Winter 2026 CS 31 Project 4

Due Date: February 18, 2026 at 11:59 P.M.

This project must be your independent work. Collaboration and AI tools are not permitted.

Array Processing

In this project, you will implement several functions that manipulate arrays of `string` values. These functions reinforce linear scanning, lexicographic comparison, subsequence detection, in-place modification, and partitioning.

Throughout this assignment, when we refer to "the array," we mean only the first n elements of the array passed to the function.

Unless otherwise specified:

- If a function receives a negative value of n, it must return -1 (or `false` for Boolean functions).
- String comparison is case-sensitive.
- No additional arrays may be used unless explicitly permitted.
- You may assume that the number of elements we use to test your code will not exceeded the size of the array.
- An array with $n = 0$ is valid.

Functions to Implement

Implement the following functions exactly as specified. We have included examples of expected behavior for each function in **array_test.cpp**.

1. **int countOccurrences(const string a[], int n, string target);**
   Return the number of elements in the array equal to `target`.
2. **int locateOccurrence(const string a[], int n, string target);**
   Return the smallest index at which `target` appears in the array. Return -1 if the target does not appear.
3. **int locateMin(const string a[], int n);**
   Return the index of the smallest string in the array (using lexicographic comparison). If there is more than one such element, return the smallest index.
4. **int eliminateDuplicates(string a[], int n);**
   For each sequence of consecutive identical strings, retain only one copy. The retained elements must occupy positions 0 through $r - 1$, where r is the number of retained elements. Return r.
5. **bool contains(const string a1[], int n1, const string a2[], int n2);**
   Return `true` if all elements of `a2` appear in `a1` in the same relative order (not necessarily consecutively). Return `false` otherwise.

6. **int divide(string a[], int n, string divider);**
   Rearrange the array so that all elements less than `divider` come before all other elements, and all elements greater than `divider` come after all other elements. Return the index of the first element that is not less than `divider`, or n if all elements are less.

7. **int moveToBack(string a[], int n, int pos);**
   Remove the element at index `pos` by shifting subsequent elements left by one position. Place the removed element at position $n - 1$. Return the original position of the moved element.

Programming Requirements

1. You may not use global variables that are not constant.
2. Your code must not read input from `cin` nor write output to `cout`. Use `cerr` for debugging.
3. You must not use the C++ Standard Library sorting functions.
4. You must not use additional arrays unless explicitly allowed.
5. Be careful with index bounds and edge cases.
6. Your program must compile and run correctly using **g31** on **cs31.seas.ucla.edu**.
7. To compile: **g31 -o array_test array.cpp array_test.cpp**

Written Component

In addition to your code, include brief comments in your implementation files explaining:

- The condition that is true for each iteration of your loop in `divide`.
- Why `contains` correctly determines subsequence membership.
- Any edge cases that required special handling.
- Your **array.cpp** file must include meaningful comments explaining your algorithms.
- Your **array_test.cpp** file must contain a `main` function with comprehensive test cases and comments explaining the test cases

What to Submit

Submit a zip file containing:

- **array.h**
- **array.cpp**
- **array_test.cpp**: **array_test.cpp** file must contain a `main` function with comprehensive test cases and comments explaining the test cases