HW_01_19기_예비단원_구도연

<소스 코드>

BoolStack.h

```
#ifndef BOOLSTACK_H
#define BOOLSTACK_H

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
};

#endif
```

BoolQueue.h

```
#ifndef BOOLQUEUE_H
#define BOOLQUEUE_H

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
};

#endif
```

BoolStack.cpp

```cpp
#include "BoolStack.h"
#include <stdexcept>  // 예외 처리를 위한 라이브러리
#include <iostream>

BoolStack::BoolStack() : topNode(nullptr), size(0) {}

BoolStack::~BoolStack() {
    while (!isEmpty()) {
        pop();  // 스택이 비어질 때까지 항목 제거
    }
}

void BoolStack::push(bool item) {
    Node* newNode = new Node(item);   // 새로운 노드 생성
    newNode->next = topNode;          // 새 노드의 next를 기존 topNode로 설정
    topNode = newNode;                // topNode를 새 노드로 갱신
    size++;                           // 스택의 크기 증가
}

bool BoolStack::pop() {
    if (isEmpty()) {
    // 스택이 비어있을 때 예외 처리
        throw std::out_of_range("Stack is empty, cannot pop.");
    }

    Node* tempNode = topNode;        // 현재 최상단 노드를 임시로 저장
    bool poppedData = topNode->data; // 최상단 노드의 데이터를 저장
    topNode = topNode->next;         // 최상단을 그 다음 노드로 변경
    delete tempNode;                 // 이전 최상단 노드 메모리 해제
    size--;                          // 스택 크기 감소

    return poppedData;               // 꺼낸 데이터 반환
}

bool BoolStack::top()const{ // 최상단 요소 반환
    if (isEmpty()) { // 스택이 비어 있을 때 예외처리
            throw std::out_of_range("스택이 비어있어 접근할 수 없습니다.");
        }
        return topNode->data;  // 최상단 노드의 데이터 반환
};

bool BoolStack::isEmpty() const { // 스택이 비었는지 확인
    return size == 0; // size가 0이면 true 반환
};

int BoolStack::getSize() const{ //스택의 현재 크기 반환
    return size; // 스택의 현재 크기 반환
};
```

BoolQueue.cpp

```cpp
#include "BoolQueue.h"
#include <stdexcept> // 예외 처리를 위한 라이브러리
#include <iostream>

//Q의 front, rear을 초기화하고 크기를 0으로 설정
//생성자
BoolQueue::BoolQueue() : frontNode(nullptr), rearNode(nullptr), size(0) {}

//소멸자
BoolQueue::~BoolQueue() {
    while (!isEmpty()) {
        dequeue();  // 큐가 비어질 때까지 항목 제거
    }
}

void BoolQueue::enqueue(bool item) { // 큐에 값 추가
    Node* newNode = new Node(item);  // 새로운 노드 생성
    if (isEmpty()) {
        frontNode = rearNode = newNode; // 큐가 비어있으면 front와 rear 모두 새 노드로 설정
    } else {
        rearNode->next = newNode;       // 기존 rear 노드의 next를 새 노드로 설정
        rearNode = newNode;             // rear 노드를 새 노드로 갱신
    }
    size++;  // 큐 크기 증가
}

bool BoolQueue::dequeue() { // 큐의 맨 앞 요소 제거 이후 반환
    if (isEmpty()) {
        throw std::out_of_range("큐가 비어있습니다."); // 비었을 경우 예외 처리
    }

    Node* tempNode = frontNode;         // 현재 front 노드를 임시로 저장
    bool dequeuedData = frontNode->data; // front 노드의 데이터를 저장
    frontNode = frontNode->next;        // front를 그 다음 노드로 변경
    delete tempNode;                    // 이전 front 노드 메모리 해제
    size--;                              // 큐 크기 감소

    if (isEmpty()) {
        rearNode = nullptr;              // 큐가 비어있으면 rear도 nullptr로 설정
    }

    return dequeuedData;                // 꺼낸 데이터 반환
}

// 큐가 비어있는지 확인
bool BoolQueue::front() const {
    if (isEmpty()) {
        throw std::out_of_range("큐가 비어있습니다.");
    }
    return frontNode->data;             // front 노드의 데이터 반환
}

// 큐가 비어있는지 확인
bool BoolQueue::isEmpty() const {
    return size == 0;                    // 큐 크기가 0이면 true 반환
}
```

```
int BoolQueue::getSize() const {
    return size;                        // 현재 큐 크기 반환
}
```

main.cpp

```
#include <iostream>
#include "BoolStack.h"
#include "BoolQueue.h"

int main()
{
    BoolStack stack; // BoolStack 인스턴스 생성
    BoolQueue queue; // BoolQueue 인스턴스 생성

    int Select1, Select2; // 입력 변수 생성

    std::cout << "원하는 작업을 선택하세요. : (1. Queue 2. Stack ) "; // 작업 선택 입력 받기
    std::cin >> Select1;

    if (Select1 == 2) { // 스택 동작 선택
        while (true) {
            std::cout << "원하는 작업을 선택하세요. :₩n";
            std::cout << "(1. push 2. pop 3. top 4. isEmpty 5. getSize 6. 종료)₩n";
            std::cin >> Select2;

            if (Select2 == 1) { // push
                bool item;
                while (true) {
                    std::cout << "추가할 값을 입력하세요 (0 또는 1): ";
                    std::cin >> item;
                    if (item == 0 || item == 1) { // 입력 값이 0, 1 둘 중 하나인지 확인
                        stack.push(item);
                        std::cout << "값이 추가되었습니다.₩n"; // 스택에 값 추가
                        break; // 유효한 값일 경우 종료
                    } else {
                        std::cout << "유효하지 않은 값입니다. 0 또는 1만 입력.₩n";
                    }
                }
            }

            else if (Select2 == 2) { // pop
                try {
                    bool poppedValue = stack.pop(); // 스택에서 값 꺼내기
                    std::cout << "꺼낸 값: " << poppedValue << "₩n";
                } catch (const std::out_of_range& e) {
                    std::cout << e.what() << "₩n"; // 예외 발생 시 메시지 출력
                }
            }

            else if (Select2 == 3) { // top
                try {
                    bool topValue = stack.top(); // 최상단 값 확인
```

```
                    std::cout << "최상단 값: " << topValue << "₩n";
                } catch (const std::out_of_range& e) {  // 에외 발생 메시지
                    std::cout << e.what() << "₩n";
                }
            }

            else if (Select2 == 4) { // isEmpty
                std::cout << (stack.isEmpty() ? "스택이 비어 있습니다." : "스택에 항목이
있습니다.") << "₩n";
            }

            else if (Select2 == 5) { // getSize
                std::cout << "스택 크기: " << stack.getSize() << "₩n";
            }

            else if (Select2 == 6) { // 종료
                break;
            }

            else {
                std::cout << "지원하지 않는 작업입니다.₩n"; // 잘못된 선택 예외 처리
            }
        }
    }

    if (Select1 == 1) { // 큐 동작 선택
        while (true) {
            std::cout << "원하는 작업을 선택하세요. :₩n";
            std::cout << "(1. enqueue 2. dequeue 3. front 4. isEmpty 5. getSize 6. 종료)₩n";
            std::cin >> Select2;

            if (Select2 == 1) { // enqueue
                bool item;
                while (true) {
                    std::cout << "추가할 값을 입력하세요 (0 또는 1): ";
                    std::cin >> item;
                    if (item == 0 || item == 1) { // 입력 값 확인
                        queue.enqueue(item);
                        std::cout << "값이 추가되었습니다.₩n";
                        break;
                    } else {
                        std::cout << "유효하지 않은 값입니다. 0 또는 1만 입력하세요.₩n";
                    }
                }
            }

            else if (Select2 == 2) { // dequeue
                try {
                    bool dequeuedValue = queue.dequeue(); // 큐에서 값 꺼내기
                    std::cout << "꺼낸 값: " << dequeuedValue << "₩n";
                } catch (const std::out_of_range& e) {
                    std::cout << e.what() << "₩n";
                }
            }
```

```
            else if (Select2 == 3) { // front
                try {
                    bool frontValue = queue.front(); // 가장 앞 값 확인
                    std::cout << "가장 앞 값: " << frontValue << "₩n";
                } catch (const std::out_of_range& e) {
                    std::cout << e.what() << "₩n";
                }
            }

            else if (Select2 == 4) { // isEmpty
                std::cout << (queue.isEmpty() ? "큐가 비어 있습니다." : "큐에 항목이
있습니다.") << "₩n";
            }

            else if (Select2 == 5) { // getSize
                std::cout << "큐 크기: " << queue.getSize() << "₩n";
            }

            else if (Select2 == 6) { // 종료
                break;
            }

            else {
                std::cout << "지원하지 않는 작업입니다.₩n";
            }
        }

    }
    else {
        std::cout << "지원하지 않는 작업입니다.";
    }

    return 0;
}
```

<코드 (주석 외) 부가 설명>

스택과 큐에 값을 넣을 때 0, 1 외의 다른 값을 넣었을 경우 예외 처리를 진행하였으나 코드의 문제인지 작동하지 않는 오류가 발생함. 실제로 0, 1외의 다른 값을 넣었을 때 실행 창에서 오류가 발생하는 것을 확인하였다. 그러나 이 문제를 해결하지는 못함.

<실행 결과.>

** 우분투에서 실행 결과 캡쳐 방법을 몰라서 별도의 실행 결과 사진을 첨부하지 못하였습니다. 죄송합니다.