HW1

1. 소스코드

- boolqueue.h

```
#ifndefBOOLQUEUE_H
#defineBOOLQUEUE_H

classBoolQueue{
private:
  ststructNode{
    boolbooldata;   // 데이터 저장 璢
    Node*nex*next;  // 다음 노드를 가리키는 포인터
    NodeNode(boolitem) : data(item), next(nullptr) {}
  };

  Node*f*frontNode;  // 큐의 앞쪽 노드를 가리킴 ₩ǯ
  Node*r*rearNode;   // 큐의 뒤쪽 노드를 가리킴 ↰ î
  inintsize;      // 큐의 크기

public:
  BoBoolQueue();    // 생성자
  ~B~BoolQueue();    // 소멸자 鰓ʒ

  vovoidenqueue(boolitem);// 항목 삽입
  boobooldequeue();     // 항목 제거 및 반환
  boboolfront() const;   // 가장 앞 항목 반환
  boboolisEmpty() const;  // 큐가 비어있는지 확인 ➔
  inintgetSize() const;  // 큐의 크기 반환 耀
};
#endif// BOOLQUEUE_H
```

- boolstack.h

```
#ifndefBOOLSTACK_H
#defineBOOLSTACK_H
```

```
classBoolStack{
private:
  ststructNode{
    boolbooldata;
    Node*nex*next;
    NodeNode(boolitem) : data(item), next(nullptr) {}//bool: 1바이트 크기의 참/거짓 자료형
  };

  Node*t*topNode; // 스택의 최상단 노드를 가리킴
  inintsize;    // 스택의 크기 鏹ὸY

public:
  BoBoolStack();  // 생성자
  ~B~BoolStack();  // 소멸자 썲

  vovoidpush(boolitem);// 항목 삽입
  boboolpop();      // 항목 제거 및 반환
  booltop() const;  // 최상단 항목 반환 㸙ᅬ
  boboolisEmpty() const;// 스택이 비어있는지 확인
  inintgetSize() const; // 스택의 크기 반환
};
#endif// BOOLSTACK_H
```

- boolqueue.cpp

```
#include"boolqueue.h"
#include<stdexcept>

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

BoolQueue::~BoolQueue() {
  whwhile(!isEmpty()) {
    dequdequeue();// 스택이 비어질 때까지 항목 제거
  } 痃
}
```

```
voidBoolQueue::enqueue(boolitem) {
  Node*n*newNode =newNode(item); // 새로운 노드 생성
  nenewNode->next=nullptr;    // 새 노드의 next를 기존 topNode로 설정

  ifif(isEmpty()) {
    frontNode =new=newNode;
    rearNode =new=newNode;
  } 婀
  elelse{
    rearrearNode->next=newNode;
    rearNode =new=newNode;
  } d
  size++++;             // 스택의 크기 증가
}

boolBoolQueue::dequeue() {
  ifif(isEmpty()) {
  // 큐가 비어있을 때 예외 처리
    throthrowstd::out_of_range("Queue is empty.");

  }

  Node*t*tempNode =frontNode;   // 현재 최상단 노드를 임시로 저장 ή˚`
  boboolpoppedData =frontNode->data;// 최상단 노드의 데이터를 저장
  frontNode =f=frontNode->next;    // 최상단을 그 다음 노드로 변경 ゞడ吓

  ifif(frontNode ==nullptr) {
    rearNode =nul=nullptr;
  }
  dedeletetempNode;        // 이전 최상단 노드 메모리 해제
  size----;            // 스택 크기 감소 밟ж騃믐  return

  rereturnpoppedData;       // 꺼낸 데이터 반환
}
```

```
boolBoolQueue::front() const{
  ifif(isEmpty())
    throthrowstd::out_of_range("Queue is empty.");
  elelse
    retureturnfrontNode->data;
}

intBoolQueue::getSize() const{
  rereturnsize;
}

boolBoolQueue::isEmpty() const{
  rereturnsize ==0;
}
```

- boolstack.cpp

```
#include"boolstack.h"
#include<stdexcept>

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

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

voidBoolStack::push(boolitem) {
  Node*n*newNode =newNode(item); // 새로운 노드 생성
  nenewNode->next=topNode;    // 새 노드의 next를 기존 topNode로 설정 濭  贀蠙
  topNode =n=newNode;       // topNode를 새 노드로 갱신 玥0  谾  t
  size++++;            // 스택의 크기 증가
}

boolBoolStack::pop() {
  ifif(isEmpty()) {
```

```
  // 스택이 비어있을 때 예외 처리
    throthrowstd::out_of_range("Stack is empty.");
  } t

  Node*t*tempNode =topNode;   // 현재 최상단 노드를 임시로 저장
  boboolpoppedData =topNode->data;// 최상단 노드의 데이터를 저장
  topNode =t=topNode->next;    // 최상단을 그 다음 노드로 변경
  dedeletetempNode;        // 이전 최상단 노드 메모리 해제
  size----;           // 스택 크기 감소 뽆0됀틈  return

  rereturnpoppedData;       // 꺼낸 데이터 반환
}

boolBoolStack::top() const{
  ifif(isEmpty())
    throthrowstd::out_of_range("Stack is empty.");
  elelse
    retureturntopNode->data;
}

intBoolStack::getSize() const{
  rereturnsize;
}

boolBoolStack::isEmpty() const{
  rereturnsize ==0;
}
```

- main.cpp

```
#include"boolstack.h"
#include"boolqueue.h"
#include<iostream>
#include<string>

usingnamespacestd;

intmain()
```

```
{
  BoolStack stack;//// 객체 설정
  BoolQueue queue;//// 객체 설정

  inintnumber =0;
  inintstackNumber;// 기능선택 변수
  inintqueueNumber;// 기능 선택 변수
  inintpushNumber;// 스택에 추가할 원소 변수
  inintenqueueNumber;// 큐에 추가할 원소 변수

  whwhile(1) {

    switswitch(number) {
      case0:case0: {// 스택 or 큐 선택
        std::coustd::cout <<std::endl;
        std::coustd::cout <<"1.Stack 2.Queue"<<std::endl;
        std::coustd::cout <<"원하는 기능의 번호를 입력하세요 : ";
        std::cinstd::cin >>number;
        std::coustd::cout <<std::endl;
        break; break;
      }
      case1:case1: {// 스택에서 원하는 기능 선택
        std::coustd::cout <<std::endl;
        std::coustd::cout     <<"1.push     2.pop     3.top     4.empty     5.getSize
6.exit"<<std::endl;
        std::coustd::cout <<"원하는 기능의 번호를 입력하세요 : ";
        std::cinstd::cin >>stackNumber;
        std::coustd::cout <<std::endl;

        switch(sswitch(stackNumber) {// 선택한 기능 작동
          case1:// pcase1:// push
            std::cout <<std::cout <<"몇을 입력 할 것 인가요? (0, 1 이외 모든 숫자는 1로
처리)";
            std::cin >>pstd::cin >>pushNumber;
            stack.push(pstack.push(pushNumber);
```

```
            break; break;
          case2:// pcase2:// pop
            std::cout <<std::cout <<stack.pop() <<std::endl;
            break; break;
          case3://tocase3://top
            std::cout <<std::cout <<stack.top() <<std::endl;
            break; break;
          case4:// ecase4:// empty
            if(stack.isEif(stack.isEmpty())
              std::cout <<"비std::cout <<"비어있습니다."<<std::endl;
            else  谀  婧唄else
              std::cout <<"비std::cout <<"비어있지 않습니다."<<std::endl;
            break; break;
          case5:// gcase5:// getSize
            std::cout <<std::cout <<stack.getSize() <<std::endl;
            break; 核ゝ訌ќ—break;
          case6:// ecase6:// exit
            std::cout <<std::cout <<"exit"<<std::endl;
            number =0; =0;
            break; 刁耀鞣B▒break;
        }
        break; break;
      }
      case2:case2: {// 원하는 기능 선택
        std::coustd::cout <<std::endl;
        std::coustd::cout   <<"1.enqueue   2.dequeue   3.front   4.empty   5.getSize
6.exit"<<std::endl;
        std::coustd::cout <<"원하는 기능의 번호를 입력하세요 : ";
        std::cinstd::cin >>queueNumber;
        std::coustd::cout <<std::endl;

        switch(qswitch(queueNumber) {
          case1:// ecase1:// enqueue
            std::cout <<std::cout <<"몇을 입력 할 것 인가요? (0, 1 이외 모든 숫자는 1로
```

```
처리)";
            std::cin >>estd::cin >>enqueueNumber;
            queue.enqueuqueue.enqueue(enqueueNumber);
            break; ﹎﹎韇ĸ`谀break;
          case2:// dcase2:// dequeue
            std::cout <<std::cout <<queue.dequeue() <<std::endl;
            break; Ą耀顲ъ厶break;
          case3:// fcase3:// front
            std::cout <<std::cout <<queue.front() <<std::endl;
            break;   耀昑沛₌break;
          case4:// ecase4:// empty
            if(queue.isEif(queue.isEmpty())
              std::cout <<"비std::cout <<"비어있습니다."<<std::endl;
            else    耀劗else
              std::cout <<"비std::cout <<"비어있지 않습니다."<<std::endl;
            break; ? 頷㎚ break;
          case5:// gcase5:// getSize
            std::cout <<std::cout <<queue.getSize() <<std::endl;
            break; break;
          case6:// ecase6:// exit
            std::cout <<std::cout <<"exit"<<std::endl;
            number =0; =0;
            break;  霪н³%break;
        }
       break; break;
     } c

    }

  } ₹

  rereturn0;
}
```

2. 코드를 작성할 때 가지고 한 생각

코드를 작성할 때 항상 디버깅하는 과정에 중간중간 들어가야한다고 생각한다. 그래서 기능 하나를 구현하고 바로 디버깅하는 과정을 거치며 코드를 작성하였다.

3. 고찰

겹치는 부분에 대해서 함수화를 진행하였다면 가독성 높은 코드가 될 것 같다는 생각을 하였다.