Lab 1: Review

1 Pointer

Complete the following functions using the Pointer technique:

1. Input a n-element integer array with `int *a` is the pointer point to the allocated dynamic memory:
   - `void inputArray(int* &a, int &n)`
2. Remove allocated dynamic memory:
   - `void dellocateArray(int* &a)`
3. Output all elements of the array:
   - `void printArray(int* a, int n)`
4. Find the smallest value from the array:
   - `int findMin(int* a, int n)`
5. Find the greatest absolute value from the array:
   - `int findMaxModulus(int* a, int n)`
6. Determine if the array is ascending:
   - `bool isAscending(int* a, int n)`
7. Find the total value of all elements of the array:
   - `int sumOfArray(int* a, int n)`
8. Count the number of prime numbers in the array:
   - `int countPrime(int* a, int n)`
9. Create a new dynamic array which is the reverse of the given array:
   - `int reverseArray(int* &a, int* b, int n)`

   From No. 10. *to No.* 13. *are Searching Algorithms. Return the first position found, else, return* -1.
10. Sequential Search:
    - `int LinearSearch(int* a, int n, int key)`
11. Sequential Search (using flag):
    - `int SentinelLinearSearch(int* a, int n, int key)`
12. Binary Search:
    - `int BinarySearch(int* a, int n, int key)`
13. Binary Search (using recursion):
    - `int RecursiveBinarySearch(int* a, int left, int right, int key)`

2 Recursion

Complete the following functions using the Recursion technique (*you may declare some sub-functions*):

1. Find the total value of all integers that less than or equal to n: $S = 1^2 + 2^2 + ... + n^2$.
   - `int sumOfSquares(int n)`
2. Find the greatest common divisor of 2 integers **a** and **b**:
   - `int GCD(int a, int b)`
3. Determine if a given array is palindrome:
   - `bool isPalindrome(int a[], int n)`
4. Find the Factorial of a number:
   - `int Factorial(int n)`
5. Count the digits of a given number:
   - `int countDigit(int a)`
6. Find the n^{th} Fibonacci number using by the following formula: $F(n) = F(n-1) + F(n-2)$.
   - `int Fib(int n)`

3 File Handling

3.1 Data Description

This lab's data is the anonymized data of the result of the High Graduation Exam 2018 - 2019. The information is provided in the file *"data.txt"*, which has the content as follow:

```
Số Báo Danh, Họ và Tên, Toán, Ngữ Văn, Vật Lý, Hóa Học, Sinh Học, Lịch Sử, Địa Lý, GDCD, KHTN, KHXH, Ngoại Ngữ, Ghi Chú, Tỉnh
BD1200000,,8.6,6.5,4.0,7.25,5.5,,,,,,8.4,N1,BinhDinh
BD1200001,,4.0,5.0,,,,4.25,7.0,7.75,,,2.0,N1,BinhDinh
BD1200002,,7.0,6.25,6.0,6.25,6.5,,,,,,5.2,N1,BinhDinh
BD1200003,,5.2,5.75,,,,5.75,7.25,9.25,,,4.6,N1,BinhDinh
BD1200004,,7.6,6.25,7.0,6.5,4.5,,,,,,6.2,N1,BinhDinh
BD1200005,,8.6,6.5,4.0,7.25,5.5,,,,,,8.4,N1,BinhDinh
```

in which:

- The first line provides the included information fields.
- For the next lines, each one is the information of 1 candidate, separated by a comma ",".
- The empty fields mean there is no information. If the empty field is a subject, that equal to a 0.
- The scores in the fields: Natural Sciences (KHTN) and Social Sciences (KHXH) will be instructed in the next part.

3.2 Programming

Given the `Examinee` data structure definition:

```
// Examinee.h
struct Examinee
{
    string id;
    float math, literature, physic, chemistry, biology, history, geography, civic_education, natural_science,
        social_science, foreign_language;
};
```

Fulfill the following requirements:

1. Read the information of one candidate:
   - `Examinee readExaminee(string line_info);`
   - **Input**: `line_info` - a line from *"data.txt"* which provides the information of 1 contestant.
   - **Output**: Return `Examinee` variable, which stores the info of the given contestant.
2. Read the information of a list of candidates:
   - `vector<Examinee> readExamineeList(string file_name);`
   - **Input**: `file_name` - path to input file *"data.txt"*.
   - **Output**: Return `vector<Examinee>` variable, which store the info of all contestants from the file.
3. Write the total score of candidates to file:
   - `void writeTotal(vector<Examinee> examinee_list, string out_file_name);`
   - **Input**: `examinee_list` - List of contestants.
     `out_file_name` - name of file to write.
   - **Output**: Calculate the total score of each contestant and write them to the *out_file_name* file using the following format:
     - Each line contains info of only one contestant.
     - Each contestant's info consists of ID and the total score separated by a single space.
   - **Example**:
     XX001 42.0
     XX002 38.5
     ...
     XX999 23.25

The total score is calculated as follows:

- The score of Natural Sciences and Social Sciences column in *data.txt* is not available by default. Calculate the score for each combination and store them into struct `Examinee`.
- The score of Natural Sciences combination = physic + chemistry + biology
- The score of Social Sciences combination = history + geography + civic education
- The total score = math + literature + foreign language + natural sciences + social sciences

4 Linkedlist

Given the following Linkedlist definition:

```
struct NODE{
    int key;
    NODE* p_next;
};
```

```
struct List{
    NODE* p_head;
    NODE* p_tail;
};
```

Complete the following functions to fulfill the given requirements:

1. Initialize a `NODE` from a given integer:
   - `NODE* createNode(int data)`
2. Initialize a `List` from a give `NODE`:
   - `List* createList(NODE* p_node)`
3. Insert an integer to the head of a given `List`:
   - `bool addHead(List* &L, int data)`
4. Insert an integer to the tail of a given `List`:
   - `bool addTail(List* &L, int data)`
5. Remove the first `NODE` of a given `List`:
   - `void removeHead(List* &L)`
6. Remove the last `NODE` of a given `List`:
   - `void removeTail(List* &L)`
7. Remove all `NODE` from a given `List`:
   - `void removeAll(List* &L)`
8. Print all elements of a given `List`:
   - `void printList(List* L)`
9. Count the number of elements `List`:
   - `int countElements(List* L)`
10. Create a new `List` by reverse a given `List`:
    - `List* reverseList(List* L)`
11. Remove all duplicates from a given `List`:
    - `void RemoveDuplicate(List* &L)`
12. Remove all `key` value from a given `List`:
    - `bool RemoveElement(List* &L, int key)`

5 Stack - Queue

Following is the representation of a Singly linked list node:

```
struct NODE{
    int key;
    NODE* pNext;
};
```

Utilize the Linked list above, define the data structure of Stack and Queue, then implement functions to execute the following operations:

1. Stack
   - **Initialize** a stack from a given key.
   - **Push** a key into a given stack.
   - **Pop** an element out of a given stack, return the key's value.
   - **Count** the number of elements of a given queue.
   - Determine if a given stack **is empty**.
2. Queue
   - **Initialize** a queue from a given key.
   - **Enqueue** a key into a given queue.
   - **Dequeue** an element out of a given queue, return the key's value.
   - **Count** the number of element of a given queue.
   - Determine if a given queue **is empty**.