Lab 1

Review

In this lab, we will review the techniques for programming with pointers, recursion, linked lists, stacks, and queues.

1 Exercise 1: Pointers

Write a program with the following requirements:

1. **Input Array**: Write a function to input an array from the keyboard with a known size.

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

2. **Print Array**: Write a function to print the elements of the array to console.

```
void printArray(int* arr, int n);
```

3. **Find Maximum Value**: Write a function to find the maximum value in the array.

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

4. **Sum of array**: Write a function to calculate the sum of the elements in the array.

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

5. **Concatenate Arrays**: Write a function to concatenate two arrays into a new array.

```
void concatArrays(int* a, int na, int* b, int nb, int* &res, int &nres);
```

6. *** Longest Ascending Subarray**: Write a function to find the longest ascending subarray.

```
void findLongestAscendingSubarray(int* arr, int n, int* &res, int &nres);
```

7. **Main function**: In the `main` function, perform the following tasks:

   - Enter the number of elements in array `a`. Then enter the array `a`.
   - Enter the number of elements in array `b`. Then enter the array `b`.
   - Print the array `c`, which is the concatenation of arrays `a` and `b`.
   - Print the maximum value in the array `c`.
   - Print the sum of elements in the array `c`.
   - * Print the longest ascending subarray in the array `c`.

2 Exercise 2: Pointer to Pointer

Write a program with the following requirements:

1. **Read Matrix from File**: Write a function to read a matrix from a file.

   Return `false` if the file cannot be opened or the file structure is invalid. Else, return `true`.

```
bool readMatrix(const char* filename, int** &matrix, int &rows, int &cols)
```

   Each row of the matrix on a separate line and elements separated by spaces. For example:

   1 2 3

   4 5 6

2. **Print Matrix to File**: Write a function to print a matrix to a file.

```
void printMatrix(const char* filename, int** matrix, int rows, int cols)
```

   Each row of the matrix on a separate line and elements separated by spaces. For example:

   1 2 3

   4 5 6

3. **Matrix Multiplication**: Write a function to multiply two matrices.

   Return a boolean value indicating whether the multiplication is successful or not.

```
bool multiplyMatrices(int** a, int aRows, int aCols,
                      int** b, int bRows, int bCols,
                      int** &res, int &resRows, int &resCols);
```

4. *** Matrix Transposition**: Write a function to calculate the transpose of a matrix.

```
void transposeMatrix(int** matrix, int rows, int cols,
                     int** &res, int &resRows, int &resCols);
```

5. **Main Function**: In the `main` function, perform the following tasks:

   - Read the matrix `a` from the file `matrix_a.txt`.
   - Read the matrix `b` from the file `matrix_b.txt`.
   - Multiply the matrices `a` and `b` to get matrix `c`. Check if the multiplication is successful.
   - Print the resulting matrix `c` to the file `matrix_c.txt` if the multiplication is successful.
   - * Calculate the transpose of matrix `c` and print it to the file `matrix_c_transposed.txt`.

3 Recursion

3.1 Sum

Calculate the sum $S = 1 + 2 + 3 + \ldots + n$, where n is a positive integer.

```
int sum(int n);
```

For example:

Input:

5

Output:

15

3.2 Power

Calculate x^n, where x is a real number and n is a positive integer.

```
double power(double x, int n);
```

For example:

Input:

2 3

Output:

8

3.3 * Fibonacci

Calculate the i^{th} Fibonacci number with the following conditions:

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ (for $n \geq 2$)

```
int fibonacci(int i);
```

For example:

Input:

5

Output:

5

4 Linked List

Consider that each node in the Linked List has the following basic structure:

```
struct Node
{
    int data;
    Node* next;
};
```

Please implement these basic functions and operations as follows:

1. **Traversal nodes**: Print the data of the linked list to console.
2. **Count nodes**: Count and return the total number of nodes in the linked list.
3. **Add head**: Append a new node at the beginning of the linked list.
4. **Add tail**: Append a new node at the end of the linked list.
5. **Remove head**: Delete the head node of the linked list.
6. **Remove tail**: Delete the tail node of the linked list.
7. *** Remove duplicate**: Eliminate duplicate nodes in the linked list.

5 Stack

Utilize the Linked List above, implement the following Stack operations:

1. **push**: push a new item into stack.
2. **pop**: pop the top item from the stack.
3. **top**: get the value of the top item.

6 Queue

Utilize the Linked List above, implement the following Queue operations:

1. **enqueue**: enqueue a new item into queue.
2. **dequeue**: dequeue the front item from the queue.
3. **front**: get the value of the front item.

Submission

Your source code must be contributed in the form of a compressed file and named your submission according to the format `StudentID.zip`. Here is a detail of the directory organization:

```
StudentID
├── Exercise_1.cpp
├── Exercise_2.cpp
├── Exercise_3_1.cpp
├── Exercise_3_2.cpp
├── Exercise_3_3.cpp
├── Exercise_4.cpp
├── Exercise_5.cpp
└── Exercise_6.cpp
```

The end.