MTH 4300: Computer Science and Programming II

Spring 2026

Section: SMWA

Midterm 2 Practice Exam 1

Instructions:

- Each question below is a **programming** problem.
- In **Part A**, write clear pseudocode that outlines your approach.
- In **Part B**, write a complete C++ implementation.
- Use `std::` prefix where appropriate. Do not use `using namespace std`.
- You may assume all necessary headers are included unless stated otherwise.

Note: Submit handwritten responses as a PDF.

Question 1: Object-Oriented Programming (Rule of 5)

Implement a class `Polynomial` that represents a polynomial of the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1}$$

where the coefficients are stored in a **dynamically allocated** array of `double`.

Your class must have the following private members:

- `double* coeffs` — pointer to the coefficient array
- `int degree` — the degree of the polynomial (so the array has `degree + 1` entries)

Your class must implement **all five** of the following special member functions:

1. Constructor that takes an `int degree` and a `double* values` (the caller's array of length `degree + 1`) and creates a deep copy.
2. Destructor that releases the dynamically allocated memory.
3. Copy constructor (deep copy).
4. Copy assignment operator (deep copy, handling self-assignment).
5. Move constructor.
6. Move assignment operator.

Additionally, implement a public method:

- `double evaluate(double x) const` — returns the value of the polynomial at `x`.

Part A: Pseudocode

Write pseudocode describing the constructor, destructor, copy constructor, copy assignment, move constructor, move assignment, and `evaluate`.

Part B: C++ Implementation

Write the complete class definition including all six member functions listed above.

Question 2: Stacks (Daily Temperatures)

Write a C++ function

```
std::vector<int> daily_temperatures(const std::vector<int>& temps);
```

that, given a vector `temps` of daily temperatures, returns a vector `answer` of the same length such that `answer[i]` is the **number of days** you have to wait after day `i` to encounter a **strictly warmer** temperature. If no such future day exists, set `answer[i] = 0`.

Examples:

- `daily_temperatures({73, 74, 75, 71, 69, 72, 76, 73})` returns `{1, 1, 4, 2, 1, 1, 0, 0}`
- `daily_temperatures({30, 40, 50, 60})` returns `{1, 1, 1, 0}`
- `daily_temperatures({30, 60, 90})` returns `{1, 1, 0}`
- `daily_temperatures({50, 50, 50})` returns `{0, 0, 0}`

Requirement: your solution must run in $O(n)$ time using a **monotonic stack** of **indices** (`std::stack<int>` from the `<stack>` header). The idea:

- Iterate through `temps` from left to right.
- Maintain a stack of indices whose answer has not yet been determined, kept in order of decreasing temperature.
- When the current temperature is strictly greater than the temperature at the index on top of the stack, pop that index and record the difference between the current index and the popped index.

Then, write a `main` function that tests your implementation on the four examples above and prints each output vector in the format `[a, b, c, ...]`.

Part A: Pseudocode

Describe the monotonic-stack algorithm step by step, and explain why it runs in $O(n)$ time.

Part B: C++ Implementation

Write the complete program, including all necessary `#include` directives.

Question 3: Hashing with Maps (Group Anagrams)

Two strings are **anagrams** of each other if one can be rearranged to form the other (e.g., `"eat"`, `"tea"`, and `"ate"` are all anagrams).

Write a C++ function

```
std::vector<std::vector<std::string>> group_anagrams(const std::vector<std::string>& words);
```

that groups the input words so that all anagrams of one another appear in the same inner vector. The order of the groups in the outer vector and the order of words within each group do not matter.

Examples:

- `group_anagrams({"eat", "tea", "tan", "ate", "nat", "bat"})` returns groups equivalent to `{{"eat", "tea", "ate"}, {"tan", "nat"}, {"bat"}}`
- `group_anagrams({""})` returns `{{""}}`
- `group_anagrams({"abc", "cba", "xyz", "zyx", "foo"})` returns groups equivalent to `{{"abc", "cba"}, {"xyz", "zyx"}, {"foo"}}`

Requirement: your solution must use `std::unordered_map<std::string, std::vector<std::string>>`. The **key** is a canonical form of the word (the word's characters sorted alphabetically), and the **value** is the vector of all input words that share that canonical form. You may use `std::sort` from `<algorithm>` to sort a copy of each word.

Then, write a `main` function that tests your function on the three examples above. Print each group on its own line in the format `[word1, word2, ...]`.

Part A: Pseudocode

Describe your algorithm: how you build the map, how you assemble the final result, and the role of the sorted string as a key.

Part B: C++ Implementation

Write the complete program, including all necessary `#include` directives.

Question 4: Hashing with Sets (Longest Consecutive Sequence)

Write a C++ function

```
int longest_consecutive(const std::vector<int>& nums);
```

that returns the length of the longest sequence of **consecutive integers** (in the mathematical sense, e.g., `4, 5, 6, 7`) that can be formed from the values in `nums`. The numbers do not need to appear in order in the input, and duplicates should be ignored.

Examples:

- `longest_consecutive({100, 4, 200, 1, 3, 2})` returns `4` (the sequence `1, 2, 3, 4`)
- `longest_consecutive({0, 3, 7, 2, 5, 8, 4, 6, 0, 1})` returns `9` (the sequence `0, 1, 2, ..., 8`)
- `longest_consecutive({})` returns `0`
- `longest_consecutive({10})` returns `1`
- `longest_consecutive({1, 2, 0, 1})` returns `3` (the sequence `0, 1, 2`; the duplicate `1` is ignored)

Requirement: your solution must run in $O(n)$ expected time using `std::unordered_set<int>`. A simple sort-based approach is **not acceptable**. The idea is:

- Insert every value of `nums` into a `std::unordered_set<int>`.
- For each value `x` in the set, check whether `x - 1` is also in the set. If it is, then `x` is **not** the start of a sequence — skip it.
- If `x - 1` is **not** in the set, then `x` is the start of a sequence. Count how many consecutive values (`x`, `x + 1`, `x + 2`, …) are in the set, and update the running maximum.

This guarantees that the inner counting loop only runs once per sequence, so the total work is $O(n)$ expected.

Then, write a `main` function that tests your function on the five examples above.

Part A: Pseudocode

Describe the algorithm above, and explain why it runs in $O(n)$ expected time even though it appears to contain nested loops.

Part B: C++ Implementation

Write the complete program, including all necessary `#include` directives.