MTH 4300: Algorithms, Computers, and Programming II

HW #4

Due Date: Dec 1st, 2025

Problem 1

You are to implement a simple `Library` system where a `Library` object manages a dynamically allocated array of `Book` objects(Use your implementation of Book from assignment 3). Your task is to implement **the Big Five** (constructor, copy constructor, copy assignment operator, move constructor, move assignment operator, and destructor) for the `Library` class to ensure proper resource management.

Task: Implement the `Library` Class

You must implement the following `Library` class:

Listing 1: Library class specification

```cpp
class Library {
private:
    Book* books;      // dynamically allocated array of Book objects
    size_t size;

public:
    // 1. Default constructor
    Library();

    // 2. Parameterized constructor
    Library(size_t n);

    // 3. Copy constructor
    Library(const Library& other);

    // 4. Copy assignment operator
    Library& operator=(const Library& other);

    // 5. Move constructor
    Library(Library&& other);

    // 6. Move assignment operator
    Library& operator=(Library&& other);

    // 7. Destructor
    ~Library();

    // Utility functions
    void setBook(size_t index, const Book& book);
    void print() const;
    size_t getSize() const;
};
```

Requirements

- The parameterized constructor should allocate an array of `n` `Book` objects.
- The copy constructor and copy assignment operator must perform **deep copies** of all `Book` objects.
- The move constructor and move assignment operator should **transfer ownership** of the array.
- The destructor must release all dynamically allocated memory.
- The `setBook()` method replaces a `Book` at a specific index.
- The `print()` method displays all books in the library.

Example Usage

Listing 2: Example usage of the Library class

```cpp
int main() {
    Library lib1(2);
    lib1.setBook(0, Book("1984", "George Orwell"));
    lib1.setBook(1, Book("Brave New World", "Aldous Huxley"));
    lib1.print();

    // Output:
    // "1984" by George Orwell
    // "Brave New World" by Aldous Huxley

    Library lib2 = lib1;  // Copy constructor
    lib2.setBook(0, Book("Fahrenheit 451", "Ray Bradbury"));
    lib2.print();
    lib1.print(); // lib1 should remain unchanged

    Library lib3;
    lib3 = lib1; // Copy assignment
    lib3.print();

    Library lib4 = std::move(lib1); // Move constructor
    lib4.print(); // Works
    lib1.print(); // Safe, but should show empty

    Library lib5;
    lib5 = std::move(lib4); // Move assignment
    lib5.print();
}
```

Problem 2

Description

Implement a simple text editor that supports two operations:

- `TYPE <word>` — adds a word to the current text.
- `UNDO` — removes the most recently typed word.

Use a **stack** to track the history of typed words and support undo operations.

Example

```
Input:
TYPE hello
TYPE world
UNDO
TYPE there

Output:
Text: hello there
```

Implementation Hints

- Use a `std::stack<std::string>` to store the typed words.
- Pop from the stack when an `UNDO` is requested.
- At the end, rebuild the text by popping elements or iterating over the stack.

Example Code Templates

Stack Example Skeleton

```cpp
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int main() {
    stack<string> history;
    string command, word;

    while (cin >> command) {
        if (command == "TYPE") {
            cin >> word;
            history.push(word);
        } else if (command == "UNDO") {
            if (!history.empty()) history.pop();
        }
    }

    // Print the current text
    stack<string> temp;
    while (!history.empty()) {
        temp.push(history.top());
        history.pop();
    }
    cout << "Text: ";
    while (!temp.empty()) {
        cout << temp.top() << " ";
        temp.pop();
    }
}
```

Problem 3

Simulate a simple **call center queue system**. Customers call in and are served in the order they arrive. Each call takes a variable amount of time to handle.

Use a **queue** to store incoming calls.

Example Input and Output

```
Input Calls (ID, duration in minutes):
(1, 3), (2, 5), (3, 2)

Output:
Call 1 started (3 min)
Call 1 finished
Call 2 started (5 min)
Call 2 finished
Call 3 started (2 min)
Call 3 finished
Total handling time: 10 min
```

Implementation Hints

- Represent each call as a `std::pair<int, int>` where the first element is the call ID and the second is the duration.
- Use a `std::queue<std::pair<int, int>>` to hold calls.
- Process them in FIFO order and sum up the total duration.

Example Code Templates

Queue Example Skeleton

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<pair<int,int>> calls;
    calls.push({1, 3});
    calls.push({2, 5});
    calls.push({3, 2});

    int total = 0;
    while (!calls.empty()) {
        auto call = calls.front();
        calls.pop();

        cout << "Call " << call.first << " started ("
             << call.second << " min)" << endl;
        total += call.second;
        cout << "Call " << call.first << " finished" << endl;
    }

    cout << "Total handling time: " << total << " min" << endl;
}
```

Problem 4

For the questions below, **Do not use STL List** and modify the LinkedList implementation we covered in class

0.1

You are given two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the merged linked list. What is the runtime and spacetime complexity of your algorithm.

```
LinkedList merge(LinkedList list1, LinkedList list2){...}
```

Write a method to remove the nth node from the end of the list. Return true if deletion was successful, otherwise return false. What is the runtime and spacetime complexity of your algorithm.

```
bool LinkedList::deleteNthNodeFromEnd(int n){...}
```

Given the node:

```
struct node
{
        string first_name;
        string last_name;
        node* next;
}
```

1. Modify the link list class to sort by first name, then by last name. You must implement the sort function yourself, using the selection sort strategy. Add your sort function as a method to LinkedList class. What is the runtime and spacetime complexity of your algorithm?
2. In the main function, open the file **names_list.txt** in c++ and write your name to the file. (use fstream)
3. Read from the names_list.txt(including your name), and create a linked list object. Each line in the file should correspond to one node, containing a first name and last name. Use the sort algorithm you wrote in part 1, to sort this linked list. Create a new file **sorted_names.txt**, and write the sorted list to this file, one name per line.

Problem 5

Do problem 4 using stl container list