MTH 4300: Algorithms, Computers and Programming II

Fall 2025

Section: STRA

Problem Set 4

Due Date: October 31, 2025

Question 1: String Class with Copy Control

Implement a `MyString` class that manages a dynamically allocated C-style string (char array). This question focuses on proper resource management using the Rule of Five.

Class Specification:

```
class MyString {
private:
    char* data;
    size_t length;

public:
    // Constructors
    MyString();
    MyString(const char* str);

    // Rule of Five
    ~MyString();
    MyString(const MyString& other);
    MyString& operator=(const MyString& other);
    MyString(MyString&& other) noexcept;
    MyString& operator=(MyString&& other) noexcept;

    // Member functions
    size_t size() const;
    const char* c_str() const;
    void append(const char* str);
    void append(const MyString& other);
    char& operator[](size_t index);
    const char& operator[](size_t index) const;

    // Operator overloading
    MyString operator+(const MyString& other) const;
    bool operator==(const MyString& other) const;
    friend std::ostream& operator<<(std::ostream& os, const MyString& str);
};
```

Requirements:

1. **Default Constructor:** Initialize with empty string (allocate char array of size 1 with null terminator `'\0'`)
2. **Parameterized Constructor:** Initialize with given C-string
   - Calculate length using `strlen()` or manual loop
   - Allocate memory: `length + 1` bytes (for null terminator)
   - Copy characters including null terminator
3. **Destructor:** Properly delete the dynamically allocated char array
4. **Copy Constructor:** Perform deep copy of the string
   - Allocate new memory
   - Copy all characters including null terminator
5. **Copy Assignment Operator:**
   - Check for self-assignment (`this != &other`)
   - Delete old data
   - Perform deep copy
   - Return `*this`
6. **Move Constructor:**
   - Transfer ownership of resources (steal pointer)
   - Leave source in valid empty state (data points to empty string, length=0)
7. **Move Assignment Operator:**
   - Check for self-assignment
   - Delete old data
   - Transfer ownership
   - Leave source in valid empty state
   - Return `*this`
8. **size():** Return the length of the string (not including null terminator)
9. **c_str():** Return pointer to internal char array (for C-style string compatibility)
10. **append():** Add another string to the end
    - Allocate new larger array
    - Copy existing content
    - Copy appended content
    - Delete old array
11. **Subscript Operator:** Access characters by index
    - No bounds checking required (assume valid indices)
12. **operator+:** Concatenate two strings, return new MyString
13. **operator==:** Compare two strings for equality (use `strcmp()` or manual comparison)
14. **Stream operator (<<):** Output the string content

Example Usage:

```
// Construction
MyString s1;                    // Empty string
MyString s2("Hello");           // "Hello"
MyString s3("World");           // "World"

std::cout << s2 << std::endl;   // Output: Hello
std::cout << "Length: " << s2.size() << std::endl;  // Output: Length: 5

// Copy construction (deep copy)
MyString s4 = s2;
s4.append(" C++");
std::cout << s4 << std::endl;   // Output: Hello C++
std::cout << s2 << std::endl;   // Output: Hello (unchanged)

// Copy assignment
MyString s5;
s5 = s2;
std::cout << s5 << std::endl;   // Output: Hello

// Concatenation with operator+
MyString s6 = s2 + MyString(" ") + s3;
std::cout << s6 << std::endl;   // Output: Hello World

// Move construction (transfer ownership)
MyString s7 = std::move(s2);
std::cout << s7 << std::endl;   // Output: Hello
std::cout << s2 << std::endl;   // Output: (empty string)

// Comparison
if (s3 == MyString("World")) {
    std::cout << "Strings are equal" << std::endl;
}

// Subscript operator
std::cout << s3[0] << std::endl;  // Output: W
s3[0] = 'w';
std::cout << s3 << std::endl;     // Output: world
```

Testing Requirements:

- Test both constructors (empty and with C-string)
- Test copy constructor creates independent copy (modify one, other unchanged)
- Test copy assignment operator (including self-assignment: `s = s;`)
- Test move constructor leaves source in empty state
- Test move assignment leaves source in empty state
- Test append() with both C-strings and MyString objects
- Test operator+ for concatenation
- Test operator== for comparison
- Test subscript operator for read and write access
- Verify no memory leaks (run with valgrind or similar tool)

Memory Leak Check:

```
valgrind --leak-check=full ./your_program
# Should show: All heap blocks were freed -- no leaks are possible
```

Hints:

- Use `strlen()` from `<cstring>` to calculate string length
- Use `strcpy()` or manual loop to copy strings
- Use `strcmp()` or manual loop to compare strings
- Remember to always allocate `length + 1` bytes for null terminator
- For move operations, leaving source as empty string is safer than nullptr

Question 2: Linked List

Given Node Structure:

```
struct Node {
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    /** use the implementation we created in class */

    // Operations to implement
    void reverse();
};
```

Implement `void reverse()` that reverses the linked list in-place (without creating new nodes).

Algorithm Hint: Use three pointers (previous, current, next) to reverse the links.

Example:

```
LinkedList list;
list.pushFront(30);
list.pushFront(20);
list.pushFront(10);
std::cout << list << std::endl; // Output: 10 -> 20 -> 30 -> nullptr

list.reverse();
std::cout << list << std::endl;  // Output: 30 -> 20 -> 10 -> nullptr
```

Requirements:

- In-place reversal (O(1) space, don't create new nodes)
- Time complexity: O(n)
- Handle edge cases: empty list, single node