

## 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  $n$ :  $S = 1^2 + 2^2 + \dots + n^2$ .

- `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:

```

1  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
2  BD1200000,,8.6,6.5,4.0,7.25,5.5,,,,,8.4,N1,BìnhDinh
3  BD1200001,,4.0,5.0,,,4.25,7.0,7.75,,2.0,N1,BìnhDinh
4  BD1200002,,7.0,6.25,6.0,6.25,6.5,,,,,5.2,N1,BìnhDinh
5  BD1200003,,5.2,5.75,,,5.75,7.25,9.25,,4.6,N1,BìnhDinh
6  BD1200004,,7.6,6.25,7.0,6.5,4.5,,,,,6.2,N1,BìnhDinh
7  BD1200005,,8.6,6.5,4.0,7.25,5.5,,,,,8.4,N1,BìnhDinh

```

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:

- |   |   |
|---|---|
| <ol style="list-style-type: none"> <li>1. Initialize a NODE from a given integer:           <ul style="list-style-type: none"> <li>• NODE* createNode(int data)</li> </ul> </li> <li>2. Initialize a List from a give NODE:           <ul style="list-style-type: none"> <li>• List* createList(NODE* p_node)</li> </ul> </li> <li>3. Insert an integer to the head of a given List:           <ul style="list-style-type: none"> <li>• bool addHead(List* &amp;L, int data)</li> </ul> </li> <li>4. Insert an integer to the tail of a given List:           <ul style="list-style-type: none"> <li>• bool addTail(List* &amp;L, int data)</li> </ul> </li> <li>5. Remove the first NODE of a given List:           <ul style="list-style-type: none"> <li>• void removeHead(List* &amp;L)</li> </ul> </li> <li>6. Remove the last NODE of a given List:           <ul style="list-style-type: none"> <li>• void removeTail(List* &amp;L)</li> </ul> </li> </ol> | <ol style="list-style-type: none"> <li>7. Remove all NODE from a given List:           <ul style="list-style-type: none"> <li>• void removeAll(List* &amp;L)</li> </ul> </li> <li>8. Print all elements of a given List:           <ul style="list-style-type: none"> <li>• void printList(List* L)</li> </ul> </li> <li>9. Count the number of elements List:           <ul style="list-style-type: none"> <li>• int countElements(List* L)</li> </ul> </li> <li>10. Create a new List by reverse a given List:           <ul style="list-style-type: none"> <li>• List* reverseList(List* L)</li> </ul> </li> <li>11. Remove all duplicates from a given List:           <ul style="list-style-type: none"> <li>• void RemoveDuplicate(List* &amp;L)</li> </ul> </li> <li>12. Remove all key value from a given List:           <ul style="list-style-type: none"> <li>• bool RemoveElement(List* &amp;L, int key)</li> </ul> </li> </ol> |
|---|---|

## 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:

- |   |   |
|---|---|
| <ol style="list-style-type: none"> <li>1. Stack           <ul style="list-style-type: none"> <li>• <b>Initialize</b> a stack from a given key.</li> <li>• <b>Push</b> a key into a given stack.</li> <li>• <b>Pop</b> an element out of a given stack, return the key's value.</li> <li>• <b>Count</b> the number of elements of a given queue.</li> <li>• Determine if a given stack <b>is empty</b>.</li> </ul> </li> </ol> | <ol style="list-style-type: none"> <li>2. Queue           <ul style="list-style-type: none"> <li>• <b>Initialize</b> a queue from a given key.</li> <li>• <b>Enqueue</b> a key into a given queue.</li> <li>• <b>Dequeue</b> an element out of a given queue, return the key's value.</li> <li>• <b>Count</b> the number of element of a given queue.</li> <li>• Determine if a given queue <b>is empty</b>.</li> </ul> </li> </ol> |
|---|---|