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:** 07 - 31 - 25
Section: CPE21S4	**Date Submitted:** 07 - 31 - 25
Name(s): Santiago, David Owen A.	**Instructor:** Engr. Jimlord Quejado

6. Output

Discussion:

A. Variables

Reference Operator (&)

```
1  // Online C++ compiler to run C++ program online
2  #include <iostream>
3
4  int main() {
5
6      int x = 10;
7      std::cout << x << std::endl;
8      std::cout << &x << std::endl << std::endl;
```

```
Output

10
0x7ffe1820772c
```

Dereference Operator (*)

```
1  // Online C++ compiler to run C++ program online
2  #include <iostream>
3
4  int main() {
5
6      int y = 10;
7      std::cout << y << std::endl;
8      std::cout << &y << std::endl;
9      std::cout << *&y << std::endl;
10
11     return 0;
12 }
```

```
Output

10
0x7ffe1c03a02c
10
```

B. Pointers

```cpp
// Online C++ compiler to run C++ program online
#include <iostream>

int main() {

    int pointee = 10;
    int *pointer = &pointee;

    int *point = nullptr;

    std::cout << pointee << std::endl;
    std::cout << *pointer << std::endl;
    std::cout << &pointee << std::endl;
    std::cout<< *point << std::endl;

    return 0;
}
```

```
10
10
0x7fff803c018c
Segmentation fault
```

C. Dynamic Memory Allocation

Arrays

main.cpp Run Output Clear

```cpp
#include <iostream>

int main() {

    int array[] = {1, 2, 3, 4};
    int *ptrArray;
    array = ptrArray; //compiler error
    //ptrArray = array; //no errors

}
```

```
ERROR!
/tmp/XO9qbkV9rb/main.cpp: In function 'int main()':
/tmp/XO9qbkV9rb/main.cpp:7:11: error: incompatible
    types in assignment of 'int*' to 'int [4]'
    7 |     array = ptrArray; //compiler error
      |     ~~~~~~^~~~~~~~~~


=== Code Exited With Errors ===
```

main.cpp Run Output

```cpp
#include <iostream>

int main() {

    int array[] = {1, 2, 3, 4};
    int *ptrArray;
    //array = ptrArray; //compiler error
    ptrArray = array; //no errors

}
```

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

D. Pointers with Objects/Classes

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

class Student{
    public:
    std::string name1;
    Student(std::string name = "John Doe"){ name1
        = name;
    }
};

int main() {

    Student *a = new Student("John");

    std::cout << (*a).name1 << std::endl;
    std::cout << a->name1 << std::endl;

    return 0;
}
```

```
John
John


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

Destructors

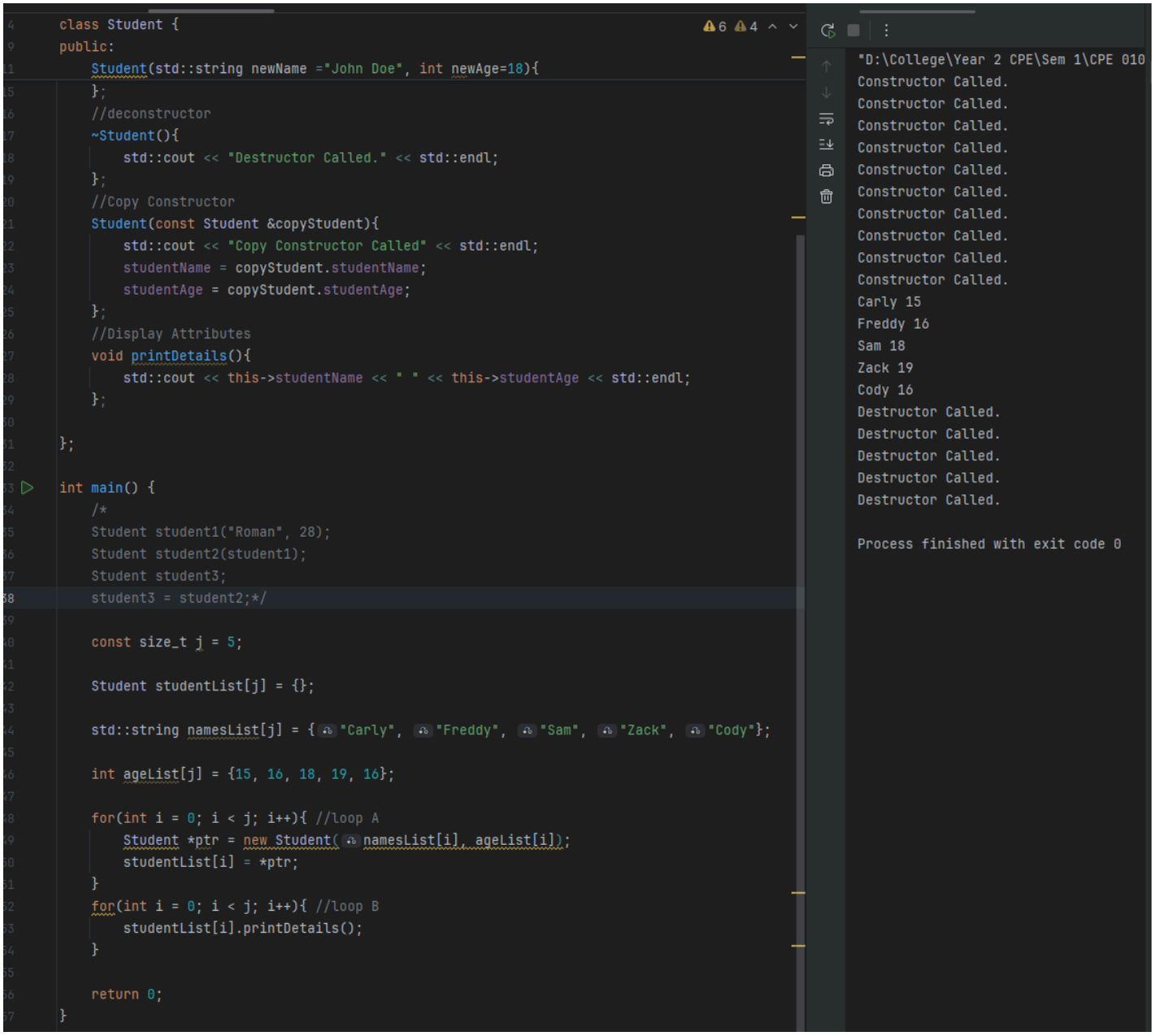
Table 2-1. Initial Driver Program

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

class Student {
private:
    std::string studentName;
    int studentAge;

public:
    //constructor
    Student(std::string newName ="John Doe", int newAge=18){
        studentName = std::move(newName);
        studentAge = newAge;
        std::cout << "Constructor Called." << std::endl;
    };
    //deconstructor
    ~Student(){
        std::cout << "Destructor Called." << std::endl;
    };
    //Copy Constructor
    Student(const Student &copyStudent){
        std::cout << "Copy Constructor Called" << std::endl;
        studentName = copyStudent.studentName;
        studentAge = copyStudent.studentAge;
    };
    //Display Attributes
    void printDetails(){
        std::cout << this->studentName << " " << this->studentAge << std::endl;
    };

};

int main() {

    Student student1( newName: "Roman", newAge: 28);
    Student student2( student1);
    Student student3;
    student3 = student2;

    return 0;
}
```

```
"D:\College\Year 2 CPE\Sem 1\CPE 010
Constructor Called.
Copy Constructor Called
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.

Process finished with exit code 0
```

This code demonstrates the use of the constructor, copy constructor, and destructor. Whenever a student object is created, it calls the constructor function in the public members of the Student class. The first instance calls this, which also prints the “Constructor Called” line. When another student object is called with an existing student object argument, the copy constructor is called. Once all of these are done, the destructor is called to reset the memory.

Table 2-2. Modified Driver Program with Student Lists



```
class Student {
public:
    Student(std::string newName ="John Doe", int newAge=18){
    };
    //deconstructor
    ~Student(){
        std::cout << "Destructor Called." << std::endl;
    };
    //Copy Constructor
    Student(const Student &copyStudent){
        std::cout << "Copy Constructor Called" << std::endl;
        studentName = copyStudent.studentName;
        studentAge = copyStudent.studentAge;
    };
    //Display Attributes
    void printDetails(){
        std::cout << this->studentName << " " << this->studentAge << std::endl;
    };

};

int main() {
    /*
    Student student1("Roman", 28);
    Student student2(student1);
    Student student3;
    student3 = student2;*/

    const size_t j = 5;

    Student studentList[j] = {};

    std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"};

    int ageList[j] = {15, 16, 18, 19, 16};

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

    return 0;
}
```

```
"D:\College\Year 2 CPE\Sem 1\CPE 010
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.

Process finished with exit code 0
```

This code now utilizes a student list array to assign the names and ages of each student object. It uses a for loop to assign the appropriate student name and student age using a pointer. The second loop is used to call the printdetails function for each student to print out the assigned age per name.

Table 2-3. Final Driver Program

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

class Student {
private:
    std::string studentName;
    int studentAge;

public:
    //constructor
    Student(std::string newName ="John Doe", int newAge=18){
        studentName = std::move(newName);
        studentAge = newAge;
        std::cout << "Constructor Called." << std::endl;
    };
    //deconstructor
    ~Student(){
        std::cout << "Destructor Called." << std::endl;
    };
    //Copy Constructor
    Student(const Student &copyStudent){
        std::cout << "Copy Constructor Called" << std::endl;
        studentName = copyStudent.studentName;
        studentAge = copyStudent.studentAge;
    };
    //Display Attributes
    void printDetails(){
        std::cout << this->studentName << " " << this->studentAge << std::endl;
    };

};

int main() {
    const size_t j = 5;

    Student studentList[j] = {};
    std::string namesList[j] = { "Carly", "Freddy", "Sam", "Zack", "Cody"};
    int ageList[j] = {15, 16, 18, 19, 16};

    std::cout<<std::endl;
    for(int i = 0; i < j; i++){ //loop A
        Student *ptr = new Student( namesList[i], ageList[i]);
        studentList[i] = *ptr;
    }
    std::cout<<std::endl;
    for(int i = 0; i < j; i++){ //loop B
        studentList[i].printDetails();
    }
    std::cout<<std::endl;

    return 0;
}
```

LOOP A:	LOOP B:
Constructor Called. Constructor Called. Constructor Called. Constructor Called. Constructor Called.	Carly 15 Freddy 16 Sam 18 Zack 19 Cody 16
The constructors are called because each iteration creates a new student object that points toward the specific names and age in the previously created arrays.	The new student objects are now printed using the printdetails function, again, by iteration.

Table 2.4

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

class Student {
private:
    std::string studentName;
    int studentAge;

public:
    //constructor
    Student(std::string newName ="John Doe", int newAge=18){
        studentName = std::move(newName);
        studentAge = newAge;
        std::cout << "Constructor Called." << std::endl;
    };
    //deconstructor
    ~Student(){
        std::cout << "Destructor Called." << std::endl;
    };
    //Copy Constructor
    Student(const Student &copyStudent){
        std::cout << "Copy Constructor Called" << std::endl;
        studentName = copyStudent.studentName;
        studentAge = copyStudent.studentAge;
    };
    //Display Attributes
    void printDetails(){
        std::cout << this->studentName << " " << this->studentAge << std::endl;
    };

};

int main() {
    const size_t j = 5;

    std::cout << "--Student Names and Ages--\n";
    Student studentList[j] = {};
    std::string namesList[j] = { "Carly",  "Freddy",  "Sam",  "Zack",  "Cody"};
    int ageList[j] = {15, 16, 18, 19, 16};

    std::cout<<"\n--Assigning Names to Age (And Calling Copy Constructor)--"<<std::endl;
    for(int i = 0; i < j; i++){ //loop A
        Student *ptr = new Student( namesList[i], ageList[i]);
        studentList[i] = *ptr;
        Student studentCopy(Student(studentList[i]));
    }

    std::cout<<"\n--Printing Each Student and Age--"<<std::endl;
    for(int i = 0; i < j; i++){ //loop B
        studentList[i].printDetails();
    }
    std::cout<<std::endl;

    return 0;
}
```

```
"D:\College\Year 2 CPE\Sem 1\CPE 010\main1.exe"
--Student Names and Ages--
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.

--Assigning Names to Age (And Calling Copy Constructor)--
Constructor Called.
Copy Constructor Called
Destructor Called.
Constructor Called.
Copy Constructor Called
Destructor Called.
Constructor Called.
Copy Constructor Called
Destructor Called.
Constructor Called.
Copy Constructor Called
Destructor Called.
Constructor Called.
Copy Constructor Called
Destructor Called.

--Printing Each Student and Age--
Carly 15
Freddy 16
Sam 18
Zack 19
Cody 16

Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

In this finalized program, I included an instance of the copy constructor to verify that the copy constructor is working as intended. The program calls the constructors when creating the two arrays. The Loop A calls new constructors that create new student objects that pass arguments—which are pointing toward the names and lists in the array to make sure that each name and age match. The last loop prints these newly created objects—names with matching ages. Once the program is finished utilizing these objects, the program calls the destructors to preserve and allocate memory.

7. Supplementary Activity

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

```cpp
// Santiago CPE21S4
// Problem 1
class GroceryItem {
private:
   std::string itemName;
   float price;
   int quantity;

public:
   // Constructor
   GroceryItem(std::string name = "None", float i = 0.0, int j = 0) {
       itemName = std::move(name);
       price = i;
       quantity = j;
       std::cout << "Constructor Called for " << itemName << std::endl;
   }

   // Destructor
   ~GroceryItem() {
       std::cout << "Destructor Called for " << itemName << std::endl;
   }

   // Copy Constructor
   GroceryItem(const GroceryItem& other) {
       itemName = other.itemName;
       price = other.price;
       quantity = other.quantity;
       std::cout << "Copy Constructor Called for " << itemName << std::endl;
   }

   // Assignment Operator
   GroceryItem& operator=(const GroceryItem& other) {
       std::cout << "Assignment Operator Called for " << itemName << std::endl;
       if (this != &other) {
           itemName = other.itemName;
           price = other.price;
           quantity = other.quantity;
       }
       return *this;
   }

   // Calculate total price
   float getTotal() {
       return price * quantity;
   }

   // Display
   void printDetails() {
       std::cout << itemName << ": PHP " << price << " x" << quantity
                 << " = PHP " << getTotal() << std::endl;
   }

   std::string getName() {
       return itemName;
```

```
    }

};

int main() {
    const int size = 4;
    std::string names[size] = {"Apple", "Banana", "Broccoli", "Lettuce"};
    float price[size] = {10, 10, 60, 50};
    int quantity[size] = {7, 8, 12, 10};

    // Problem 2
    GroceryItem groceryList[size] = {};
    for (int i = 0; i < size; i++) {
        GroceryItem* ptr = new GroceryItem(names[i], price[i], quantity[i]);
        groceryList[i] = *ptr;
        delete ptr;
    }

    std::cout << "\n--- Grocery List ---"<<std::endl;
    for (int i = 0; i < size; i++) {
        groceryList[i].printDetails();
    }

    float total = 0;
    for (int i = 0; i < size; i++) {
        total += groceryList[i].getTotal();
    }

    std::cout << "\nTotal Amount: PHP " << total << std::endl;

    // Deleting Lettuce
    for (int i = 0; i < size; i++) {
        if (groceryList[i].getName() == "Lettuce") {
            std::cout << "\nRemoving Lettuce from list."<<std::endl;
            groceryList[i] = GroceryItem();
        }
    }

    std::cout << "\n--- Updated Grocery List ---"<<std::endl;
    for (int i = 0; i < size; i++) {
        if (groceryList[i].getName() != "None") {
            groceryList[i].printDetails();
        }
    }
    float total_2 = 0;
    for (int i = 0; i < size; i++) {
        total_2 += groceryList[i].getTotal();
    }
    std::cout << "\nNew Total Amount: PHP " << total_2 << std::endl;

    std::cout<<"\n";

    return 0;
}
```

```
Assignment Operator Called for None
Destructor Called for Banana
Constructor Called for Broccoli
Assignment Operator Called for None
Destructor Called for Broccoli
Constructor Called for Lettuce
Assignment Operator Called for None
Destructor Called for Lettuce

--- Grocery List ---
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 Amount: PHP 1370

Removing Lettuce from list.
Constructor Called for None
Assignment Operator Called for Lettuce
Destructor Called for None

--- Updated Grocery List ---
Apple: PHP 10 x7 = PHP 70
Banana: PHP 10 x8 = PHP 80
Broccoli: PHP 60 x12 = PHP 720

New Total Amount: PHP 870

Destructor Called for None
Destructor Called for Broccoli
Destructor Called for Banana
Destructor Called for Apple
```

In this activity, I applied the code from the procedure to create a program for the groceries. The class for the grocery items have constructors, destructors, copy constructors, and assignment operators. By default the objects from GroceryItem are named “none,” hence the program first calls the constructors for these. However, once the arrays with the labelled names, prices, and quantities are created, then the constructors will call accordingly. Since the objects are created and values are unchanged, the assignment operator will still, by default, be called to “none.” However, once the value changes—like how the lettuce is removed, then the assignment operator will now be called to the “lettuce.” The sums of the prices of the grocery lists before and after removing lettuce are also included accordingly.

8. Conclusion

In conclusion, this activity explores the basics of constructors, arrays and pointers, and memory allocation. A class is defined with appropriate constructors and destructors for easy creation of objects in the program. These classes include constructors, which save memory and code efficiency when creating objects. Destructors are also introduced—these functions serve as “clean-up”; they are responsible for proper memory management to ensure resources (like memory) are being used efficiently to avoid leaks and for easy tracking of objects for debugging. I also practiced the use of pointers, particularly when assigning the names, prices, and quantities of each object.

9. Assessment Rubric