

Activity No. 3	
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: September 11, 2024
Section: CPE21S4	Date Submitted: September 13, 2024
Name(s): Don Eleazar T. Fernandez	Instructor: Maria Rizette Sayo

6. Output

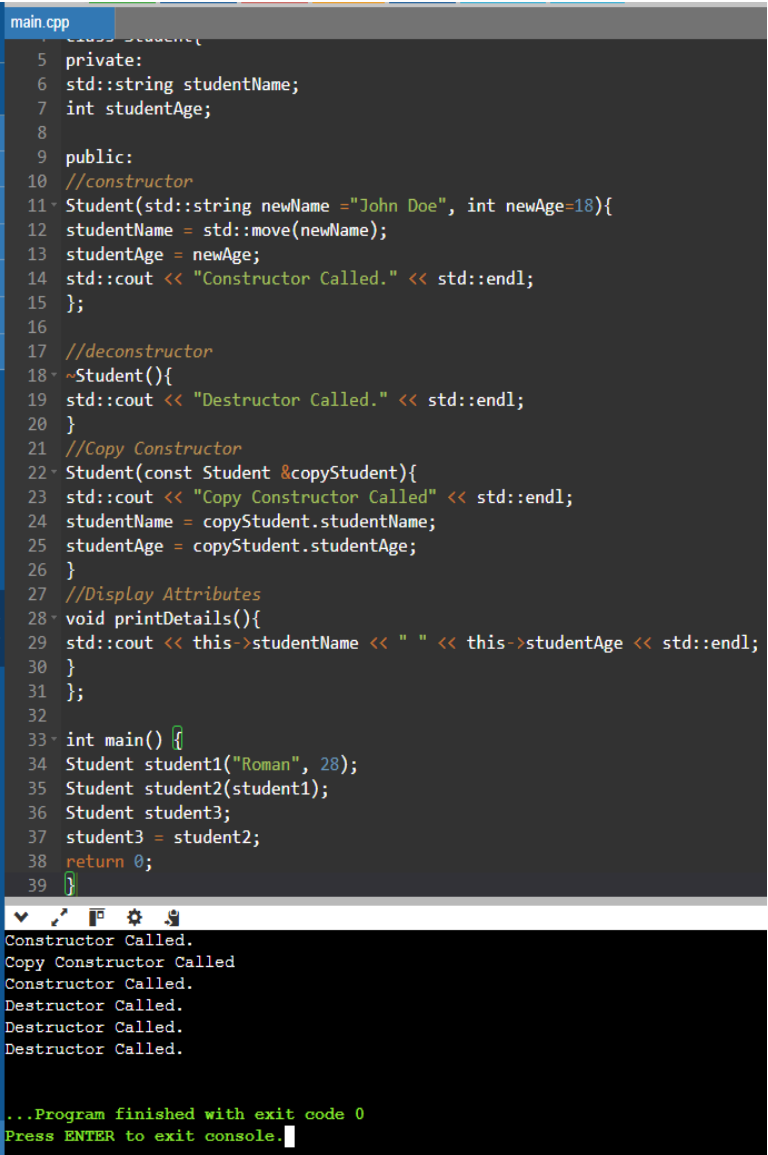
Screenshot	 <pre> main.cpp 1 class Student { 2 private: 3 std::string studentName; 4 int studentAge; 5 public: 6 //constructor 7 Student(std::string newName = "John Doe", int newAge=18){ 8 studentName = std::move(newName); 9 studentAge = newAge; 10 std::cout << "Constructor Called." << std::endl; 11 }; 12 13 //destructor 14 ~Student(){ 15 std::cout << "Destructor Called." << std::endl; 16 } 17 //Copy Constructor 18 Student(const Student &copyStudent){ 19 std::cout << "Copy Constructor Called" << std::endl; 20 studentName = copyStudent.studentName; 21 studentAge = copyStudent.studentAge; 22 } 23 //Display Attributes 24 void printDetails(){ 25 std::cout << this->studentName << " " << this->studentAge << std::endl; 26 } 27 }; 28 29 int main() { 30 Student student1("Roman", 28); 31 Student student2(student1); 32 Student student3; 33 student3 = student2; 34 return 0; 35 } </pre> <pre> Constructor Called. Copy Constructor Called Constructor Called. Destructor Called. Destructor Called. Destructor Called. ...Program finished with exit code 0 Press ENTER to exit console. </pre>
Observation	<ul style="list-style-type: none"> - The constructor initializes the object with the acquired input and passes it on to the parameters, while the destructor removes the object from the class. And the copy constructor actually creates an object by copying the existing object.

Table 2-1. Initial Driver Program

Screenshot

```
main.cpp
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 //Copy Constructor
22 Student(const Student &copyStudent){
23 std::cout << "Copy Constructor Called" << std::endl;
24 studentName = copyStudent.studentName;
25 studentAge = copyStudent.studentAge;
26 }
27 //Display Attributes
28 void printDetails(){
29 std::cout << this->studentName << " " << this->studentAge << std::endl;
30 }
31 };
32
33 int main() {
34 const size_t j = 5;
35 Student studentList[j] = {};
36 std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"};
37 int ageList[j] = {15, 16, 18, 19, 16};
38 15;
39 return 0;
40 }
```

```

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

Observation

- The output reveals that the constructor is called five times, once for each student. Five calls to the destructor demonstrate that each object has been cleaned up when the program finishes. This illustrates how constructors and destructors should be used when managing objects in an array.

Table 2-2. Modified Driver Program with Student Lists

Loop A

```

main.cpp
14 std::cout << "Constructor Called." << std::endl;
15 };
16
17 //destructor
18 ~Student(){
19     std::cout << "Destructor Called." << std::endl;
20 }
21 //Copy Constructor
22 Student(const Student &copyStudent){
23     std::cout << "Copy Constructor Called" << std::endl;
24     studentName = copyStudent.studentName;
25     studentAge = copyStudent.studentAge;
26 }
27 //Display Attributes
28 void printDetails(){
29     std::cout << this->studentName << " " << this->studentAge << std::endl;
30 }
31 };
32
33 int main() {
34     const size_t j = 5;
35     Student studentList[j] = {};
36     std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"};
37     int ageList[j] = {15, 16, 18, 19, 16};
38     for(int i = 0; i < j; i++){ //Loop A
39         Student *ptr = new Student(namesList[i], ageList[i]);
40         studentList[i] = *ptr;
41     }
42     /*for(int i = 0; i < j; i++){ //Loop B
43         studentList[i].printDetails();
44     }*/
45     return 0;
46 }

```

```

Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.

```

Observation

- In Loop A, the constructor is called five times. Though no delete operation was done, each object's memory was not deallocated which resulted in ten. This means that the memory was not released until the program's execution is complete, and the destructors only take effect for the studentList array objects.

Loop B

```
main.cpp
14 std::cout << "Constructor Called." << std::endl;
15 };
16
17 //destructor
18 ~Student(){
19     std::cout << "Destructor Called." << std::endl;
20 }
21 //Copy Constructor
22 Student(const Student &copyStudent){
23     std::cout << "Copy Constructor Called" << std::endl;
24     studentName = copyStudent.studentName;
25     studentAge = copyStudent.studentAge;
26 }
27 //Display Attributes
28 void printDetails(){
29     std::cout << this->studentName << " " << this->studentAge << std::endl;
30 }
31 };
32
33 int main() {
34     const size_t j = 5;
35     Student studentList[j] = {};
36     std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"};
37     int ageList[j] = {15, 16, 18, 19, 16};
38     /*for(int i = 0; i < j; i++){ //Loop A
39         Student *ptr = new Student(namesList[i], ageList[i]);
40         studentList[i] = *ptr;
41     }*/
42     for(int i = 0; i < j; i++){ //Loop B
43         studentList[i].printDetails();
44     }
45     return 0;
46 }
```

```

v  ↗  📄  ⚙  🗑
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
John Doe 18
John Doe 18
John Doe 18
John Doe 18
John Doe 18
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

Observation

- An array of Student objects was formed in the main function, but there is a problem with student initialization, which causes all students to be printed as "John Doe" rather than by their own names.

Output	<pre> main.cpp 18 ~Student(){ 19 std::cout << "Destructor Called." << std::endl; 20 } 21 //Copy Constructor 22 Student(const Student &copyStudent){ 23 std::cout << "Copy Constructor Called" << std::endl; 24 studentName = copyStudent.studentName; 25 studentAge = copyStudent.studentAge; 26 } 27 //Display Attributes 28 void printDetails(){ 29 std::cout << this->studentName << " " << this->studentAge << std::endl; 30 } 31 }; 32 33 int main() { 34 const size_t j = 5; 35 Student studentList[j] = {}; 36 std::string namesList[j] = {"Carly", "Freddy", "Sam", "Zack", "Cody"}; 37 int ageList[j] = {15, 16, 18, 19, 16}; 38 for(int i = 0; i < j; i++){ //Loop A 39 Student *ptr = new Student(namesList[i], ageList[i]); 40 studentList[i] = *ptr; 41 } 42 for(int i = 0; i < j; i++){ //Loop B 43 studentList[i].printDetails(); 44 } 45 return 0; 46 } </pre> <pre> 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. </pre>
Observation	<ul style="list-style-type: none"> - The earlier problem was addressed this time, which properly initializes and shows the name and age of each student. The object initialization and cleanup were successful.

Table 2-3. Final Driver Program

Modification	<pre> 38 for(int i = 0; i < j; i++){ //Loop A 39 Student *ptr = new Student(namesList[i], ageList[i]); 40 studentList[i] = *ptr; 41 } 42 for(int i = 0; i < j; i++){ //Loop B 43 studentList[i].printDetails(); 44 } </pre>
Observation	<ul style="list-style-type: none"> - The program ensures the initialization of the studentList in Loop A. Only the initialized data from the studentList is read by Loop B. In short, Loop B handles data output while Loop A handles object construction.

Table 2-4. Modifications/Corrections Necessary

7. Supplementary Activity

```

#include <iomanip>
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class GroceryItem {
private:
    string name;
    int price;
    int quantity;

public:
    // The constructor
    GroceryItem(const string & name, int price, int quantity)
        : name(name), price(price), quantity(quantity) {}

    // The destructor
    virtual ~GroceryItem() {}

    // The copy constructor
    GroceryItem(const GroceryItem & other)
        : name(other.name), price(other.price), quantity(other.quantity) {}

    // The copy assignment operator
    GroceryItem & operator = (const GroceryItem & other) {
        if (this != &other) {
            name = other.name;
            price = other.price;
            quantity = other.quantity;
        }
        return *this;
    }

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

    void display() const {
        cout << left << setw(10) << name << "PHP " << setw(4) << price << "x" << setw(2) << quantity << endl;
    }

    const string & getName() const {
        return name;
    }
};

// Class for Fruit
class Fruit : public GroceryItem {
public:
    Fruit(const string & name, int price, int quantity)

```

```

        : GroceryItem(name, price, quantity) {}

    ~Fruit() {}
};

// Class for Vegetable
class Vegetable : public GroceryItem {
public:
    Vegetable(const string & name, int price, int quantity)
        : GroceryItem(name, price, quantity) {}

    ~Vegetable() {}
};

// Function to display all items in the list
void displayGroceryList(const vector <GroceryItem*> & groceryList) {
    for (const auto & item : groceryList) {
        item -> display();
    }
}

// Function to calculate the total sum of all items in the list
int totalSum(const vector <GroceryItem*> & groceryList) {
    int sum = 0;
    for (const auto & item : groceryList) {
        sum += item -> calculateSum();
    }
    return sum;
}

// Function to delete an item from the list
void deleteItem(vector <GroceryItem*> & groceryList, const string& itemName) {
    for (auto it = groceryList.begin(); it != groceryList.end(); ++it) {
        if ((*it) -> getName() == itemName) {
            delete *it;
            groceryList.erase(it);
            break;
        }
    }
}

int main() {
    // Problem 2: Create an array GroceryList
    vector <GroceryItem*> groceryList = {
        new Fruit("Apple", 10, 7),
        new Fruit("Banana", 10, 8),
        new Vegetable("Broccoli", 60, 12),
        new Vegetable("Lettuce", 50, 10)
    };
    cout << "Grocery List:\n";
    displayGroceryList(groceryList);
}

```

```

// Problem 3: Calculate the total sum
cout << "\nTotal Sum: PHP " << totalSum(groceryList) << endl;

// Problem 4: Delete Lettuce and deallocate memory
deleteItem(groceryList, "Lettuce");
cout << "\nGrocery List (After the deletion of Lettuce):\n";
displayGroceryList(groceryList);
cout << "\nTotal Sum: PHP " << totalSum(groceryList) << endl;
for (auto & item : groceryList) {
    delete item;
}

return 0;
}

```

8. Conclusion

The activity has discussed the destructor, copy constructor, and copy assignment operator, which developed the use of classes in C++ and furthered the functionality. The task on the procedure was easy for me to perform and understand as everything was laid out. Contrary to the supplementary activity, it was complex to dissect at first, where different constructors were required. The activity gave me the realization that I performed it quite unwell, as I was confused on how I would do it.

9. Assessment Rubric