NYU, Tandon School of Engineering
Bridge to CS — Summer 2020

Homework #7
Due by Friday 8/28, 11:55pm

Submission instructions:

1. For this assignment, you should turn in 7 '.cpp' files (for questions 1 - 7). Name these files: 'YourNetID_hw7_q1.cpp', 'YourNetID_hw7_q2.cpp', etc.

2. **You should submit your homework in the Gradescope system.**

3. Pay special attention to the style of your code. Indent your code correctly, choose meaningful names for your variables, define constants where needed, choose most suitable control statements, break down your solutions by defining functions, etc.

Question 1:

Write a program that will read in a line of text and output the number of words in the line and the number of occurrences of each letter.

Define a word to be any string of letters that is delimited at each end by either whitespace, a period, a comma, or the beginning or end of the line.

You can assume that the input consists entirely of letters, whitespace, commas, and periods.

When outputting the number of letters that occur in a line, be sure to count upper and lowercase versions of a letter as the same letter.

Output the letters in alphabetical order and list only those letters that do occur in the input line.

Your program should interact with the user **exactly** as it shows in the following example:

Please enter a line of text:

I say Hi.

3 words

1 a

1 h

2 i

1 s

1 y

Notes:

1. Think how to break down your implementation to functions.
2. Pay attention to the running time of your program. If the input line contains n characters, an efficient implementation would run in a linear time (that is $\Theta(n)$).

Question 2:

Two strings are **anagrams** if the letters can be rearranged to form each other. For example, "Eleven plus two" is an anagram of "Twelve plus one". Each string contains one 'v', three 'e's, two 'l's, etc.

Write a program that determines if two strings are anagrams. The program should not be case sensitive and should disregard any punctuation or spaces.

Notes:

1. Think how to break down your implementation to functions.
2. Pay attention to the running time of your program. If each input string contains n characters, an efficient implementation would run in a linear time (that is $\Theta(n)$).

Question 3:

Implement the function:

```
void oddsKeepEvensFlip(int arr[], int arrSize)
```

This function gets an array of integers `arr` and its logical size `arrSize`.

When called, it should **reorder** the elements of `arr` so that:

1. All the odd numbers come before all the even numbers
2. The odd numbers will keep their original relative order
3. The even numbers will be placed in a reversed order (relative to their original order).

For example, if `arr = [5, 2, 11, 7, 6, 4]`,

after calling `oddsKeepEvensFlip(arr, 6)`, `arr` will be:

`[5, 11, 7, 4, 6, 2]`

Implementation requirements:

1. Your function should run in **linear time**. That is, if there are n items in `arr`, calling `oddsKeepEvensFlip(arr, n)` will run in $\theta(n)$.
2. Write a `main()` program that tests this function.
3. Note that at the end, the elements should be stored starting at the same base address, as was given in `arr`.

Question 4:

Implement the function:

```
string* createWordsArray(string sentence, int& outWordsArrSize)
```

This function gets a string `sentence` containing a sentence.

When called, it should create and return a new array (of strings), that contains all the words in `sentence`. The function should also update the output parameter, `outWordsArrSize`, with the logical size of the new array that was created.

Note: Assume that the words in the sentence are separated by a single space.

For example, if `sentence="You can do it"`, after calling `createWordsArray(sentence, outWordsArrSize)`, the function should create and return an array that contains `["You" , "can" , "do" , "it"]`, and update the value in `outWordsArrSize` to be `4`.

Implementation requirements:

1. You may want to use some of the string methods, such as `find`, `substr`, etc.
2. Your function should run in linear time. That is, if `sentence` contains n characters, your function should run in $\theta(n)$.
3. Write a `main()` program that tests this function..

Question 5:

In this question, you will write **four versions** of a function `getPosNums` that gets an array of integers `arr`, and its logical size. When called it creates a new array containing only the positive numbers from `arr`.
For example, if `arr=[3, -1, -3, 0, 6, 4]`, the functions should create an array containing the following 3 elements: `[3, 6, 4]`,

The four versions you should implement differ in the way the output is returned.
The prototypes of the functions are:

a) `int* getPosNums1(int* arr, int arrSize, int& outPosArrSize)`
returns the base address of the array (containing the positive numbers), and updates the output parameter `outPosArrSize` with the array's logical size.

b) `void getPosNums2(int* arr, int arrSize, int*& outPosArr, int& outPosArrSize)`
updates the output parameter `outPosArr` with the base address of the array (containing the positive numbers), and the output parameter `outPosArrSize` with the array's logical size.

c) `int* getPosNums3(int* arr, int arrSize, int* outPosArrSizePtr)`
returns the base address of the array (containing the positive numbers), and uses the pointer `outPosArrSizePtr` to update the array's logical size.

d) `void getPosNums4(int* arr, int arrSize, int** outPosArrPtr, int* outPosArrSizePtr)`
uses the pointer `outPosArrPtr` to update the base address of the array (containing the positive numbers), and the pointer `outPosArrSizePtr` to update the array's logical size.

Note: You should also write a program that calls and tests all these functions.

Question 6:

In this question, you will write **two versions** of a program that reads from the user a sequence of positive integers ending with -1, and another positive integer `num` that the user wishes to search for.

The program should then print all the line numbers in sequence entered by the user, that contain `num`, or a message saying that `num` does not show at all in the sequence.

Your program should interact with the user **exactly** as it shows in the following example:

```
Please enter a sequence of positive integers, each in a separate line.
End you input by typing -1.
13
5
8
2
9
5
8
8
-1
Please enter a number you want to search:
5
5 shows in lines 2, 6.
```

a) The first version of the program, is not allowed to use the `vector` data structure.
b) The second version of the program, should use the `vector` data structure.

Implementation requirements (for both programs):

1. Think how to break down your implementation to functions.
2. Your programs should run in **linear time**. That is, if there are n numbers in the input sequence, your program should run in $\theta(n)$.
3. Write the two programs in two functions named `main1()` and `main2()`. Also have the `main()` test these two functions.

Question 7:

Implement the function:

```
int* findMissing(int arr[], int n, int& resArrSize)
```

This function gets an array of integers `arr` and its logical size `n`. All elements in `arr` are in the range $\{0, 1, 2, \dots, n\}$.

Note that since the array contains `n` numbers taken from a range of size $n+1$, there must be at least one number that is missing (could be more than one number missing, if there are duplicate values in `arr`).

When called, it should create and return a new array, that contains all the numbers in range $\{0, 1, 2, \dots, n\}$ that are not in `arr`. The function should also update the output parameter, `resArrSize`, with the logical size of the new array that was created.

For example, if `arr=[3, 1, 3, 0, 6, 4]`, after calling `findMissing(arr, 6, resArrSize)`, the function should create and return an array that contains `[2, 5]`, and update the value in `resArrSize` to be `2`.

Implementation requirements:

1. Your function should run in **linear time**. That is, it should run in $\theta(n)$.
2. Write a `main()` program that tests this function..