

CSE 108 (January 2024)

Offline 02

Deadline: September 29 (11:55 PM)

Congratulations on successfully completing your first offline assignment! Now, for your second offline, there are a few important modifications needed to build on the concepts you've learned so far. In this offline, we will be focusing on more advanced topics such as dynamic memory allocation, copy constructors, function overloading, and object handling between functions.

Below are the important changes that you need to apply on **Offline 01** to complete this offline assignment:

Key Modifications:

1. Function Overloading (2 marks):

- In the *ShapeCollection* class, replace the individual functions *addRectangle*, *addTriangle*, and *addCircle* with a **single overloaded function** *add()* that can handle all shape types:
 - *add(Rectangle r)*
 - *add(Triangle t)*
 - *add(Circle c)*
- Each overloaded version of the *add()* method will receive its respective shape object by value.

2. Dynamic Allocation of Objects (2 marks):

- **ShapeCollection Class:**
 - The arrays for storing *Rectangle*, *Triangle*, and *Circle* objects will be **dynamically** allocated using *new*.
 - You must implement dynamic memory allocation and deallocation for these shapes within the *ShapeCollection* class, including handling resizing as the number of shapes grows.
- **Resizing Policy:**
 - Maintain a *capacity* member variable for each shape type in the *ShapeCollection* class, **initialized to 1**. In the overloaded *add()* functions, when the count exceeds the capacity, **double the capacity** and resize the dynamic array accordingly. You can provide a message during resizing (check output).

- **Reallocating Memory :**

- **Using *new* :** Allocate a new array of increased capacity. Use the *clone()* method of each shape to ensure a **deep copy** of existing shapes into the new array. After copying, delete the old array to free up memory.

3. Copy Constructors (2 marks):

- Implement **copy constructors** in each shape class to handle deep copying of dynamically allocated memory (e.g., *color*).
- The copy constructors will be invoked when objects are passed to the overloaded *add()* function and when objects are returned from the *clone()* function.

4. Deep Copying via *clone()* Method (2 marks):

→ ***clone()* Method:**

- To prevent issues with shallow copies (where only pointers are copied instead of the actual data), each shape class must implement a *clone()* method.
- The *clone()* method will create a new object of the same type (e.g., a *new Rectangle*), ensuring that all the attributes of the original object are copied.
- This includes performing a **deep copy** of any dynamically allocated memory (such as the *color* attribute). This guarantees that the new object has its own copy of the data, rather than sharing memory with the original object.
- The *clone()* method will return a **new object** (deep copy), and this object will be directly assigned to the dynamic arrays in the overloaded *add()* method, preserving the original object's data without causing memory conflicts.

5. Destructor Implementation (2 marks):

- Each class must have a **destructor** to free any dynamically allocated memory (e.g., *color*).
- For the *ShapeCollection* class, the **destructor** must ensure that all dynamically allocated arrays and their elements (shapes) are properly deleted.

You can copy the *main()* function for your convenience: [link](#)

Expected output:

```
Rectangle Perimeter: 60
Rectangle Area: 200
Rectangle Color: Red
Rectangle Color: Yellow
-----
Triangle Perimeter: 12
Triangle Color: Blue
Triangle Area: 6
Triangle Color: Orange
-----
Circle Perimeter: 43.96
Circle Area: 153.86
Circle Color: Green
Circle Color: Purple
-----
Increasing capacity of rectangles from 1 to 2
Increasing capacity of rectangles from 2 to 4
Increasing capacity of triangles from 1 to 2

Number of Rectangles: 3
Number of Triangles: 2
Number of Circles: 1
Number of Rectangles: 3
Number of Rectangles: 3
Number of Triangles: 2
Number of Circles: 1
-----
Rectangle 0: length: 10 width: 20
Rectangle 1: length: 15 width: 25
Rectangle 2: length: 150 width: 225
Triangle 0: a: 3 b: 4 c: 5
Triangle 1: a: 5 b: 12 c: 13
Circle 0: radius: 7
```

Shallow Copy vs Deep Copy

1. Shallow Copy:

- Only copies the pointer to the original object's memory. Both objects will point to the same memory location.
- If one object modifies the memory, it affects the other, and if both try to free the same memory, it causes errors (like double frees).

2. Deep Copy:

- Allocates new memory for each copied object, ensuring that each object has its own independent copy of the data.
- This prevents issues like memory corruption and double frees.

Submission Guidelines:

1. Create a folder named after your student ID.
2. Write your code in a single file named your_ID.cpp.
3. Move the .cpp file into the folder and zip the folder.
4. Submit the zip file on Moodle.

Deadline: September 29 (11:55 PM)

**Please do not copy from any source (friends, internet, AI tools like ChatGPT, etc.).
Doing so may lead to severe penalties.**