Exercise 1

Implementing Stack and Queue from scratch

1 Description

Stack and Queue are two fundamental data structures in programming. Stack allows adding and retrieving elements according to the LIFO (Last In, First Out) principle, while Queue allows adding and retrieving elements according to the FIFO (First In, First Out) principle. In this Exercise 1, you are required to implement these two data structures, using `struct` and `template`.

1.1 Stack (Array version)

Implement a **stack** using **array** with the following variables:

- `items`: data array.
- `top`: index of the top item (`-1` if stack is empty).
- `maxSize`: maximum size of the stack, non-negative integer.
- Other variables if necessary.

and functions:

- `init`: initialize an empty stack with the provided size.
- `copyStack`: initialize a stack from another stack.
- `release`: clean up the stack when not in use anymore.
- `isEmpty`: check the stack is empty or not.
- `push`: an operator to push a new item into stack. Print to the console if the stack is full.
- `pop`: an operator to pop the top item.
- `topValue`: get the value of the top item.
- `print`: print the stack to the console.
- Other functions if necessary.

Please note that the program should be organized into (at least) 3 separate files, including `stack.h` (containing the declaration of the stack), `stack.cpp` (containing the implementation for the stack), and `main.cpp` (to use the stack as a library).
Figure 1 and Figure 2 below are examples of `stack.h` and `stack.cpp` files.

```
#pragma once

template <typename T>
struct Stack
{
    T* items;
    int top;
    unsigned int maxSize;

    void init(unsigned int stackSize);
    void copyStack(const Stack<T> &s);
    void release();

    bool isEmpty();
    void push(T newItem);
    T pop();
    T topValue();

    void print();
};

#include "stack.cpp"
```

Figure 1: Example of header file for stack using array.

```
#include "stack.h"

#include <iostream>

using namespace std;

template <typename T>
void Stack<T>::init(unsigned int stackSize)
{
    // To do
}

template <typename T>
void Stack<T>::copyStack(const Stack<T> &s)
{
    // To do
}
```

Figure 2: Example of implementation file for stack using array.

In `main.cpp`, create a menu to test the stack operators (`pop` and `push`). Run the program with normal test cases, when the stack is empty, and when the stack is full. Capture the results and include them in the **report**.

1.2 Stack (Linked List version)

Implement a **stack** using **linked list** with the following variables:

- `top`: a pointer to the top node of the stack. The node here is a simple struct, including data and a pointer to another node.

  For example:

```
template <typename T>
struct Node
{
    T data;
    Node* next;
};
```

- Other variables if necessary.

and functions:

- `init`: initialize an empty stack.
- `copyStack`: initialize a stack from another stack.
- `release`: clean up the stack when not in use anymore.
- `isEmpty`: check the stack is empty or not.
- `push`: an operator to push a new item into stack. Print to the console if the stack is full.
- `pop`: an operator to pop the top item.
- `topValue`: get the value of the top item.
- `print`: print the stack to the console.
- Other functions if necessary.

Similar to when implementing the stack (array version), please organize the program into (at least) 3 separate files (`stack.h`, `stack.cpp`, and `main.cpp`). In `main.cpp`, create a menu to test the stack operators (`pop` and `push`). Run the program with normal test cases, when the stack is empty, and when the stack is full. Capture the results and include them in the **report**.

Figure 3 below is an example of `stack.h` file for stack using linked list.

```
template <typename T>
struct Stack
{
    struct Node
    {
        T data;
        Node* next;
    };

    Node* top;

    void init();
    void copyStack(const Stack<T> &s);
    void release();

    bool isEmpty();
    void push(T newItem);
    T pop();
    T topValue();

    void print();
};
```

Figure 3: Example of header file for stack using linked list.

1.3 Queue (Array version)

Similar to requirement 1.1, please implement a **queue** using **array** and report carefully.

```
template <typename T>
struct Queue
{
    T* items;
    int front;
    int rear;
    unsigned int count;
    unsigned int maxSize;

    void init(unsigned int queueSize);
    void copyQueue(const Queue<T> &q);
    void release();

    bool isEmpty();
    void enqueue(T newItem);
    T dequeue();
    T frontValue();

    void print();
};
```

Figure 4: Example of header file for queue using array.

1.4 Queue (Linked List version)

Similar to requirement 1.2, please implement a **queue** using **linked list** and report carefully.

```
#pragma once

template <typename T>
struct Queue
{
    struct Node
    {
        T data;
        Node* next;
    };

    Node* front;
    Node* rear;

    void init();
    void release();

    bool isEmpty();
    void enqueue(T newItem);
    T dequeue();
    T frontValue();

    void print();
};

#include "queue.cpp"
```

Figure 5: Example of header file for queue using linked list.

1.5 Recursive versions

During the implementation of the above data structures, there will be functions that use loops. **Convert these loop functions into recursive form**. Analyze the theoretical time complexity and compare the actual execution time when using recursion and loops.
Please record the results in the **report**.

2 Submission

Source code folder and Report file must be contributed in the form of a compressed file (.zip, .rar) and named according to your Student ID. For example:

```
StudentID.zip
├── Source
└── Report.pdf
```

2.1 Source code

The source code folder contains programming files. For example:

```
Source
├── Stack Array
│   ├── stack.h
│   ├── stack.cpp
│   ├── recursive_stack.h
│   ├── recursive_stack.cpp
│   ├── main.cpp
│   └── ...
├── Stack Linked List
├── Queue Array
└── Queue Linked List
```

2.2 Report

The report must fully give the following information:

- Student information (Student ID, Full name, etc.).
- Self-evaluation.
- Detailed experiments.
- Exercise feedback (what you learned, what was difficult).
- The report needs to be well-formatted and exported to PDF. If there are figures cut off by the page break, etc., points will be deducted.
- References (if any).

3 Assessment

No.	Details	Score
1	Stack (Array version)	15%
2	Stack (Linked List version)	15%
3	Queue (Array version)	15%
4	Queue (Linked List version)	15%
5	Recursive versions	20%
6	Report	20%
	Total	**100%**

4 Notices

Please pay attention to the following notices:

- This is an **INDIVIDUAL** assignment.
- Duration: about 2 weeks.
- Any plagiarism, any tricks, or any lie will have a 0 point for the course grade.