Texas Tech University Whitacre College of Engineering

CS 2413/5401 – Data Structures
Spring 2022
Lab Assignment 2

Acknowledge your collaborators or source of solutions, if any. **Online submission is required.**

While designing your programs or answering items, you are free to come up with your own assumptions based upon concepts and material learned in the course, if every potential specification is not given to you. Just be reasonable and document your assumptions. Such assumptions should not conflict with concepts and material learned in the course.

Your compliance with the "PROGRAMMING STYLE GUIDELINE" for CS 2413/5401 will affect your actual grade. All assignments will be checked for academic misconduct (cheating, plagiarism, collusion, falsifying academic records, misrepresenting facts, violations of published professional ethics/standards, and any act or attempted act designed to give unfair academic advantage to oneself or another student) defined by "OP 34.12: Grading Procedures, Including Academic Integrity" of TTU. Special software will be used to uncover such attempts.

A subset of answers submitted in this lab may be graded.

Objective: Practice using C arrays and strings

Tasks:

1. Write a C program, problem1.c, with at least **one more function** than the main function to print the number of peaks and their locations in an elevation grid. A peak is defined to be a location at which the north, south, east, and west neighbors have lower values than the peak. If a location does not have all four neighbors, then it cannot be a peak.
    - **a.** Declare and initialize the elevation grid as a 2-dimensional array with the following data. In the grid, locations (2,1), (2,5), and (4,3) are peaks.
        - **i.** 5039 5127 5238 5259 5248 5310 5299
        - **ii.** 5150 5392 5410 5401 5320 5820 5321
        - **iii.** 5290 5560 5490 5421 5530 5831 5210
        - **iv.** 5110 5429 5430 5411 5459 5630 5319
        - **v.** 4920 5129 4921 5821 4722 4921 5129
        - **vi.** 5023 5129 4822 4872 4794 4862 4245
    - **b.** Use define macro constants for the number of rows and columns in the grid so that the grid size is not hard coded throughout the program.
    - **c.** Example output:
        - **i.** 3 peaks found:
        - **ii.** Location at row 2 and column 1
        - **iii.** Location at row 2 and column 5
        - **iv.** Location at row 4 and column 3
    - **d.** In comments at the end of the program, note
        - **i.** the output for the program
        - **ii.** for an mxn elevation grid, the Big-O storage complexity required to store the data, the number of peaks, and the peak locations (think carefully about how big the peaks storage needs to be)
        - **iii.** for an mxn elevation grid, the Big-O time complexity required to process the elevation grid to find the peaks, and print the peaks
    - **e.** ***Extra Challenge if you would like to exercise your skills further BUT PLEASE DO NOT SUBMIT WITH THIS ASSIGNMENT:

i. Store the data in a file called grid.txt which has the format:
   1. 6 7
   2. 5039 5127 5238 5259 5248 5310 5299
   3. …

ii. Dynamically allocate the 2-dimensional array after reading in the matrix size from grid.txt and then read the grid from the file into the dynamically allocated array

iii. Make a grid_none.txt file of a different grid size with no peaks and run your program to be sure it answers properly, such as
   1. 0 peaks found

iv. Make a grid_one.txt file of a different grid size with one peak and run your program to be sure it answers properly, such as
   1. 1 peak found
   2. Location at row 2 and column 1

2. Write a C program, problem2.c, with search function to search for a substring within a string that returns -1 if the substring is not found and the position in the string if found. The search function only takes the substring and string as its formal parameters.
Implement the search function with the brute force method given below. Alternatively, you can implement the search function with the suggested optimization given below.
The main function should test the search function to demonstrate substrings are not found if not present in the string even if similar characters are at either end of the string or in the middle, can be found at the string beginning or end, and can be found in the middle of the string.
No "string.h" functions may be used.

a. A brute force way to find a substring within another is to search a string one character at a time until a character matches the first character of the substring as in the example shown below.

i. Find sub = "carrot" in s = "Bob likes cars and carrots"
   1. Compare B to c - mismatch
   2. Compare o to c - mismatch
   3. Compare b to c - mismatch
   4. Compare space to c - mismatch
   5. …
   6. Compare space to c - mismatch
   7. Compare c to c – match (at "cars" in s)
   8. Compare a to a - match
   9. Compare r to r - match
   10. Compare s to r – mismatch, start back at the "a" in "cars"
   11. Compare a to c – mismatch
   12. Compare r to c – mismatch
   13. Compare s to c - mismatch
   14. Compare space to c – mismatch
   15. …
   16. Compare c to c – match
   17. Compare a to a – match
   18. Compare r to r – match
   19. Compare r to r – match
   20. Compare o to o – match
   21. Compare t to t – match
   22. Return location 19 in s

   ii. Think about how to optimize the search so the search algorithm does not have to go back to the "a" in "cars" on a mismatch, such as
      1. Compare B to c – mismatch
      2. …
      3. Compare c to c – match
      4. Compare a to a – match
      5. Compare r to r – match
      6. Compare s to r – mismatch (since "c" was not encountered again, start at the space after "cars")
      7. Compare space to c – mismatch
      8. …

b. Example output for the brute force search function:
   i. Search for 'carrot' in 'carro carro carro carro' – location -1
   ii. Search for 'carrot' in 'carrot carro carro carro' – location 0
   iii. Search for 'caccoc' in 'cacco cacco caccoc cacco' - location 12

c. Example output for the optimized search function with a trace of the i index variable that traverses string s:
   i. substring - i = 0
   ii. substring - i = 6
   iii. substring - i = 12
   iv. substring - i = 18
   v. Search for 'carrot' in 'carro carro carro carro' – location -1
   vi. Substring – i = 0
   vii. Search for 'carrot' in 'carrot carro carro carro' – location 0
   viii. substring - i = 0
   ix. substring - i = 2
   x. substring - i = 3
   xi. substring - i = 4
   xii. substring - i = 5
   xiii. substring - i = 6
   xiv. substring - i = 8
   xv. substring - i = 9
   xvi. substring - i = 10
   xvii. substring - i = 11
   xviii. substring - i = 12
   xix. Search for 'caccoc' in 'cacco cacco caccoc cacco' - location 12

d. In comments at the end of the program, note
   i. the output for the program
   ii. Suppose the substring of size m occurs at the end of a size n string, then,
      1. what would be a worst-case string and substring example for the **brute force** string search and what would be the Big O time complexity for the worst case?
      2. what would be the best-case string and substring example for the **optimized** string search and what would be the Big O time complexity for the best case?

Learning Outcomes:

- Understand C program concepts, such as arrays and strings
- Understand how to analyze the complexity of a program or algorithm

Grading: 40 points

Standard Deductions - 10 points
Problem 1 – 15 points (problem1.c)
Problem 2 – 15 points (problem2.c, zone.txt)

Due Date:

1/28/2022, 11:59pm (submitted on Blackboard)