INHA UNIVERSITY

ICE2004 자료구조론

9

제 목

자료구조론 과제 2 보고서

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2021년 10월 23일

학부

학년

성명 성시열

학번

[개요]

- 파일은 ArrayStack.h 헤더파일과 그 파일의 기능을 활용할 수 있는 ArrayStackTest.cpp 드라이버 파일로 구성되어있다.
- ArrayStack.h 헤더파일은 총 4개의 클래스로 구성되어있다. 예외와 관련된 RuntimeException 클래스와 그 클래스를 상속받는 StackEmptyException, StackFullException 클래스로 구성되어있다. 또한, template를 활용한 ArrayStack 클래스가 존재한다. Stack과 관련된 size(), push(), pop(), top(), empty() 등은 모두 ArryStack 클래스의 멤버함수로 생성하였다.
- ArrayStackTest.cpp 드라이버 파일은 ArrayStack.h 헤더파일을 활용하여 총 3가지 스택을 생성한다. (예외가 없이 size(), push(), pop(), top()이 실행되는 mystack, StackFullException을 발생시키는 mystack1, StackEmptyException을 발생3시키는 mystack2)

[구현상 특징]

- RuntimeException 클래스를 통해 런타임에서 발생하는 예외를 출력할 때 사용한다. 스택이 비어있을 때와 가득 찼을 경우, 이 두가지 예외 상황에 활용이 가능하며 이 두 경우도 StackEmptyException과 StackFullException 클래스를 생성한 뒤 RuntimeException을 상속받아 이 클래스의 멤버들을 사용할 수 있게 했다.

연산자 오버로딩을 통해 << 연산자를 사용하여 RuntimeException 클래스를 data type으로 가지는 문자열 errorMsg를 출력할 수 있게 했다.

또한 inline 함수를 사용함으로써 컴파일 과정에서 함수의 선언을 치환하여 해당 함수가 실행될 때 선언된다. 함수의 선언과 실행이 따로 하지 않아 불필요한 작업을 줄일 수 있다.

Stack과 함께 template를 사용하여 원하는 자료형에 맞게 ArrayStack 클래스를 활용할 수 있다. 실제 코드에서는 stack의 공간을 생성할 때 다양한 자료형의 공간을 만들 수 있었다. top함수에서는 반환형을 const E&로 하여 상황에 맞는 data type을 reference로 반환하였고, const를 통해 수정할 수 없게 하였다. push에서는 parameter을 const E& e로 하여 push를 통해 받는 data type을 다양하게 할 수 있다.

[코드 분석]

1. ArrayStack.h

코드, 목적 및 상세설명

`RuntimeException(const string& err)`
`: errorMsg(err) {`
`}`

parameter을 reference를 통해 값을 그대로 받음. const를 통해 수정을 막음. 생성자를 통해 받은 문자열 err을 errorMsg값으로 지정

```
inline std::ostream& operator << (std::ostream& out, const RuntimeException& e) {
    return out << e.getMessage();
}
```

<< 연산자 오버로딩, << 기호를 활용하여 private의 값인 errorMsg를 return해주는 getMessage함수 출력. inline 함수를 사용함으로써 함수의 선언과 실행이 따로 되는 것이 아니라 컴파일 과정에서 치환되어 해당 함수로 들어가게 됨. 이를 통해 불필요한 과정을 줄여줌.

```
class StackEmptyException : public RuntimeException {
public:
    StackEmptyException(const string& err)
        : RuntimeException(err) {

    }
};
```

Stack 공간이 비어있는데 pop을 통해 제거하려고 하면 예외가 발생한다. 이때 실행될 StackEmptyException 클래스 생성

```
class StackFullException : public RuntimeException {
public:
    StackFullException(const string& err)
        : RuntimeException(err) {

    }
};
```

Stack 공간이 꽉 차있는데 push를 통해 추가하려고 하면 예외가 발생한다. 이때 실행될 StackFullException 클래스 생성

```
template <typename E>
```

template을 활용하여 저장할 변수의 타입이나 클래스를 지정할 수 있다. 아래 코드에서 E를 원하는 datatype으로 지정하여 변수나 함수의 datatype을 바꿀 수 있다.

```
ArrayStack(int cap= DEF_CAPACITY )
        :S(new E[cap]), capacity(cap), t(-1) {

}
```

여기서 t를 -1로 하는 이유는 나중에 empty()에서 음수일 때 비어있다는 조건을 위해 실제로 비어있을 때 t를 음수로 지정.

```
int size() const {
        return (t + 1);
    }
```

t에 1을 더한 이유는 크기는 0부터 시작인데 t를 -1로 초기화했기 때문에 1을 더해 0으로 만든다.

```
throw StackEmptyException
…
throw StackFullException
```

throw를 통해 예외일 경우, 예외를 처리할 코드 catch에 StackEmptyException(혹은 StackFullException) 넘김

2. ArrayStackTest.cpp

코드, 목적 및 상세설명

try {
…
}

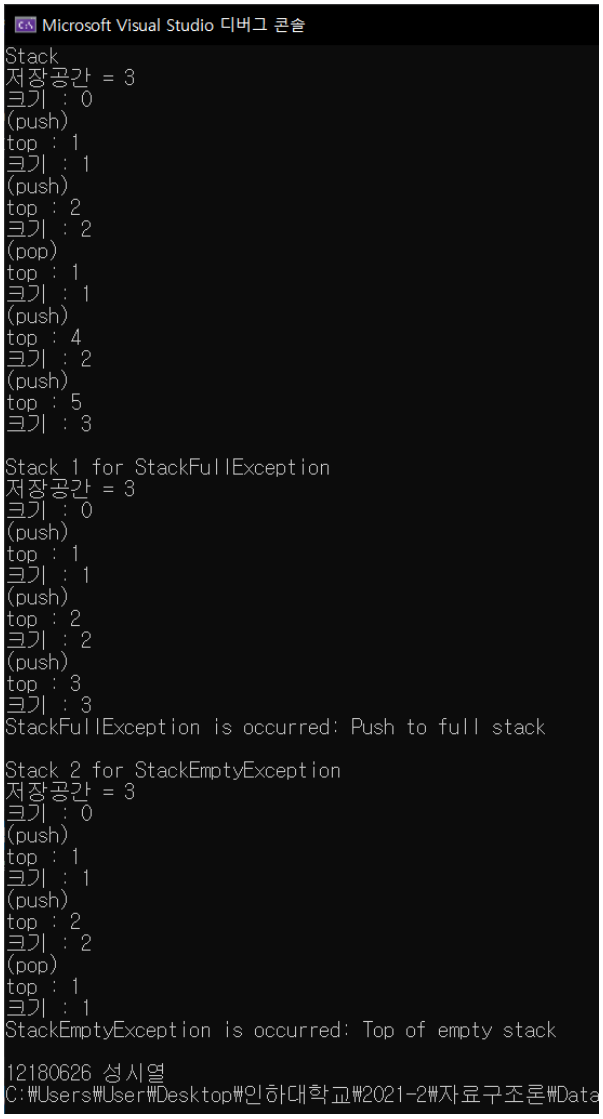
try를 통해 예외에 대한 검사 범위를 지정한다. 이 범위 내에서 예외가 발생하면 그 범위의 catch를 실행한다. catch의 parameter은 throw를 통해 넘겨받는다.

catch (const StackEmptyException& e) {
cout << "StackEmptyException is occurred: " << e << endl;
}
catch (const StackFullException& e) {
cout << "StackFullException is occurred: " << e << endl;
}

예외가 발생했을 시 catch의 코드를 실행, 각각의 함수 내에서 예외의 상황에 맞게 해당 클래스에 어떠한 값을 넣을지 지정함.
ArrayStack<int> mystack(stackSize); …
mystack은 예외상황이 없는 stack으로 push(), pop(), top(), size() 등 함수가 잘 작동되는지 확인하기 위한 코드.
ArrayStack<int> mystack1(stackSize); …
mystack1은 push()에서 StackFullException을 throw하는 예외 상황을 확인하기 위한 코드. 초기에 저장공간을 3으로 하여 stack을 생성하고, 총 4번의 push()를 실행, 꽉 찬 stack 공간에 4번째 push()이 실행될 때, 예외가 발생하여 catch문이 실행된다.
ArrayStack<int> mystack2(stackSize); …
mystack2는 pop에서 StackEmptyException을 throw하는 예외 상황을 확인하기 위한 코드. 초기에 저장공간을 3으로 한 뒤 2번의 push()를 실행 후, 3번의 pop()을 실행, 빈 stack 공간에서 3번째 pop()이 실행될 때, 예외가 발생하여 catch문이 실행된다.

이 외의 코드에 대한 설명은 주석으로 작성함.

[실행결과분석]



먼저 mystack을 나타낼 "Stack" 출력, 사용자가 지정한 stack의 크기를 출력한다.
처음의 size()를 실행했을 때는 mystack에 아무런 값을 넣지 않았으므로 0이 출력된다.
push() 함수를 통해 top에 1이 들어가고 크기가 1이 증가한다.
Pop() 함수를 통해 top 위치의 값이 제거 되고 크기가 1이 감소된다. top의 값이 제거될 때 그 아래있던 값이 top 위치로 이동한다.
mystack1과 mystack2에서도 push()와 pop()은 위와 같이 사용된다.
mystack2의 저장공간이 가득 찼을 때 push()를 실행하면 예외 상황이 발생했으므로 catch문으로 이동하여 해당되는 문장을 실행한다.
mystack2의 저장공간이 비어 있을 때 pop()을 실행하면 예외 상황이 발생했으므로 catch문으로 이동하여 해당되는 문장을 실행한다.
이후 학번 이름을 출력한다.

[결론 및 느낀점, 어려웠던 점]

자료구조론 두 번째 과제를 완성하기 위해 코드를 작성해보면서 객체지향프로그래밍의 요소 중 하나인 stack에 대해 알게 되었고, 그와 관련된 함수들도 알게 되었다. 처음에 설명을 들을 때는 어려운 듯 했으나 코드를 직접 사용해보면서 stack에서 사용할 수 있는 함수들은 비교적 간단하고 행동양식도 직관적이라 쉽게 관련 코드를 작성할 수 있었다. 또한, 스택을 사용할 때 필요한 템플릿의 사용방법에 대해서도 알게 되었다. 같은 기능을 하는 함수들인데 자료형이 다다를 때 쓸 수 있는 구문인 것까진 알고 있었는데, 실제로 구현을 해보면서 템플릿에 대한 이해도를 높일 수 있었다. 이러한 두가지 요소는 교수님께서 간단하시며 설명해주실 때는 처음 접하는 구문이라 잘 와 닿지 못했는데, 과제를 통해 직접 구현을 해보면서 해당 기능에 대해 다룰 수 있게 되었다.

이 코드를 작성하면서 가장 어려웠던 점은 바로 예외와 관련된 구문이었다. 강의노트에 있는 예제들은 쉽게 이해가 되는 듯 했으나, 직접 코드를 작성할 때는 쉽지 않았다. 예외도 위에서 말했던 구문들과 같이 처음 접하는 구문이었는데, 스택과 템플릿에 대한 개념이 잘 잡히지 않은 상태에서 예외에 대한 구문을 작성하려다보니 어려움이 있었다.

두번째 과제를 하며 이러한 구문들을 통해 쇼핑몰의 장바구니와 같은 서비스를 구현할 수 있다는 생각도 해보았다. 장바구니라는 stack에 상품을 추가할 때마다 push, 장바구니에서 상품을 제거할 때 pop을 실행하고, 장바구니의 최대 품목을 넘어서 추가를 할 시에 오류 메세지를 출력하는 등, 추후에 서비스를 개발할 때 유용하게 사용될 기능일 것 같다는 생각을 해보았다.