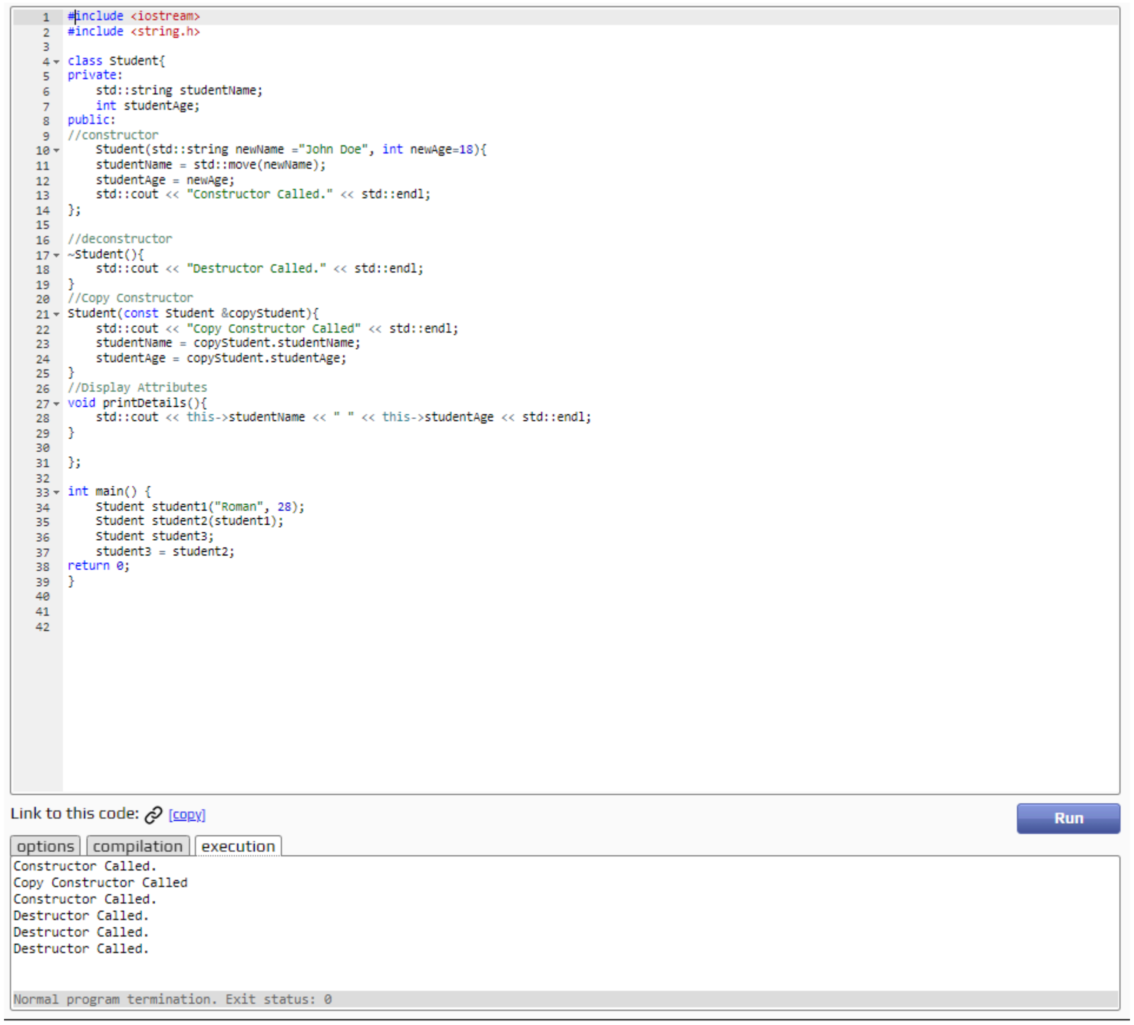
Activity No. 2

ARRAYS POINTER AND DYNAMIC MEMORY ALLOCATION

Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 09/11/24**
Section: CPE21S4	**Date Submitted:**
Name(s): Ma. Kassandra Nicole D. Santos	**Instructor: Ms. Sayo**

6. Output

Screenshot



```
1  #include <iostream>
2  #include <string.h>
3
4  class Student{
5  private:
6      std::string studentName;
7      int studentAge;
8  public:
9  //constructor
10     Student(std::string newName ="John Doe", int newAge=18){
11     studentName = std::move(newName);
12     studentAge = newAge;
13     std::cout << "Constructor Called." << std::endl;
14 };
15
16 //deconstructor
17 ~Student(){
18     std::cout << "Destructor Called." << std::endl;
19 }
20 //Copy Constructor
21 Student(const Student &copyStudent){
22     std::cout << "Copy Constructor Called" << std::endl;
23     studentName = copyStudent.studentName;
24     studentAge = copyStudent.studentAge;
25 }
26 //Display Attributes
27 void printDetails(){
28     std::cout << this->studentName << " " << this->studentAge << std::endl;
29 }
30
31 };
32
33 int main() {
34     Student student1("Roman", 28);
35     Student student2(student1);
36     Student student3;
37     student3 = student2;
38 return 0;
39 }
40
41
42
```

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

Normal program termination. Exit status: 0
```

Observation

Constructor (a special function that is automatically called when an object of a class is created)

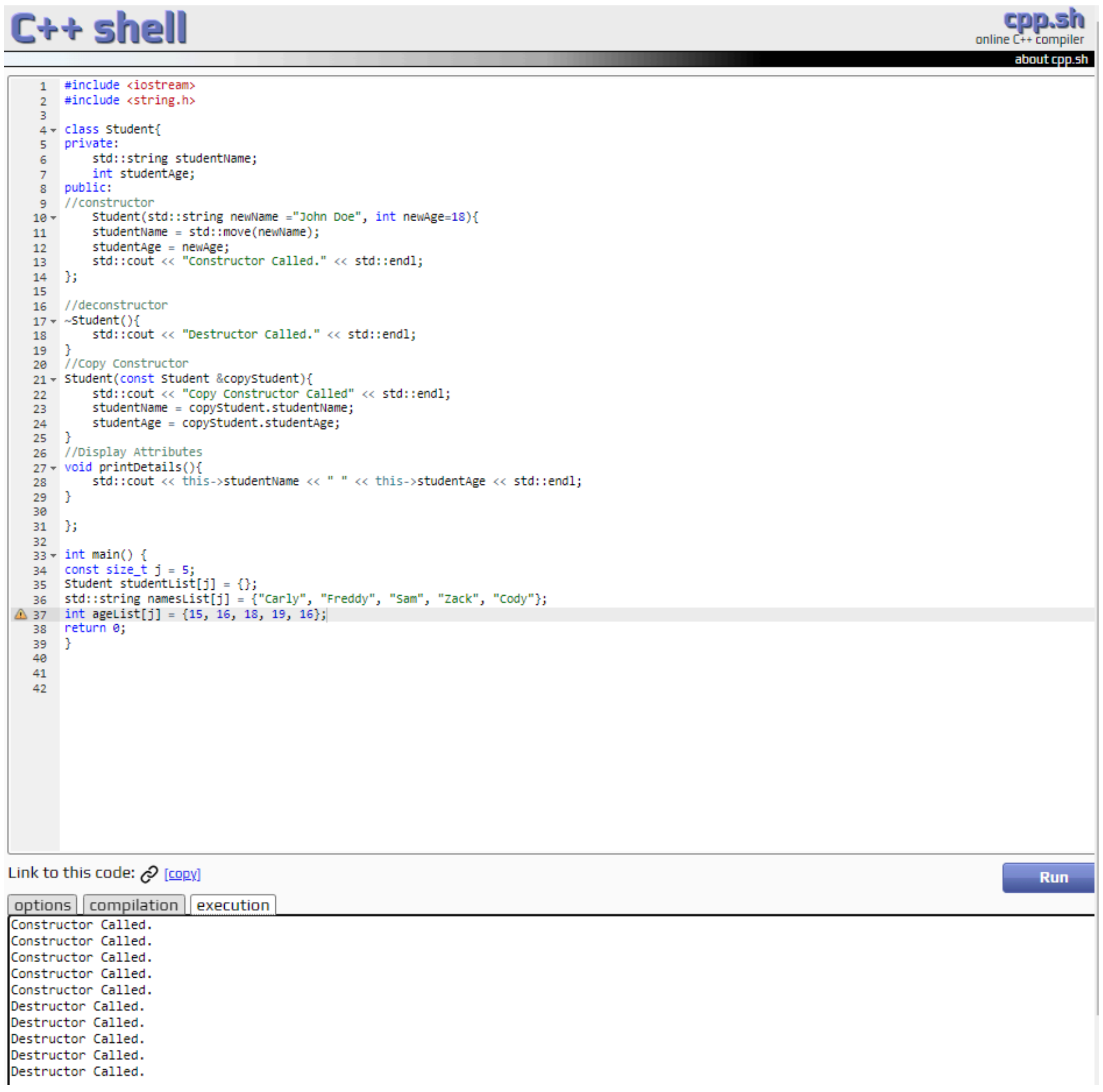
- Example of a constructor class: The student in Line 10
    - the **std::cout << "Constuctor Called." << std::endl;** was called because the constructor called the arguments "Roman" which was initialzed to studentName to "Roman" and the argument 28 was initialized to studentAge. Since there was an constructor that occurred, the program will then be call "Constructor"

Destruction (opposite to the constructor, its purpose is to be automatically called when an object is destroyed)

- If a part of the code goes out to the intended scope at the end of the program, the "Destruction Called" will then be printed.
- We see it at the end of the program at int main () where the destruction was called for

	students3, student2, and student1 as if it cleans it up when an object is destroyed **Copy Constructor** (Creates a new object another copy of an existing constructor) ● In line 21, we see that the studentName and studentAge were once use from an existing Student object which was in line 10. ● In int main () line 35, we that the Student student2(student1) where it copied the information from student 1 to student 2 hence, the output of it will be “Copt Constructor called)

Table 2-1. Initial Driver Program

Screenshot	 C++ shell cpp.sh online C++ compiler about cpp.sh Link to this code: [copy] Run options compilation execution

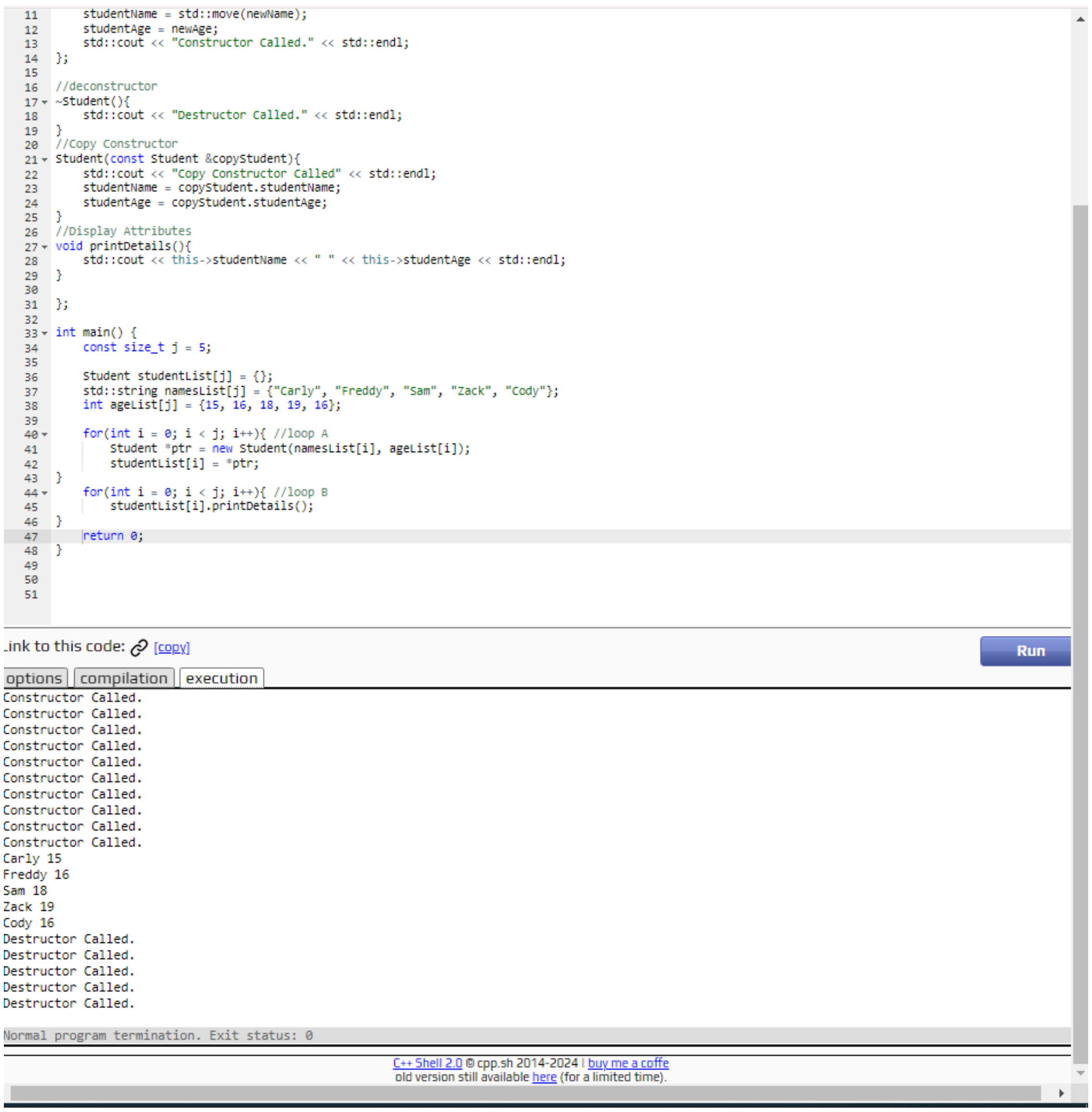
```
1   #include <iostream>
2   #include <string.h>
3
4   class Student{
5   private:
6       std::string studentName;
7       int studentAge;
8   public:
9   //constructor
10      Student(std::string newName ="John Doe", int newAge=18){
11      studentName = std::move(newName);
12      studentAge = newAge;
13      std::cout << "Constructor Called." << std::endl;
14  };
15
16  //deconstructor
17  ~Student(){
18      std::cout << "Destructor Called." << std::endl;
19  }
20  //Copy Constructor
21  Student(const Student &copyStudent){
22      std::cout << "Copy Constructor Called" << std::endl;
23      studentName = copyStudent.studentName;
24      studentAge = copyStudent.studentAge;
25  }
26  //Display Attributes
27  void printDetails(){
28      std::cout << this->studentName << " " << this->studentAge << std::endl;
29  }
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  return 0;
39  }
40
41
42
```

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

Observation	By modifying the program, we have this output: ● The nameList and ageList were defined but noty modified to the studentList therefore the output was not seen when the program was executed ● The constructor was called 5 times one for each element in the arrat with the name John Doe and age 18 ● The destructor was called 5 times when program ends as the arrays were going out of scopre already,

Table 2-2. Modified Driver Program with Students Lists

Loop A



```cpp
    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};

    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;
}
```

Link to this code: [copy]

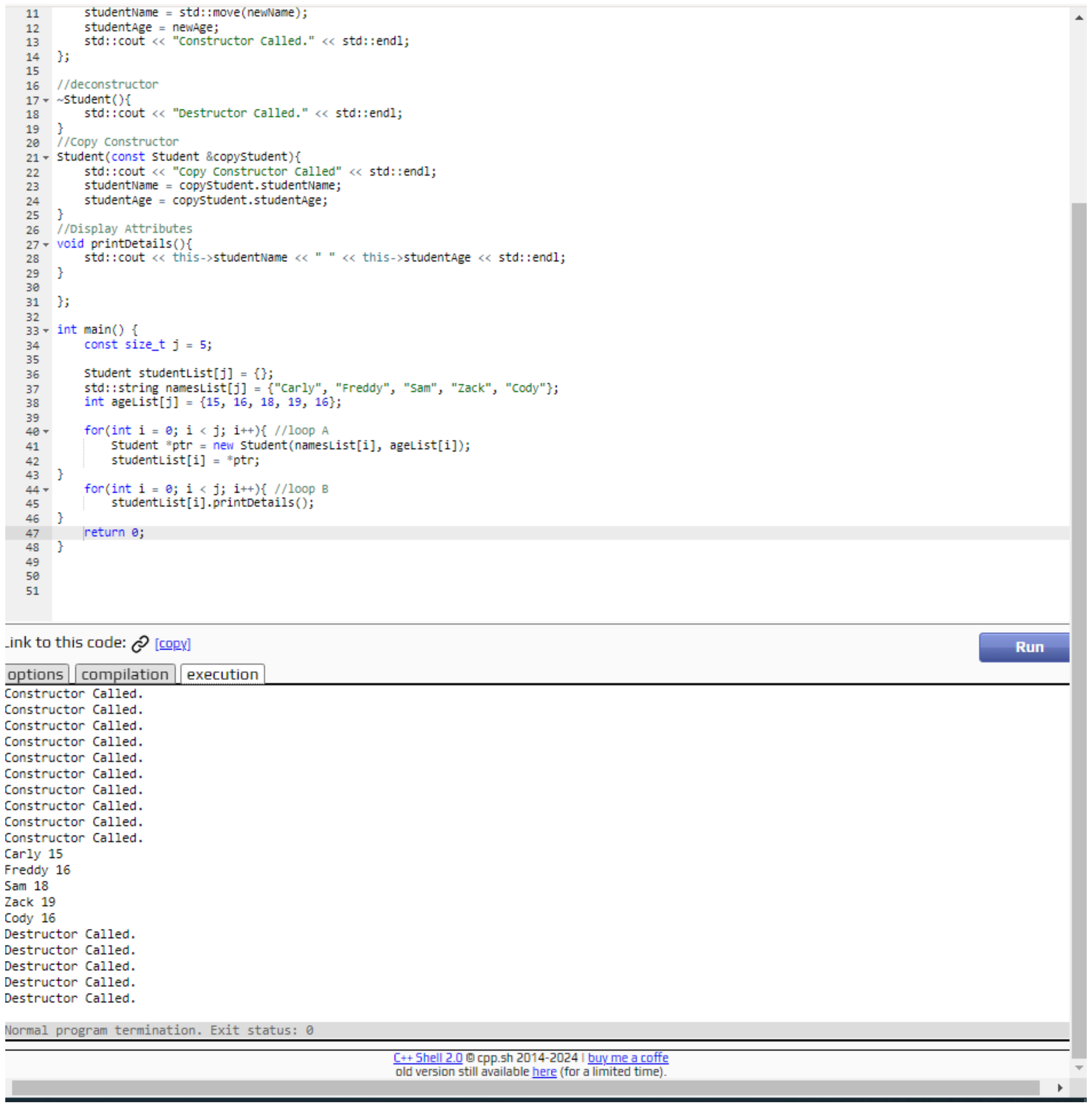
```
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Carly 15
Freddy 16
Sam 18
Zack 19
Cody 16
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.

Normal program termination. Exit status: 0
```

C++ Shell 2.0 © cpp.sh 2014-2024 | buy me a coffe
old version still available here (for a limited time).

Observation

- The student would be the constructor
- the studentList[1] would be the copy constructor
  - the output would be “Copy Constructor Called”

Loop B



```
    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};

    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;
}
```

Link to this code: [copy]

```
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.

Normal program termination. Exit status: 0
```

C++ Shell 2.0 © cpp.sh 2014-2024 | buy me a coffe
old version still available here (for a limited time).

Observation

Loop B would be the printing of the actual arrays of the nameList and ageList as it pertains to printing the details (printDetails)

Output

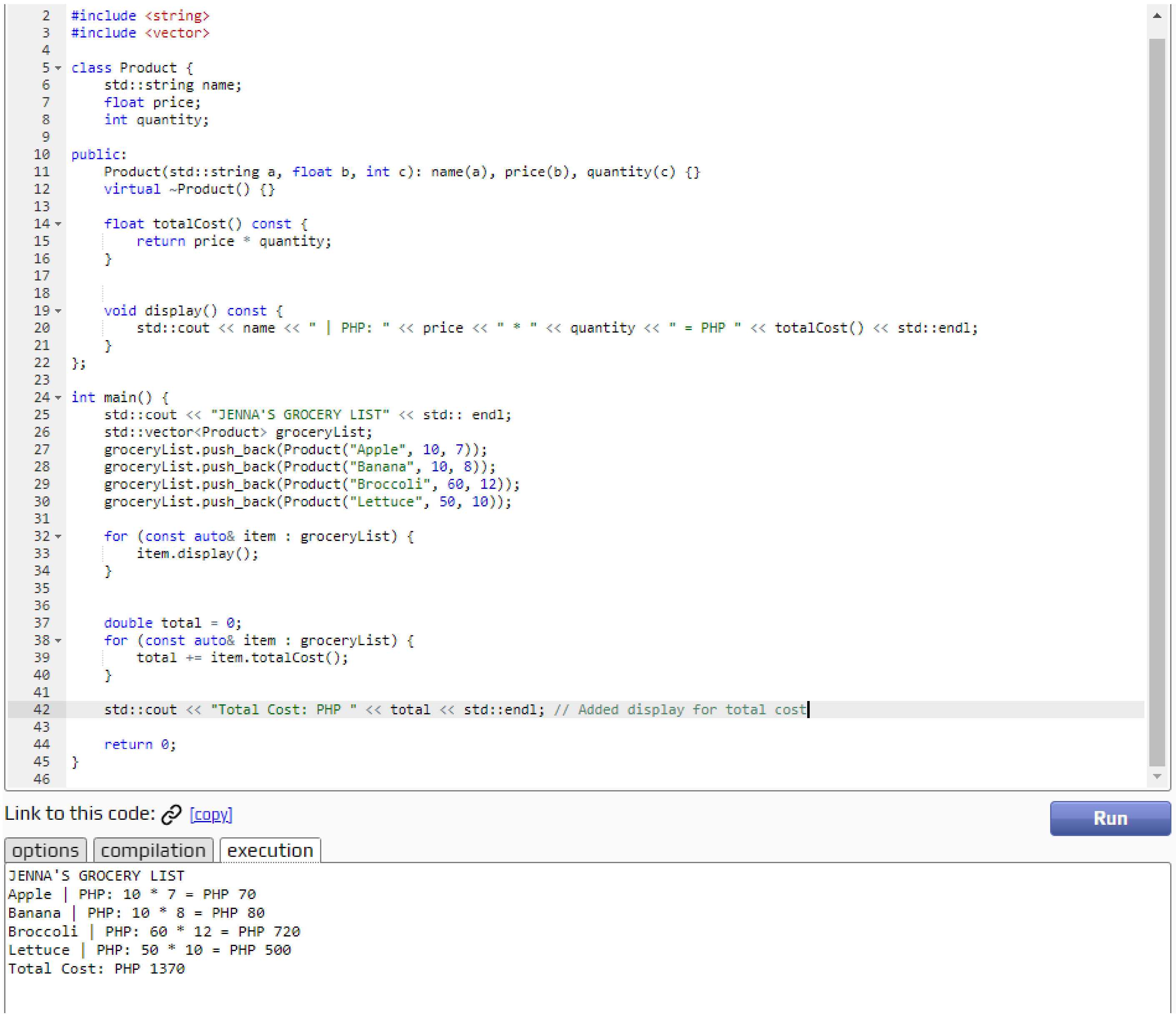
```
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Constructor Called.
Carly 15
Freddy 16
Sam 18
Zack 19
Cody 16
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
Destructor Called.
```

Observation	compared to the prior tables, we see that the details are now also part of the output

Table 2-3 Final Driver Program

7. Supplementary Activity

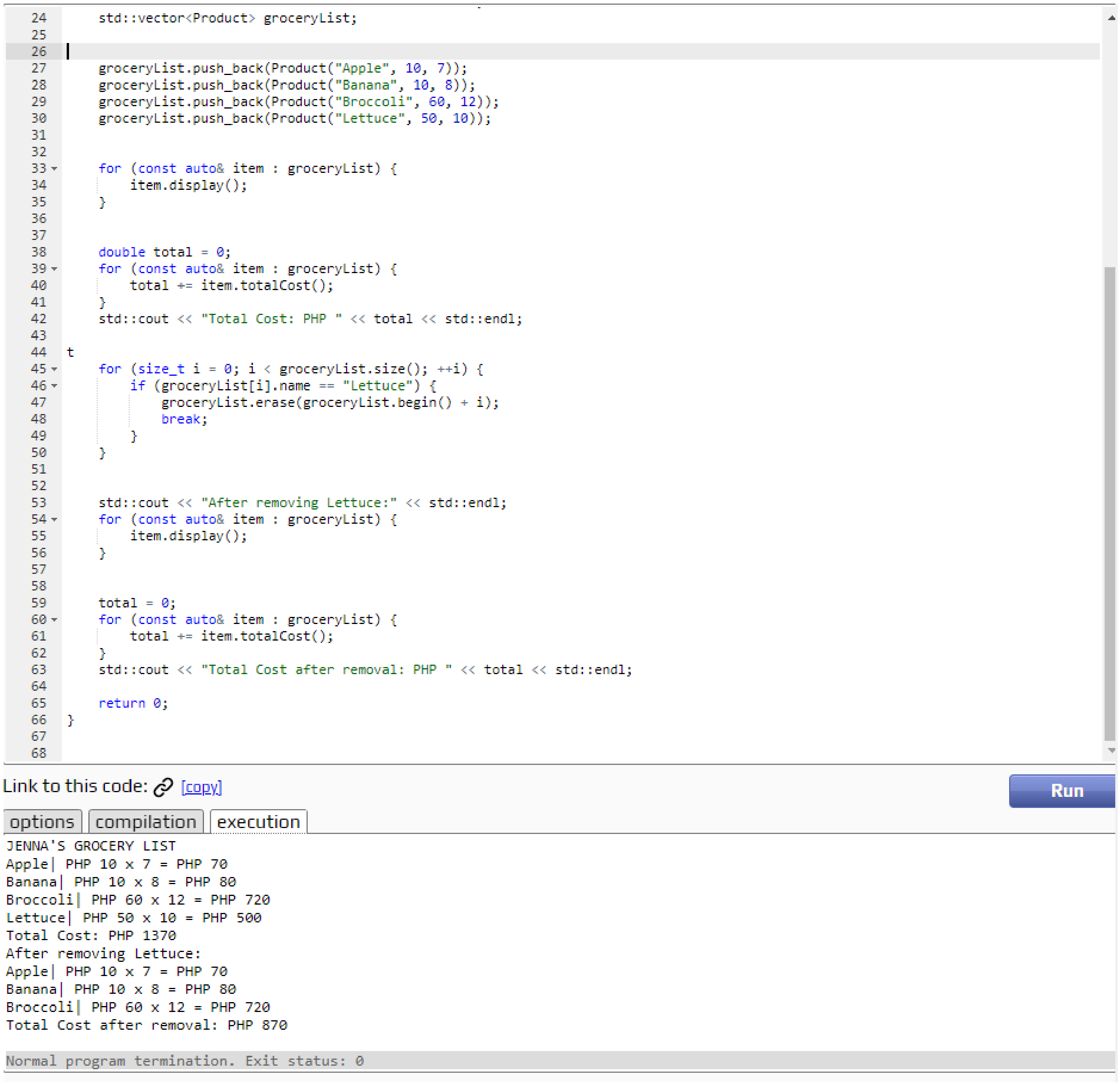
PROBLEM: 1 & 2



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

class Product {
    std::string name;
    float price;
    int quantity;

public:
    Product(std::string a, float b, int c): name(a), price(b), quantity(c) {}
    virtual ~Product() {}

    float totalCost() const {
        return price * quantity;
    }


    void display() const {
        std::cout << name << " | PHP: " << price << " * " << quantity << " = PHP " << totalCost() << std::endl;
    }
};

int main() {
    std::cout << "JENNA'S GROCERY LIST" << std:: endl;
    std::vector<Product> groceryList;
    groceryList.push_back(Product("Apple", 10, 7));
    groceryList.push_back(Product("Banana", 10, 8));
    groceryList.push_back(Product("Broccoli", 60, 12));
    groceryList.push_back(Product("Lettuce", 50, 10));

    for (const auto& item : groceryList) {
        item.display();
    }


    double total = 0;
    for (const auto& item : groceryList) {
        total += item.totalCost();
    }

    std::cout << "Total Cost: PHP " << total << std::endl; // Added display for total cost

    return 0;
}
```

```
JENNA'S GROCERY LIST
Apple | PHP: 10 * 7 = PHP 70
Banana | PHP: 10 * 8 = PHP 80
Broccoli | PHP: 60 * 12 = PHP 720
Lettuce | PHP: 50 * 10 = PHP 500
Total Cost: PHP 1370
```

PROBLEM 3 & 4



```
    std::vector<Product> groceryList;


    groceryList.push_back(Product("Apple", 10, 7));
    groceryList.push_back(Product("Banana", 10, 8));
    groceryList.push_back(Product("Broccoli", 60, 12));
    groceryList.push_back(Product("Lettuce", 50, 10));


    for (const auto& item : groceryList) {
        item.display();
    }


    double total = 0;
    for (const auto& item : groceryList) {
        total += item.totalCost();
    }
    std::cout << "Total Cost: PHP " << total << std::endl;

t
    for (size_t i = 0; i < groceryList.size(); ++i) {
        if (groceryList[i].name == "Lettuce") {
            groceryList.erase(groceryList.begin() + i);
            break;
        }
    }


    std::cout << "After removing Lettuce:" << std::endl;
    for (const auto& item : groceryList) {
        item.display();
    }


    total = 0;
    for (const auto& item : groceryList) {
        total += item.totalCost();
    }
    std::cout << "Total Cost after removal: PHP " << total << std::endl;

    return 0;
}
```

```
JENNA'S GROCERY LIST
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
Total Cost: PHP 1370
After removing Lettuce:
Apple| PHP 10 x 7 = PHP 70
Banana| PHP 10 x 8 = PHP 80
Broccoli| PHP 60 x 12 = PHP 720
Total Cost after removal: PHP 870

Normal program termination. Exit status: 0
```

8. Conclusion

Dynamic memory allocation is a helpful way to sort out information especially when storing out its characteristic to have a better understanding of the data that was given. For me, it was difficult at first to understand how the concept of memory dynamic allocation works especially on how to apply it in c++ as well as when this concept was combined in creating pointers and arrays, but eventually I manage to understand it especially when I was doing the output activity as it gave a breakdown of how constructors, destructor, and copy constructor works for every situation. I think that this lesson is important especially for when it is important to organize information such as a patient's information or a student's data.

9. Assessment Rubric