C++ 과제 1

큐 스택 구현

로봇게임단

19기 예비인턴

이원준

1. 코드

- 큐 헤더 파일

```
#ifndef BOOLQUEUE_H
#define BOOLQUEUE_H

#include <iostream>

using namespace std;

class BoolQueue {
private:
    struct Node {
        bool data;          // 데이터 저장
        Node* next;         // 다음 노드를 가리키는 포인터
        Node(bool item) : data(item), next(nullptr) {}
    };

    Node* frontNode;        // 큐의 앞쪽 노드를 가리킴
```

```
    Node* rearNode;          // 큐의 뒤쪽 노드를 가리킴
    int size;                // 큐의 크기

public:
    BoolQueue();             // 생성자
    ~BoolQueue();            // 소멸자

    void enqueue(bool item);  // 항목 삽입
    bool dequeue();           // 항목 제거 및 반환
    bool front() const;       // 가장 앞 항목 반환
    bool isEmpty() const;     // 큐가 비어있는지 확인
    int getSize() const;      // 큐의 크기 반환
};

BoolQueue::BoolQueue() : frontNode(nullptr),rearNode(nullptr),size(0) {}

BoolQueue::~BoolQueue()
{
    while (!BoolQueue::isEmpty())
    {
        dequeue();
    }
}

void BoolQueue::enqueue(bool item)
```

```
{
    Node* newNode = new Node(item);
    newNode->next = nullptr;
    if (rearNode == nullptr)
    {
        frontNode = rearNode = newNode;
    }
    else
    {
        rearNode->next = newNode;
        rearNode = newNode;
    }
    size++;
}

bool BoolQueue::dequeue()
{
    if (isEmpty())
    {
        throw std::out_of_range("Queue is empty, cannot dequeue.");
    }

    Node* tempNode = frontNode;
    bool dequeuedData = frontNode->data;
    frontNode = frontNode->next;
```

```
    if (frontNode == nullptr) {
        rearNode = nullptr;
    }

    delete tempNode;
    size--;

    return dequeuedData;
}


bool BoolQueue::front() const
{
    return frontNode->data;
}

bool BoolQueue::isEmpty() const
{
    if(size > 0)
    {
        return false;
    }
    else return true;
}

int BoolQueue::getSize() const
```

```
    {
        if (isEmpty())
        {

            throw std::out_of_range("Queue is empty, cannot dequeue.");
        }
        else return size;
    }




    #endif // BOOLQUEUE_H
```

- 스택 헤더 파일

```
#ifndef BOOLSTACK_H
#define BOOLSTACK_H

#include <iostream>

using namespace std;

class BoolStack {
private:
    struct Node {
        bool data;
```

```
        Node* next;
        Node(bool item) : data(item), next(nullptr) {}
    };

    Node* topNode;      // 스택의 최상단 노드를 가리킴
    int size;           // 스택의 크기

public:
    BoolStack();        // 생성자
    ~BoolStack();       // 소멸자

    void push(bool item);   // 항목 삽입
    bool pop();             // 항목 제거 및 반환
    bool top() const;       // 최상단 항목 반환
    bool isEmpty() const;   // 스택이 비어있는지 확인
    int getSize() const;    // 스택의 크기 반환
};

BoolStack::BoolStack()  : topNode(nullptr), size(0) {}
BoolStack::~BoolStack()
{
    while (!BoolStack::isEmpty())
    {
        pop();
    }
    cout << "~boolstack code worked" << endl;
```

```
}

void BoolStack::push(bool item)
{
    Node* newNode = new Node(item);
    newNode->next = topNode;
    topNode = newNode;
    size++;
}

bool BoolStack::pop()
{
    if (isEmpty())
    {

        throw std::out_of_range("Stack is empty, cannot pop.");
    }
    else
    {
        Node* tempNode = topNode;
        bool poppedData = topNode->data;
        topNode = topNode->next;
        delete tempNode;
        size--;

        return poppedData;
```

```
        }
}

bool BoolStack::top() const
{
        if (isEmpty())
        {

                throw std::out_of_range("Stack is empty.");
        }
        else return topNode->data;
}

bool BoolStack::isEmpty() const
{
        if(size > 0 ) return false;
        else return true;
}

int BoolStack::getSize() const
{
        if (isEmpty())
        {

                throw std::out_of_range("Stack is empty.");
        }
```

```
    return size;
}



#endif // BOOLSTACK_H
```

- 메인 코드

```
#include "BoolStack.h"
#include "BoolQueue.h"

void Test_stack(string command,BoolStack& stack);
void Test_queue(string command,BoolQueue& queue);

int main()
{
    int option;

    cout << "choose option(press 1 to Stack press 0 to Queue) : " ;
    cin >> option;

    if(option == 1) // test stack
    {
        string command;
        BoolStack stack;
```

```
        while(1)
        {
            cout << "command: ";
            cin >> command;
            Test_stack(command, stack);
        }
    }
    else if(option == 0) // test queue
    {
        string command;
        BoolQueue queue;

        while(1)
        {
            cout << "command: ";
            cin >> command;
            Test_queue(command, queue);
        }
    }
    else   // error
    {
        cout << "wrong respond try again" << endl;
    }
}
```

```
void Test_stack(string command, BoolStack& stack)
{
	if(command == "push")
	{
		bool newitem;
		cout << "new_item: ";
		cin >> newitem;
		stack.push(newitem);
	}
	else if(command == "pop")
	{
		cout << stack.pop() << endl;
	}
	else if(command == "top")
	{
		cout << stack.top() << endl;
	}
	else if(command == "isEmpty")
	{
		cout << (stack.isEmpty() ? "true" : "false") << endl;
	}
	else if(command == "getSize")
	{
		cout << stack.getSize() << endl;
	}
}
```

```
void Test_queue(string command, BoolQueue& queue)
{
	if(command == "enqueue")
	{
		bool newitem;
		cout << "new_item: ";
		cin >> newitem;
		queue.enqueue(newitem);
	}
	else if(command == "dequeue")
	{
		cout << queue.dequeue() << endl;
	}
	else if(command == "front")
	{
		cout << queue.front() << endl;
	}
	else if(command == "isEmpty")
	{
		cout << (queue.isEmpty() ? "true" : "false") << endl;
	}
	else if(command == "getSize")
	{
		cout << queue.getSize() << endl;
	}
```

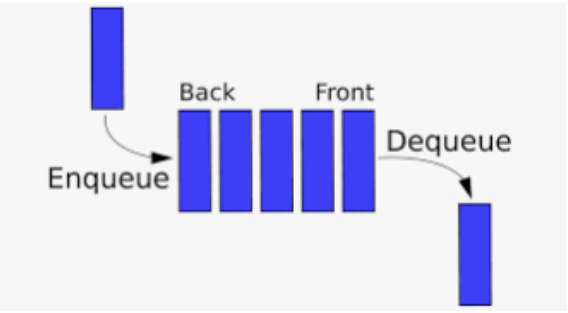
}

2. 코드 설명

2-1 큐 헤더파일

- Enqueue 구현

```
void BoolQueue::enqueue(bool item)
{
    Node* newNode = new Node(item);
    newNode->next = nullptr;
    if (rearNode == nullptr)
    {
        frontNode = rearNode = newNode;
    }
    else
    {
        rearNode->next = newNode;
        rearNode = newNode;
    }
    size++;
}
```



Enqueue를 버그 없이 구현하기 위해 2가지 경우로 나눠서 진행했다.

먼저 첫 번째 경우는 후방 노드를 가르키는 포인터가 널을 가르킬 때, 즉 후방 노드가 없는 큐 자체가 비워진 상황일 때 작동하는 코드이다.

두번째 코드는 반대의 경우로 단순하게 후방노드에 새로운 노드를 연결 시켜준다.

2-2 스택 헤더 파일

- throw std::out_of_range를 활용하여 오류 상황 발생시 적절하게 코드를 정지할 수 있도록 만들었다.

2-3 메인 코드

- 큐와 스택을 각각 개별된 함수로 테스트를 진행 할 수 있게 구성했다.
- 두 함수는 문자열 command와 메인 함수에서 선언된 큐와 스택을 레퍼런스로 가져올 수 있게 파라메터 설정을 했다.

3. 느낀 점 및 고찰

- C++의 다양한 기능들을 활용해 볼 수 있었던 기회였다.
- throw std::out_of_range 나 string같은 기능의 편리함을 경험했다.
- Cmake를 기반으로 코드를 작성할 때 주의해야 할 점이 Cmake는 이전에 빌드한 코드에서 수정점을 찾아 일부만 새로 빌드하므로 디버깅할 때 수정 이후에도 똑같은 문제가 계속 유지된다면 QT를 껐다가 다시 켜보는 것도 하나의 방법이다.