAMIC MEMORY ALLOCATION	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: September 11, 2024
Section: CPE 21S4	Date Submitted: September 12, 2024
Name(s): Virtucio, Dominic Joseph	Instructor: Ma'am Maria Rizette Sayo

6. Output

Screenshot

```

1 #include <iostream>
2 #include <string>
3
4 class Student {
5 private:
6     std::string studentName;
7     int studentAge;
8
9 public:
10    // Constructor
11    Student(std::string newName = "John Doe", int newAge = 18) {
12        studentName = std::move(newName);
13        studentAge = newAge;
14        std::cout << "Constructor Called." << std::endl;
15    }
16
17    // Destructor
18    ~Student() {
19        std::cout << "Destructor Called." << std::endl;
20    }
21
22    // Copy Constructor
23    Student(const Student &copyStudent) {
24        std::cout << "Copy Constructor Called" << std::endl;
25        studentName = copyStudent.studentName;
26        studentAge = copyStudent.studentAge;
27    }
28
29    // Copy Assignment Operator
30    Student& operator=(const Student& copy) {
31        std::cout << "Copy Assignment Operator Called." << std::endl;
32        if (this == &copy) {
33            return *this;
34        }
35        studentName = copy.studentName;
36        studentAge = copy.studentAge;
37        return *this;
38    }
39
40    // Display attributes
41    void printDetails() {
42        std::cout << studentName << " " << studentAge << std::endl;
43    }
44 };
45
46 // Initial Driver Program
47 int main() {
48     Student student1("Roman", 28);
49     Student student2(student1);
50     Student student3;
51     student3 = student2;
52
53     return 0;
54 }
55

```

/tmp/LEauf50NnA.o
Constructor Called.
Copy Constructor Called
Constructor Called.
Copy Assignment Operator Called.
Destructor Called.
Destructor Called.
Destructor Called.

=== Code Execution Successful ===

Observation	The initial driver application uses static memory allocation to create three Student objects (student1, student2, and student3). It successfully invokes the copy constructor when student2 is started from student1 and the copy assignment operator when student3 is assigned to student2, with constructor and destructor messages indicating appropriate memory allocation and deallocation.
-------------	--

Table 2-1. Initial Driver Program

Screenshot

```

1 #include <iostream>
2 #include <string>
3
4 class Student {
5 private:
6     std::string studentName;
7     int studentAge;
8
9 public:
10    // Constructor
11    Student(std::string newName = "John Doe", int newAge = 18) {
12        studentName = std::move(newName);
13        studentAge = newAge;
14        std::cout << "Constructor Called." << std::endl;
15    }
16
17    // Destructor
18    ~Student() {
19        std::cout << "Destructor Called." << std::endl;
20    }
21
22    // Copy Constructor
23    Student(const Student &copyStudent) {
24        std::cout << "Copy Constructor Called" << std::endl;
25        studentName = copyStudent.studentName;
26        studentAge = copyStudent.studentAge;
27    }
28
29    // Copy Assignment Operator
30    Student& operator=(const Student& copy) {
31        std::cout << "Copy Assignment Operator Called." << std::endl;
32        if (this == &copy) {
33            return *this;
34        }
35        studentName = copy.studentName;
36        studentAge = copy.studentAge;
37        return *this;
38    }
39
40    // Display attributes
41    void printDetails() {
42        std::cout << studentName << " " << studentAge << std::endl;
43    }
44 };
45
46 // Modified Driver Program with Student Lists
47 int main() {
48     const size_t j = 5;
49
50     Student studentList[j] = {}; // Create an array of 5 Student objects
51     std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"};
52     int ageList[j] = {15, 16, 18, 19, 16};
53
54     return 0;
55 }
56

```

```
/tmp/5QQFJupunG.o
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

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

Observation

In the modified driver program, I create an array of 5 Student objects (studentList[]) along with two supporting arrays: namesList[] and ageList[], which store the names and ages of the students. At this stage, I haven't performed any operations on the studentList array; it's merely initialized along with the names and ages arrays. No output is generated yet because there are no function calls in the main() function to interact with the student data, setting the stage for dynamic memory allocation in the upcoming steps.

Table 2-2. Modified Driver Program with Student Lists

Loop A

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

Observation

In this loop, I dynamically generate a 'Student' object for each index, populating it with names and ages from the lists before copying it into the 'studentList' array. To minimize memory leaks, I remove

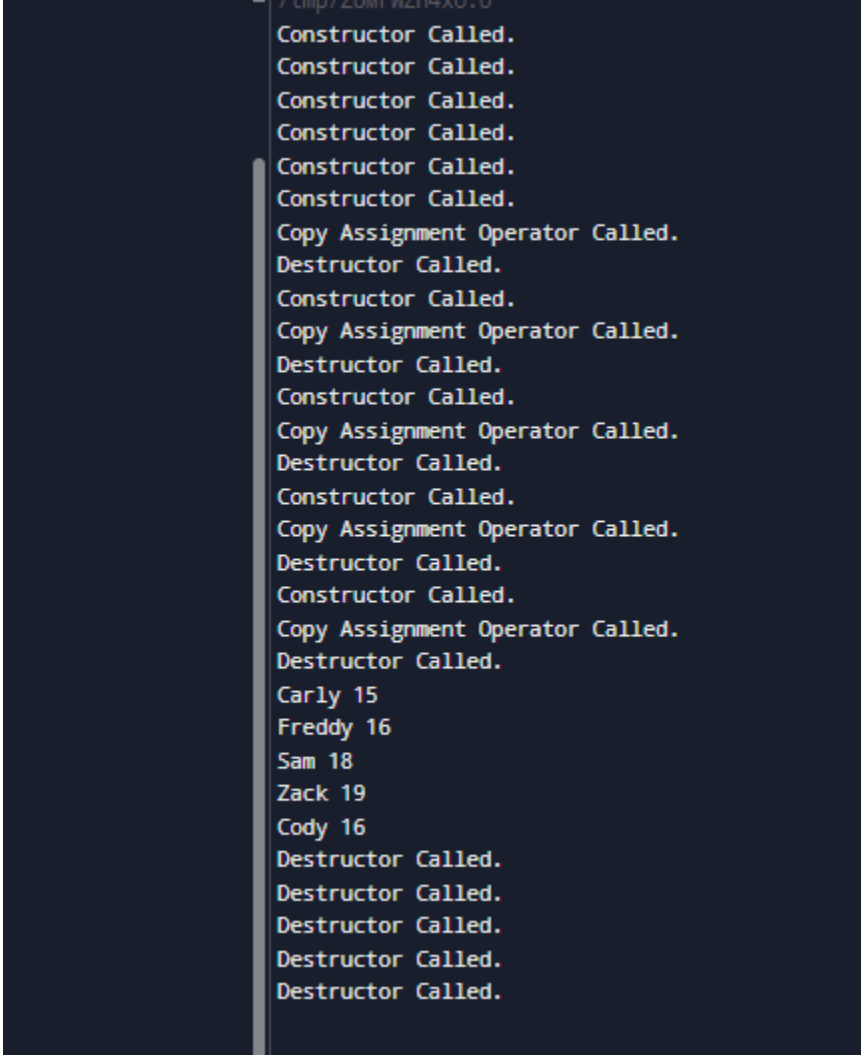
	the dynamically constructed 'Student' object immediately after adding it to the array. This solution works, however it creates and destroys temporary objects that may be avoided by directly generating the 'Student' objects in the 'studentList' array.
Loop B	<pre>for (int i = 0; i < j; i++) { studentList[i].printDetails(); }</pre>
Observation	In this loop, I go through the 'studentList' array, calling 'printDetails()' on each 'Student' object to show its properties. This quickly prints out the information for each student, ensuring that the objects are appropriately populated and placed in the array. The loop works as intended, offering a simple way to validate the accuracy of the 'Student' objects.
Output	 <pre>Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called. Copy Assignment Operator Called. Destructor Called. Constructor Called. Copy Assignment Operator Called. Destructor Called. Constructor Called. Copy Assignment Operator Called. Destructor Called. Constructor Called. Copy Assignment Operator Called. Destructor Called. Constructor Called. Copy Assignment Operator Called. Destructor Called. Carly 15 Freddy 16 Sam 18 Zack 19 Cody 16 Destructor Called. Destructor Called. Destructor Called. Destructor Called. Destructor Called.</pre>
Observation	The output shows the sequence of constructor, copy assignment operator, and destructor calls during the program's execution. Initially, each 'Student' object is created using the constructor, but it appears that the copy assignment operator is called numerous times, indicating that some 'Student' instances are being copied or reassigned. Finally, the destructor appropriately destroys each 'Student' object, indicating that memory management via smart pointers is working as planned and that the program is cleaning up all allocated resources.

Table 2-3. Final Driver Program

Modifications

```
8 int studentAge;
9
10 public:
11     Student(std::string newName = "John Doe", int newAge = 18) {
12         studentName = std::move(newName);
13         studentAge = newAge;
14         std::cout << "Constructor Called." << std::endl;
15     }
16
17     ~Student() {
18         std::cout << "Destructor Called." << std::endl;
19     }
20
21     Student(const Student &copyStudent) {
22         std::cout << "Copy Constructor Called" << std::endl;
23         studentName = copyStudent.studentName;
24         studentAge = copyStudent.studentAge;
25     }
26
27     Student& operator=(const Student& copy) {
28         std::cout << "Copy Assignment Operator Called." << std::endl;
29         if (this == &copy) {
30             return *this;
31         }
32         studentName = copy.studentName;
33         studentAge = copy.studentAge;
34         return *this;
35     }
36
37     void printDetails() {
38         std::cout << studentName << " " << studentAge << std::endl;
39     }
40 };
41
42 int main() {
43     const size_t j = 5;
44     std::unique_ptr<Student[]> studentList(new Student[j]);
45     std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"};
46     int ageList[j] = {15, 16, 18, 19, 16};
47
48     for (int i = 0; i < j; i++) {
49         studentList[i] = Student(namesList[i], ageList[i]);
50     }
51
52     for (int i = 0; i < j; i++) {
53         studentList[i].printDetails();
54     }
55
56     return 0;
57 }
58
```

/tmp/R35r2d64JR.o
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Copy Assignment Operator Called.
Destructor Called.
Constructor Called.
Copy Assignment Operator Called.
Destructor Called.
Constructor Called.
Copy Assignment Operator Called.
Destructor Called.
Copy Assignment Operator Called.
Destructor Called.
Constructor Called.
Copy Assignment Operator Called.
Destructor 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

The output shows the sequence of constructor, copy assignment operator, and destructor calls during the program's execution. Initially, each 'Student' object is created using the constructor, but it appears that the copy assignment operator is called numerous times, indicating that some 'Student' instances are being copied or reassigned. Finally, the destructor appropriately destroys each 'Student' object, indicating that memory management via smart pointers is working as planned and that the program is cleaning up all allocated resources.

Table 2-4. Modifications/Corrections Necessary

7. Supplementary Activity

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.

```

1 #include <iostream>
2 using namespace std;
3
4 class Item {
5 public:
6     string name;
7     double price;
8     int quantity;
9
10    Item(string n, double p, int q) : name(n), price(p), quantity(q) {}
11
12    double calculateSum() {
13        return price * quantity;
14    }
15
16    void display() {
17        cout << name << ": PHP " << price << " x" << quantity << " = PHP " << calculateSum() << endl;
18    }
19 };
20
21 int main() {
22     Item apple("Apple", 10, 7);
23     Item banana("Banana", 10, 8);
24     Item broccoli("Broccoli", 60, 12);
25     Item lettuce("Lettuce", 50, 10);
26
27     apple.display();
28     banana.display();
29     broccoli.display();
30     lettuce.display();
31
32     double totalSum = apple.calculateSum() + banana.calculateSum() + broccoli.calculateSum() + lettuce
        .calculateSum();
33     cout << "Total: PHP " << totalSum << endl;
34
35     return 0;
36 }
37

```

```

/tmp/qybtuuSmft.o
Apple: PHP 10 x7 = PHP 70
Banana: PHP 10 x8 = PHP 80
Broccoli: PHP 60 x12 = PHP 720
Lettuce: PHP 50 x10 = PHP 500
Total: PHP 1370

```

```

=== Code Execution Successful ===

```

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.

```

1 #include <iostream>
2 using namespace std;
3
4 class Item {
5 public:
6     string name;
7     double price;
8     int quantity;
9
10    Item(string n, double p, int q) : name(n), price(p), quantity(q) {}
11
12    double calculateSum() {
13        return price * quantity;
14    }
15
16    void display() {
17        cout << name << ": PHP " << price << " x" << quantity << " = PHP " << calculateSum() << endl;
18    }
19 };
20
21 int main() {
22     Item groceryList[] = {
23         Item("Apple", 10, 7),
24         Item("Banana", 10, 8),
25         Item("Broccoli", 60, 12),
26         Item("Lettuce", 50, 10)
27     };
28
29     double totalSum = 0;
30     for (int i = 0; i < 4; ++i) {
31         groceryList[i].display();
32         totalSum += groceryList[i].calculateSum();
33     }
34
35     cout << "Total: PHP " << totalSum << endl;
36
37     return 0;
38 }
39

```

/tmp/RteAEdMGnR.o

Apple: PHP 10 x7 = PHP 70
Banana: PHP 10 x8 = PHP 80
Broccoli: PHP 60 x12 = PHP 720
Lettuce: PHP 50 x10 = PHP 500
Total: PHP 1370

=== Code Execution Successful ===

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

```
1 #include <iostream>
2 using namespace std;
3
4 class Item {
5 public:
6     string name;
7     double price;
8     int quantity;
9
10    Item(string n, double p, int q) : name(n), price(p), quantity(q) {}
11
12    double calculateSum() {
13        return price * quantity;
14    }
15    void display() {
16        cout << name << ": PHP " << price << " x" << quantity << " = PHP " << calculateSum() << endl;
17    }
18 };
19 double TotalSum(Item groceryList[], int size) {
20     double totalSum = 0;
21     for (int i = 0; i < size; ++i) {
22         totalSum += groceryList[i].calculateSum();
23     }
24     return totalSum;
25 }
26 int main() {
27     Item groceryList[] = {
28         Item("Apple", 10, 7),
29         Item("Banana", 10, 8),
30         Item("Broccoli", 60, 12),
31         Item("Lettuce", 50, 10)
32     };
33     int size = sizeof(groceryList) / sizeof(groceryList[0]);
34
35     for (int i = 0; i < size; ++i) {
36         groceryList[i].display();
37     }
38     double totalSum = TotalSum(groceryList, size);
39     cout << "Total: PHP " << totalSum << endl;
40
41     return 0;
42 }
43
```

/tmp/RteAEdMGnR.o

Apple: PHP 10 x7 = PHP 70
Banana: PHP 10 x8 = PHP 80
Broccoli: PHP 60 x12 = PHP 720
Lettuce: PHP 50 x10 = PHP 500
Total: PHP 1370

=== Code Execution Successful ===

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

```

11 Item(string n, double p, int q): name(n), price(p), quantity(q) {}
12
13 double calculateSum() { return price * quantity; }
14
15 void display() { cout << name << " PHP " << price << " x " << quantity << " = PHP " << calculateSum() << endl
    ; }
16 };
17
18 double TotalSum(Item* groceryList[], int size) {
19     double totalSum = 0;
20     for (int i = 0; i < size; ++i) totalSum += groceryList[i]->calculateSum();
21     return totalSum;
22 }
23
24 void deleteItem(Item* groceryList[], int& size, string itemName) {
25     for (int i = 0; i < size; ++i) {
26         if (groceryList[i]->name == itemName) {
27             delete groceryList[i];
28             for (int j = i; j < size - 1; ++j) groceryList[j] = groceryList[j + 1];
29             --size; break;
30         }
31     }
32 }
33
34 int main() {
35     int size = 4;
36     Item* groceryList[] = {
37         new Item("Apple", 10, 7), new Item("Banana", 10, 8),
38         new Item("Broccoli", 60, 12), new Item("Lettuce", 50, 10)
39     };
40
41     for (int i = 0; i < size; ++i) groceryList[i]->display();
42     deleteItem(groceryList, size, "Lettuce");
43     cout << "\nAfter deletion:\n";
44     for (int i = 0; i < size; ++i) groceryList[i]->display();
45     cout << "\nTotal: PHP " << TotalSum(groceryList, size) << endl;
46     for (int i = 0; i < size; ++i) delete groceryList[i];
47
48     return 0;
49 }
50

```

/tmp/98JT8uqNEZ.o
 Apple PHP 10 x 7 = PHP 70
 Banana PHP 10 x 8 = PHP 80
 Broccoli PHP 60 x 12 = PHP 720
 Lettuce PHP 50 x 10 = PHP 500

 After deletion:
 Apple PHP 10 x 7 = PHP 70
 Banana PHP 10 x 8 = PHP 80
 Broccoli PHP 60 x 12 = PHP 720

 Total: PHP 870

 === Code Execution Successful ===

8. Conclusion

In this exercise, I was able to effectively allocate both static and dynamic memories by using arrays & pointers. At the same time, I also created and managed dynamically allocated objects while making sure that the new and delete operators that are used to aid memory management are responsible. My cognition of the real functioning of heap memory and the importance of memory leak prevention has grown.

9. Assessment Rubric