Lab 01 - Review

1 Pointer

Complete the following functions:

1. Swap 2 given integers.

   ```
   void swap(int* a, int* b);
   ```

2. Calculate the sum of 2 integers.

   ```
   int* sum(int* a, int* b);
   ```

3. Input an array of a size that will be determined at runtime.

   ```
   void inputArray(int* &a, int &n);
   ```

4. Find the largest value from a given array.

   ```
   int* findMax(int* arr, int n);
   ```

5. Find the longest ascending subarray from a given array.

   ```
   int* findLongestAscendingSubarray(int* a, int n, int &length);
   ```

6. Find the contiguous subarray within an array with the largest element sum.

   ```
   int* findLargestSumSubarray(int* a, int n, int &largestSum, int &nRes);
   ```

7. Swap 2 given arrays.

   ```
   void swapArrays(int* &a, int* &b, int &na, int &nb);
   ```

8. Concatenate 2 given arrays.

   ```
   int* concatenate2Arrays(int* a, int* b, int na, int nb, int &nc);
   ```

9. Given 2 ascending arrays with distinguishing elements. Generate a new ascending array with all elements from the given arrays.

   ```
   int* merge2Arrays(int* a, int* b, int na, int nb, int &nc);
   ```

10. Generate a random matrix with a user-defined size (entered at runtime).

    ```
    void generateRandomMatrix(int** &a, int &nRow, int &nCol);
    ```

11. Given two 1D arrays `a` and `b`. Generate the matrix `c` that `c[i][j] = a[i] * b[j]`.

```
int** generateMatrix(int* a, int* b, int na, int nb, int &nRow, &nCol);
```

12. Swap 2 columns/rows of a given matrix.

```
void swapRows(int** a, int nRow, int nCol, int ir1, int ir2);
```

```
void swapColumns(int** a, int nRow, int nCol, int ic1, int ic2);
```

13. Calculate the transpose of the input matrix (resulting in a new matrix).

```
int** transposeMatrix(int** a, int nRow, int nCol);
```

14. Concatenate 2 given size-equal matrices, horizontally/vertically.

```
int** concatenate2MatricesH(int** a, int** b,
                             int nRow, int nCol, int &resRow, int &resCol);
```

```
int** concatenate2MatricesV(int** a, int** b,
                             int nRow, int nCol, int &resRow, int &resCol);
```

15. Multiply two matrices.

```
bool multiplyMatrices(int** a, int** b, int** &res, int aRow, int aCol, int bRow, int bCol,
                      int &resRow, int &resCol);
```

16. Given a matrix, find the submatrix of a specified size (entered at runtime) with the largest sum of elements.

```
int** findSubmatrix(int** a, int nRow, int nCol, int &subRow, int &subCol);
```

Functions from #17 to #20 implement searching algorithms. They return the position of the first match, or −1 if not found.

17. Linear/sequential search.
    - `int linearSearch(int* a, int n, int key);`
18. Sentinel linear/sequential search.
    - `int sentinelLinearSearch(int* a, int n, int key);`
19. Binary search (for a sorted array).
    - `int binarySearch(int* a, int n, int key);`
20. Recursive binary search (for a sorted array).
    - `int recursiveBinarySearch(int* a, int left, int right, int key);`

2 Recursion

Complete the following functions using a recursive approach. You can define helper functions for further breakdown.

1. Calculate the sum of $S = 1 + 2 + 3 + ... + n$.
2. Calculate the factorial $n! = 1 \times 2 \times 3 \times ... \times n$.
3. Calculate x^n.
4. Count the number of digits of a given integer.
5. Verify if every digit of a given integer is even.
6. Count the number of common divisors of 2 given integers.
7. Find the Greatest common divisor (gcd) and Least common multiple (lcm) of 2 given integers.
8. Find the reverse value of a given integer.
9. Calculate the i^{th} Fibonacci number.
   - $F_0 = 0, F_1 = 1$
   - $F_n = F_{n-1} + F_{n-2}, (n \geq 2)$
10. * Generate all permutations for a set of 4 unique characters. For an extra challenge, adapt it to handle an arbitrary number of characters (n).
    - Example: ABCD, ABDC, ACBD,...

3 File Handling

3.1 Data Description

The file *"data.txt"* includes anonymized data of the National High School Graduation Exam 2018 - 2019 results. The following shows the initial lines of the file:

```
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 lists the information fields included in the data.
- For the next lines, each subsequent line represents the information for a single candidate, separated by commas (",").
- Empty fields signify missing data. For subject scores, an empty field is treated as a 0 in this lab.
- Instructions regarding Natural Sciences (KHTN) and Social Sciences (KHXH) scores will be provided in a separate section.

3.2 Programming

Given the `Examinee` data structure definition:

```
// Examinee.h
struct Examinee
{
    string id;
    float math, literature, physics, chemistry, biology, history, geography, civicEducation, naturalScience,
        socialScience, foreignLanguage;
};
```

Fulfill the following requirements:

1. Read the information of one examinee:
   - `Examinee readExaminee(string lineInfo);`
   - **Input**: `lineInfo` - a line from *"data.txt"* which provides the information of 1 examinee.
   - **Output**: Return `Examinee` variable, which stores the info of the given examinee.
   - Note: The scores of Natural Sciences and Social Sciences column in *"data.txt"* is not available by default. Calculate the scores for each combination and store them into struct `Examinee`.
     - The score of Natural Sciences combination = physics + chemistry + biology.
     - The score of Social Sciences combination = history + geography + civic education.
2. Read the information of a list of examinees:
   - `vector<Examinee> readExamineeList(string fileName);`
   - **Input**: `fileName` - path to input file *"data.txt"*.
   - **Output**: `vector<Examinee>` variable, which stores the info of all examinees from the file.
3. Write scores of examinees to file:
   - `void writeScores(vector<Examinee> examineeList, string outFileName);`
   - **Input**: `examineeList` - List of examinees.
     `outFileName` - name of file to write.
   - **Output**: Write to output file Compulsory Scores (BB), Natural Sciences Scores (KHTN), and Social Sciences (KHXH) of each examinee using the following format:
     - The first line contains header information: "SBD BB KHTN KHXH".
     - Each next line contains info of only one examinee: ID, Compulsory Scores, Natural Sciences Scores, and Social Sciences Scores separated by a single space.
     - The Compulsory Scores = maths + literature + foreign language.
   - **Example**:

     SDB BB KHTN KHXH
     XX001 28.0 29.25 0.0
     ...
     XX999 20.0 0.0 28.75

4 Singly Linked List

Given the following singly linked list definition:

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

```
struct List {
    NODE* pHead;
    NODE* pTail;
};
```

Complete the following functions:

1. Initialize a `NODE` from a given integer:
   - `NODE* createNode(int data);`
2. Initialize a `List` from a give `NODE`:
   - `List createList(NODE* pNode);`
3. Insert an integer to the head of a given `List`:
   - `void addHead(List &L, int data);`
4. Insert an integer to the tail of a given `List`:
   - `void 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. Remove node before the node has `val` value in a given `List`:
   - `void removeBefore(List &L, int val);`
9. Remove node after the node has `val` value in a given `List`:
   - `void removeAfter(List &L, int val);`
10. Insert an integer at a position of a given `List`:
    - `void addPos(List &L, int data, int pos);`
11. Remove an integer at a position of a given `List`:
    - `void removePos(List &L, int pos);`
12. Insert an integer before a value of a given `List`:
    - `void addBefore(List &L, int data, int val);`
13. Insert an integer after a value of a given `List`:
    - `void addAfter(List &L, int data, int val);`
14. Print all elements of a given `List`:
    - `void printList(List L);`
15. Count the number of elements `List`:
    - `int countElements(List L);`
16. Create a new `List` by reverse a given `List`:
    - `List reverseList(List L);`
17. Remove all duplicate elements from a given `List`, keeping the first occurrence of each one:
    - `void removeDuplicate(List L);`
18. Remove all `key` value from a given `List`:
    - `void removeElement(List &L, int key);`

5 Doubly Linked List

Given the following doubly linked list definition:

```
struct DNODE {
    int key;
    DNODE* pNext;
    DNODE* pPrev;
};
```

```
struct DList {
    DNODE* pHead;
    DNODE* pTail;

};
```

Implement functions to perform operations on singly linked lists, as shown in section 4.

6 Stack - Queue

Given the following `NODE` definition:

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

Utilize the existing linked list to define the data structures of stacks and queues. Next, implement functions for 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, the key's value will be returned.
   - **Count** the number of elements of a given stack.
   - 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, the key's value will be returned.
   - **Count** the number of elements of a given queue.
   - Determine if a given queue **is empty**.

The End