Lab Assignment 8
Functions and Basics of Recusion

COL 100

3 May 2021

1 Check Sorted

You are given an array A of size n. Write a function *isSorted*(int A[], int n) which takes as parameters A and n, and returns true if array A is sorted and else returns false.
Note: The input should be taken in the *main*() function. Then *isSorted*() function should be called and the inputs should be passed as parameters. Finally the result returned by *isSorted*() function should be printed in *main*() function.

1.1 Example

1.1.1

Input:

```
6
1 3 4 7 11 13
```

Output:

```
true
```

1.1.2

Input:

```
7
2 3 5 1 6 4 -1
```

Output:

```
false
```

2 Finding Duplicates in an Array

You are given an array of length n. The integers in this array lie in the range of [1,n]. The integers appear once or twice in the array. Your task is to print all the integers that appear twice in your original array. If no duplicates exist, print "No duplicates exist!".
Note: You can perform all the computation in the *main*() function itself and print the result.

2.1 Example

2.1.1

Input:

```
8
2 5 6 7 2 8 9 5
```

Output:

```
2 5
```

2.1.2

Input:

```
6
1 5 7 9 3 4
```

Output:

```
No duplicates exist!
```

3 To Lower Case

Write a function *toLowerCase*() that takes a string as a parameter and returns the same in lowercase.
Note:

- A string is a one-dimensional array of characters terminated by a null character '\0'.
- The input string may contain spaces or special symbols.
- The result returned by *toLowerCase*() function should be printed in the *main*() function.

3.1 Example

3.1.1

Input:

```
HELLO
```

Output:

```
hello
```

3.1.2

Input:

```
Hello World
```

Output:

```
hello world
```

3.1.3

Input:

```
Hello World!
```

Output:

```
hello world!
```

4 Check Substring

You are given two strings s1 and s2. Write a function *isSubstring*(char s1[], char s2[]) which takes two strings as input, and returns the start index if the second string is a substring of the first, else returns -1. (Case Insensitive)
Note: Print the result in the *main*() function.

4.1 Example

4.1.1

Input:

```
Starfish
Star
```

Output:

```
0
```

4.1.2

Input:

```
Starfish
Fish
```

Output:

```
4
```

4.1.3

Input:

```
Exhibition
habit
```

Output:

```
-1
```

5 Pangram

A **pangram** is a sentence where every letter of the English alphabet appears at least once.
You are given a string sentence **containing only lowercase English letters**. Write a function *isPangram*(char sentence[]) that takes the string as input and returns true if the sentence is a pangram, or false otherwise.
Note:

- The string may contain spaces.
- Print the result in the *main*() function.

Example

5.0.1

Input:

```
the quick brown fox jumps over the lazy dog
```

Output:

```
true
```

5.0.2

Input:

```
helloworld
```

Output:

```
false
```

6 Caesar's Cipher

(a) Write a function **void encrypt(char str[], int k)** to encrypt a string using *Caesar's cipher*, and print it.
If p is some plaintext (i.e., an unencrypted message), p_i is the i^{th} character in p, and k is a secret key (i.e., a non-negative integer), then each letter, c_i, in the ciphertext (i.e the encrypted message), c, is computed as:

$$c_i = (p_i + k)\%26$$

The user has to enter the string to be encrypted, and the key. Then each letter in the plaintext is 'shifted' by a number of places equal to the key.

Note: p_i denotes alphabet number from the set 0, ..., 25 with 0 corresponding to 'a' or 'A', 1 corresponding to 'b' or 'B' and so on.
Example:
Input:

```
Hello!
3
```

Output:

```
Khoor!
```

Note: Avoid transformation into ASCII 0..31, i.e. operate only in the ASCII region of 32..127. Also note that in the example, the special characters remain unchanged.
(b) Write another function **void decrypt(char str[], int k)** to decrypt a string using Caesar's cipher, and print it. **Example:**
Input:

```
Khoor!
3
```

Output:

```
Hello!
```

(c) Write a main function, which takes a string and a key as input, and prints the encrypted version using Caesar's cipher. Then it decrypts the string and prints the decrypted version using the above functions.

7 Transpose Matrix

The **transpose** of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.
Given a 2D array of size $n \times n$ as input, write a program to find the transpose of a matrix and print the result.
To read more about transpose operation on a matrix: `https://en.wikipedia.org/wiki/Transpose`
Note: You may perform the task and print the result in the *main*() function itself.

Example:

Input:

```
3
1 2 3
```

```
4 5 6
7 8 9
```

Output:

```
1 4 7
2 5 8
3 6 9
```

8 Friends

You are given a 2D array of size $n \times n$ as input. Each value of the 2D array can either be 0 or 1. Suppose indices represent people and that the value at row i, column j of the array is 1 just in case i and j are friends and 0 otherwise.

(a) Define *main*() function such that it takes as input n, followed by n^2 entries of the 2D array. Perform the tasks defined in parts (b) and (c) and print the results in the format as shown in the examples in the *main*() function itself.

(b) Count how many pairs of friends are represented in the 2D array.

Example:

Input:

```
5
0 1 0 1 1
1 0 1 0 1
0 1 0 0 0
1 0 0 0 1
1 1 0 1 0
```

Output:

```
Number of pairs of friends = 6
```

Explanation: Each friendship pair appears twice in the array, so in the example above there are 6 pairs of friends.

(c) Take as input two indices x and y ($0 <= x, y <= n-1$) and check if they have a common friend. Print true if there is an integer z such that x is a friend of z and z is a friend of y and print false otherwise.

Example:

Input:

```
5

0 1 0 1 1
1 0 1 0 1
0 1 0 0 0
1 0 0 0 1
1 1 0 1 0

0 4
```

Output:

```
true
```

Input:

```
5

0 1 0 1 1
1 0 1 0 1
0 1 0 0 0
1 0 0 0 1
1 1 0 1 0

1 2
```

Output:

```
false
```

Explanation: In the examples above, 0 and 4 are both friends with 3 (so they have a common friend), whereas 1 and 2 have no common friends.

Challenge Problems

9 Unique Occurrences

You are given an array of integers of size n. Write a function *check*() that returns true, if and only if the number of occurrences of each value in the array is unique. For all other cases, it must return false. *Note*: Print the result in the *main*() function.

Example:

Input

```
10
1 2 2 3 3 3 4 4 4 4
```

Output

```
true
```

Explanation: The value 1 has one occurrence, 2 has two occurrences, 3 has three occurrences and 4 has four. **No two values have the same number of occurrences in the array**.

10 Flowerbed Problem

Consider a flowerbed, an arrangement of flowers and pots in a line. You have some of the pots where flowers are planted, and some empty pots with you. Now you have been given the task to plant the flowers in such a way that they cannot be planted in adjacent pots.
If you consider an integer array **flowerbed**[] of size n, 0s would mean empty pots and 1 would mean used pots. Given an integer k as input, write a function **canPlaceFlowers(int flowerbed[], int n, int k)** which checks if k new flowers can be planted in the flowerbed without violating the rule.The function returns true or false statements to demonstrate the above condition.
Note: Print the result in the *main*() function.

10.1 Example

10.1.1

Input:

```
n = 5
flowerbed = 1 0 0 0 1
k = 1
```

Output:

```
true
```

10.1.2

Input:

```
n = 5
flowerbed = 1 0 0 0 1
k = 2
```

Output:

```
false
```

11 Minimum Cost of Climbing Stairs

Suppose you have an array **cost**[] of size n where $cost[i]$ is the cost of the i^{th} step on a staircase. Once you pay the cost, you can either climb one or two steps of the stairs. You can either start from the step with index 0 or step with index 1. Write a function *minCost*(int cost[], int n) that computes and returns an integer value which is the minimum cost that is incurred in reaching the top floor.
Note: Print the result in the *main*() function.

11.1 Example

11.1.1

Input:

```
3
10 15 20
```

Output:

```
Minimum Cost = 15
```

Explanation: Cheapest is: start on cost[1], pay that cost, and go to the top.

11.1.2

Input:

```
10
1 100 1 1 1 100 1 1 100 1
```

Output:

```
Minimum Cost = 6
```

Explanation: Cheapest is: start on cost[0], and only step on 1s, skipping cost[3].
Hint: $minCost[i] = cost[i] + min(minCost[i-1], minCost[i-2])$ for $i \geq 2$.
$minCost[0] = cost[0]$, $minCost[1] = cost[1]$

Submission and other logistics

Submit at least 4 solutions (from the first 8 questions) as a zip file on Gradescope (to your respective group's course). There is no need to submit solutions to the challenge problems. Submit only one .c file for each question. Use separate .c files for each new question. Please name your .c files as per the question number (q1.c, q2.c, ... etc). Following this naming convention will help TAs to figure out where to look for the answers easily. Also add the corresponding screenshots, showing output of your code on certain input examples. You can also submit more than 4 or all questions to increase your chances of full marks.

Example: To zip folder 'a8' as 'a8.zip':

```
zip -r a8.zip a8
```

It is highly **recommended** that you name the code files and variables in those code files with proper names as per the question to easily identify them. Comments in your codes are also highly **encouraged** and makes life easier for everyone.
You can check **2nd Chapter** in NASA's C style guide for styling recommendations

You can work either individually or with another student of your group for the assignment.
only one submission on gradescope is enough for a team but you need to **add your teammate** on gradescope after submission.
Follow these steps for adding your team member
Note: you can change your team for future assignments