MTH 4300: Algorithms, Computers and Programming II

Spring 2026

Section: SMWA

Problem Set 5

I know that it's very easy to just use AI to solve these problems, but please resist! I expect you all to give these problems an honest attempt. Spend about 30 minutes trying each problem yourself first. If you're still not getting to a good place with your solution, then prompt AI to guide you to a solution rather than expect it to solve the entire problem for you.

Problem 1: Shopping Cart

You are building an online shopping cart. Your task is to implement a `ShoppingCart` class that manages a dynamically allocated array of item names. Because the class owns heap memory, you must implement all five of the **Rule of Five** special member functions.

Your solution must be split across three files:

- `shopping_cart.h` — the class declaration (provided for you)
- `shopping_cart.cpp` — the class method definitions (**you write this**)
- `main.cpp` — the `main()` function (provided for you)

To compile, run:

```
g++ -std=c++17 -o out main.cpp shopping_cart.cpp
```

You only need to write `shopping_cart.cpp`. The header file and main function are provided below — copy them exactly into `shopping_cart.h` and `main.cpp`.

Provided: `shopping_cart.h`

```cpp
#ifndef SHOPPING_CART_H
#define SHOPPING_CART_H

#include <iostream>
#include <string>

class ShoppingCart {
private:
  std::string owner;
  std::string* items;
  int count;
  int capacity;

public:
  // Constructor
  ShoppingCart(const std::string& owner, int capacity);

  // Rule of Five
  ~ShoppingCart();                                        // 1. Destructor
  ShoppingCart(const ShoppingCart& other);                // 2. Copy constructor
  ShoppingCart& operator=(const ShoppingCart& other);     // 3. Copy assignment
  ShoppingCart(ShoppingCart&& other);                     // 4. Move constructor
  ShoppingCart& operator=(ShoppingCart&& other);          // 5. Move assignment

  // Member functions
  void add_item(const std::string& item);
  void print() const;
};

#endif
```

Provided: `main.cpp`

```cpp
#include "shopping_cart.h"

int main() {
  // --- Create and populate c1 ---
  ShoppingCart c1("Alice", 5);
  c1.add_item("Laptop");
  c1.add_item("Headphones");
  c1.add_item("Mouse");
  std::cout << std::endl;
  c1.print();
```

```
    // --- Copy constructor ---
    std::cout << std::endl;
    ShoppingCart c2 = c1;
    c2.add_item("Keyboard");
    std::cout << std::endl;
    std::cout << "After adding an item to the copy:" << std::endl;
    c1.print();
    c2.print();

    // --- Copy assignment ---
    std::cout << std::endl;
    ShoppingCart c3("Bob", 2);
    c3 = c1;
    c3.print();

    // --- Self-assignment ---
    std::cout << std::endl;
    std::cout << "Self-assignment test:" << std::endl;
    c3 = c3;
    c3.print();

    // --- Move constructor ---
    std::cout << std::endl;
    ShoppingCart c4 = std::move(c1);
    c4.print();
    std::cout << std::endl;
    std::cout << "After move, c1 is:" << std::endl;
    c1.print();

    // --- Move assignment ---
    std::cout << std::endl;
    ShoppingCart c5("Charlie", 1);
    c5 = std::move(c2);
    c5.print();
    std::cout << std::endl;
    std::cout << "After move, c2 is:" << std::endl;
    c2.print();

    // --- Test full cart ---
    std::cout << std::endl;
    c5.add_item("Monitor");
    c5.add_item("Webcam");
    std::cout << std::endl;

    // --- Destructors fire here in reverse order ---
    std::cout << "--- End of main ---" << std::endl;
    return 0;
}
```

Your Task: Implement `shopping_cart.cpp`

Using the provided header file, implement all methods declared in `ShoppingCart`. This includes the constructor, destructor, all four remaining Rule of Five members, `add_item`, and `print`.

Each special member function should print a message when called so you can trace what's happening:

Function	**Message**
Destructor	`Cart Alice destroyed`
Copy constructor	`Cart Alice copied`
Copy assignment	`Cart Alice copy-assigned`
Move constructor	`Cart Alice moved`
Move assignment	`Cart Alice move-assigned`

The `print` method should display the cart contents in the format shown in the expected output. A cart that has been moved from should print `(empty - moved)` instead of its items.

If `add_item` is called on a full cart, print: `Cart [owner] is full!`

Refer to Lectures 13 and 14 for guidance on implementing deep copies, move semantics, and common pitfalls like self-assignment.

Submit only `shopping_cart.cpp`.

Expected Output

```
--- Cart: Alice (3 items) ---
  1. Laptop
  2. Headphones
  3. Mouse

Cart Alice copied
After adding an item to the copy:
--- Cart: Alice (3 items) ---
  1. Laptop
  2. Headphones
  3. Mouse
--- Cart: Alice (4 items) ---
  1. Laptop
  2. Headphones
  3. Mouse
  4. Keyboard

Cart Alice copy-assigned
--- Cart: Alice (3 items) ---
  1. Laptop
  2. Headphones
  3. Mouse

Self-assignment test:
--- Cart: Alice (3 items) ---
  1. Laptop
  2. Headphones
  3. Mouse

Cart Alice moved
--- Cart: Alice (3 items) ---
  1. Laptop
  2. Headphones
  3. Mouse

After move, c1 is:
--- Cart: Alice (empty - moved) ---

Cart Alice move-assigned
--- Cart: Alice (4 items) ---
  1. Laptop
  2. Headphones
  3. Mouse
  4. Keyboard

After move, c2 is:
--- Cart: Alice (empty - moved) ---

Cart Alice is full!

--- End of main ---
Cart Alice destroyed
Cart Alice destroyed
Cart Alice destroyed
Cart Alice destroyed
Cart Alice destroyed
```

Note: The five "destroyed" messages at the end print in reverse order of construction — `c5` is destroyed first, then `c4`, `c3`, `c2`, and finally `c1`. They all have the owner "Alice" because `c3`, `c4`, and `c5` received their owner through copy/move operations from `c1`.