REPORT #3



|  |
| --- |
| **제 출 일** : 2019년 4월 5일 |
| **과 목 명** : 자료구조및실습 |
| **담당교수 : 김희철 교수님** |
| **학 과** : 컴퓨터전자시스템공학부 |
| **학 번** : 201803818 |
| **성 명 : 한아영** |

**HANKUK UNIVERSITY OF FOREIGN STUDIES**



**Problem Definition**

괄호 검사 프로그램 (입력으로 주어지는 문자들에 대하여 괄호들이 짝이 맞는지 검사

1. 괄호를 검사하여 짝이 맞을 경우 (balanced) 1을 출력, 짝이 맞지 않는 경우 (unbalanced) -1을 출력하는 프로그램
2. 괄호를 검사하여 짝이 맞을 경우 (balanced) 1을 출력, 짝이 맞지 않을 경우 처음으로 짝이 맞지 않은 괄호가 존재하는 line의 줄과 그 괄호에 해당하는 error 번호를 출력하는 프로그램 출력\_예) error 3 at line 2

**Solution**

**1)**

**Stack 클래스**에서는 우리가 구현할 stack에 가장 위에 위치한 변수를 가리키는 int 형 변수 top을 선언하였고, stack을 구현하기 위해 ItemType형 배열 c\_data를 선언하였습니다.

스택이 비어 있는지, 꽉 차 있는지 판단하기 위해 is\_empty , is\_full함수를 구현하였으며 이는 모두 bool형으로 return 합니다.

ItemType 형 배열에 ItemType형 data를 저장할 void형 함수 push ,

ItemType 형 배열에 top위치에 있는 data를 반환하고 배열에서 지우는 것처럼 동작하는 함수 ItemType형 pop을 구현하였습니다.

push함수는 ItemType형 자료를 저장하기 전에 is\_full함수로 스택이 포화상태인지 판단하고 ,

pop함수는 반환하기 전 is\_empty함수로 스택이 비어 있는지 판단하고 top 값을 -1 하는 것으로 배열에서 지우는 것처럼 구현합니다.

**main함수**에서는 입력 받은 괄호가 짝이 맞는지, 아닌지 저장하기 위한 balanced 변수를 bool형으로 선언하였습니다.

짝이 맞는지 검사하기 위해 문자 하나하나 저장할 변수를 char 형으로 symbol으로, 오른쪽 괄호가 나올 때 stack에 top에 위치한 왼쪽 괄호를 pop할 것인데 이때 이 왼쪽 괄호를 저장할 변수를 open\_symbol로 하고 char형으로 선언하였고 여는 괄호를 저장하기 위해 Stack 클래스 s를 선언하였습니다.

Typedef로 int형을 ItemType형으로 쓰도록 합니다.

문자를 하나씩 받기 위해 cin.get()함수를 쓸 것입니다.

while문에서 cin.get()함수로 문자를 계속 받으며 balanced가 True 상태일동안 무한 loop을 돌게 합니다.

Symbol의 값에 따라 if – else 구문으로 case 분류를 합니다.

(여기서 말하는 스택은 Stack클래스 s 배열 c\_data)

1. 여는 괄호 ( , { , [ 일 경우 push함수로 스택에 집어넣습니다. (c\_data 배열에 값 저장)
2. 닫는 괄호일 경우 먼저 스택이 비어 있다면 왼쪽 괄호보다 오른쪽 괄호가 먼저 나온 것이므로 balanced는 false상태가 됩니다.

비어 있지 않다면 스택의 top 위치에 있는 데이터를 open\_symbol에 저장하고 이 open\_symbol과 symbol을 비교해 짝이 맞는지 판단합니다.

1. 출력은 3가지로 나뉩니다.

첫번째로 balanced가 false일 때는 오른쪽 괄호에 의한 에러입니다. ( -1 )

Balanced가 true이면서 스택이 비어 있다면 이는 모든 괄호들이 짝이 맞는 것입니다.( 1 )

그렇지않은 경우는 왼쪽 괄호에 의한 에러입니다. ( -1 )

**2)**

기본 바탕은 1)이 됩니다.

Stack 클래스는 1)과 동일하며 main함수는 1)에서 다음 몇가지를 추가하였습니다.

symbol이 몇 번째 줄에 있는지 저장하기 위한 int형 변수 line을 선언하였습니다.

짝이 안 맞는 경우를 여러가지로 나누어 각각의 error 상황을 저장할 int 형 변수 errortype,

왼쪽 괄호에 의한 에러는 출력할 때 어떤 괄호가 남아있는지 확인하기 위해 괄호를 반환받을 char형 변수 errorcheck를 선언하였습니다. 에러가 몇 번째 줄에서 일어났는지를 저장할 int형 변수 line을, 각 괄호의 line을 저장할 Stack 클래스 l을 선언하였습니다.

while문의 기본 틀은 1)과 비슷합니다.

다만 2)에서는 몇 번째 line인지 알 필요가 있으므로 symbol이 줄 바꿈 기호일때는 line값을 1씩 올려줍니다.

symbol이 여는 괄호 (, { , [ 일 경우 push함수로 s 스택에 저장합니다. 동시에 현재의 line 값도 l 스택에 저장합니다.

symbol이 닫는 괄호 일 경우엔

1. Stack s 가 is\_empty하다면 balanced를 false 상태로 바꾸며 각각의 괄호 case 에 대해 errortype을 지정해줍니다. 그리고 errorline에 현재의 line값을 저장합니다.
2. is\_empty가 false라면 balanced인지 검사한 후 balanced라면 Stack l 에서 pop합니다.

(balanced인 여는 괄호는 Stack s에서 pop되기 때문에 해당 괄호에 대한 line 값을 저장하고 있을 필요가 없습니다.)

balanced하지 않다면 위와 같이 errorline과 errortype을 저장해줍니다.

출력은 1)과 비슷하되 errortype과 errorline을 같이 출력해야합니다.

1. balanced가 false인 경우 (오른쪽 괄호에 의한 에러)

* 변수 errortype과 errorline을 출력하면 됩니다.

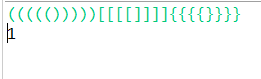
1. balanced가 true이지만 Stack s가 is\_empty 하다면 1을 출력합니다.
2. 위의 상황에 속하지 않는다면 그렇지 않다면 (왼쪽 괄호에 의한 에러)

* 사실 이 때 balanced값은 true입니다. 하지만 Stack s가 비어 있지 않기 때문에 완전히 balanced라고 할 수 없습니다. Stack s 에 남아있는 여는 괄호를 errorcheck 변수에 반환 받아서 각각의 괄호에 대해 errortype을 지정해줍니다.
* Stack s에 남아있던 여는 괄호에 대해 Stack l에도 이 여는 괄호에 대한 line 정보가 남아있으므로 Stack l 에서 pop하여 출력합니다.

**Result**

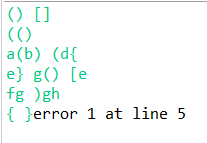
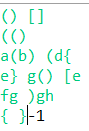
1)

<결과1>



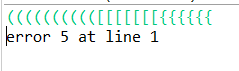
<결과2>

2)

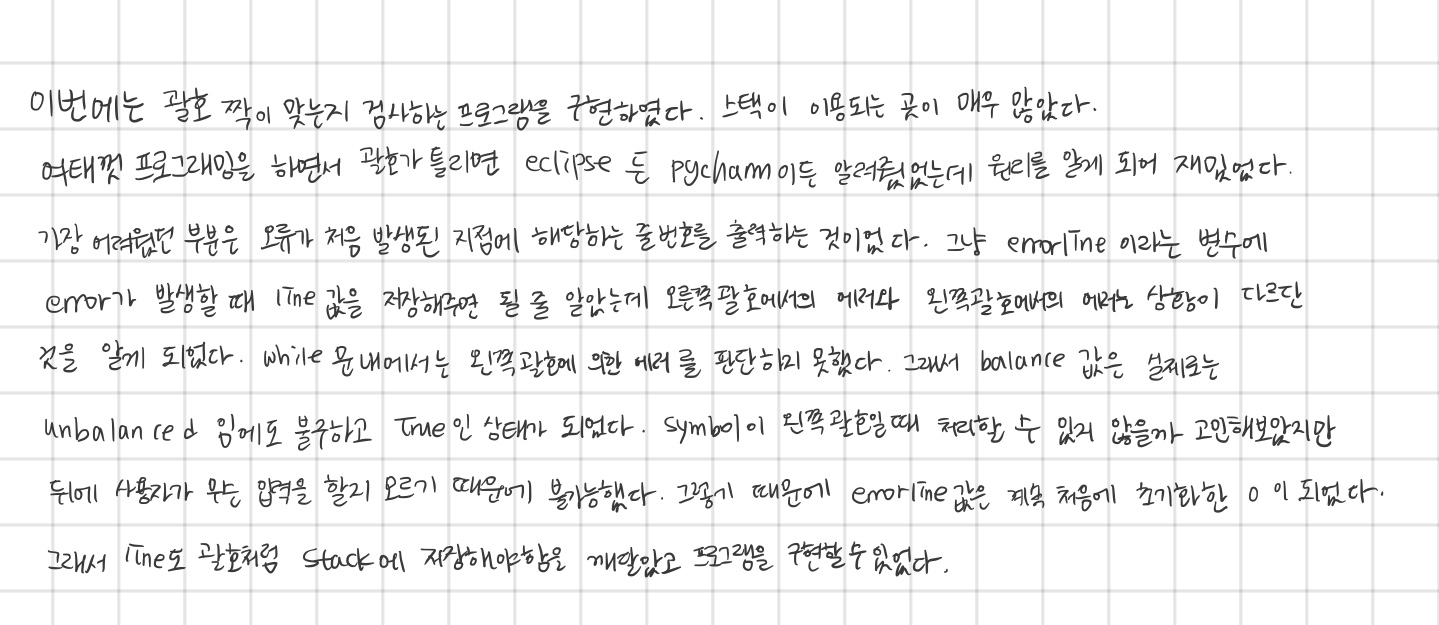


<결과 1>

<결과 2>



**Conclusion**



**Source Code**

1)

**#include**<iostream>

**#include**<cstdlib>

**using** **namespace** std;

**const** **int** MAXS = 100;

**typedef** **int** ItemType;

**class** Stack{

**int** top; //구현할 스택에서 가장 마지막에 저장 된 원소의 index

ItemType c\_data[MAXS]; //데이터를 저장할 배열

**public**:

**Stack**();

**~Stack**();

**bool** **is\_empty**(); //스택이 비어있는지 검사하는 함수

**bool** **is\_full**(); //스택이 포화상태인지 검사하는 함수

**void** **push**(ItemType item); //스택에 새로운 data를 저장하는 함수

ItemType **pop**(); //스택의 데이터를 반환하고 지우는 함수

};

**int** **main**(){

Stack s;

**bool** balanced = **true**; //괄호가 짝이 맞는지에 대한 상태를 나타내는 변수

**char** symbol; //사용자로부터 입력받을 문자를 저장하는 변수

**char** open\_symbol; //여는 괄호를 저장할 변수

**while**(cin.get(symbol)&&balanced){ //입력을 계속 받으며 balanced일 동안 무한 loop

**if**((symbol=='(')**or**(symbol=='{') **or** (symbol=='[')){ //여는 괄호이면 스택에 push

s.push(symbol);

}

/\* 닫는 괄호일때는

\* 스택이 포화상태라면 balanced = false;

\* 아니라면 스택에서 pop한 후 짝이 맞는지 검사

\*/

**else** **if**(symbol==')'){

**if**(s.is\_empty()){

balanced=**false**;

}

**else**{

open\_symbol=s.pop();

balanced = (open\_symbol == '(' && symbol ==')');

}

}

**else** **if**(symbol=='}'){

**if**(s.is\_empty()){

balanced=**false**;

}

**else**{

open\_symbol=s.pop();

balanced = (open\_symbol =='{' && symbol=='}');

}

}

**else** **if**(symbol==']'){

**if**(s.is\_empty()){

balanced=**false**;

}

**else**{

open\_symbol=s.pop();

balanced = (open\_symbol == '[' && symbol == ']');

}

}

}

**if**(!balanced){ //오른쪽 괄호에 의한 unbalanced

cout<<"-1"<<**endl**;

}

**else** **if**(s.is\_empty()){ //balanced

cout<<"1"<<**endl**;

}

**else**

cout<<"-1"<<**endl**; //왼쪽 괄호에 의한 unbalanced

**return** 0;

}

**Stack::Stack**(){

top = -1;

}

**Stack::~Stack**(){}

**bool** **Stack::is\_empty**(){

**return** top==-1; //top 이 -1 일때 true 반환

}

**bool** **Stack::is\_full**(){

**return** top==MAXS -1;//top이 MAXS-1일때 true 반환

}

**void** **Stack::push**(ItemType item){

**if**(is\_full()){

cout<<"스택 포화 오류"<<**endl**;

**exit**(1);

}

top++;

c\_data[top]=item;

}

ItemType **Stack::pop**(){

**if**(is\_empty()){

cout<<"스택 비어있음 오류" <<**endl**;

**exit**(1);

}

ItemType item;

item = c\_data[top];

top--;

**return** item;

}

2)

**#include**<iostream>

**#include**<cstdlib>

**using** **namespace** std;

**const** **int** MAXS = 100;

**typedef** **int** ItemType;

**class** Stack{

**int** top; //구현할 스택에서 가장 마지막에 저장 된 원소의 index

ItemType c\_data[MAXS]; //데이터를 저장할 배열

**public**:

**Stack**();

**~Stack**();

**bool** **is\_empty**(); //스택이 비어있는지 검사하는 함수

**bool** **is\_full**(); //스택이 포화상태인지 검사하는 함수

**void** **push**(ItemType item); //스택에 새로운 data를 저장하는 함수

ItemType **pop**(); //스택의 데이터를 반환하고 지우는 함수

};

**int** **main**(){

Stack s; //괄호를 저장할 스택

Stack l; //line을 저장할 스택

**bool** balanced = **true**; //괄호가 짝이 맞는지에 대한 상태

**char** symbol; //사용자로부터 입력받을 문자를 저장할 변수

**char** open\_symbol; //여는 괄호를 저장할 변수

**int** line=1; //현재 몇번째 줄인지 저장할 변수

**int** errortype=0; //어떤 에러인지 알려주는 변수

**char** errorcheck=0; //어떤 에러인지 구별하기 위해 사용할 변수

**int** errorline=0; //몇번째 줄에서 에러가 났는지 저장할 변수

**while**(cin.get(symbol)&&balanced){ //입력을 계속 받으며 balanced일동안 무한 loop

**if**(symbol=='\n'){ //줄바꿈 문자일 경우 line++

line++;

}

**if**((symbol=='(')**or**(symbol=='{') **or** (symbol=='[')){ //여는 괄호일 경우 symbol과 line을 각각 s와 l에 push

s.push(symbol);

l.push(line);

}

**else** **if**(symbol==')'){

**if**(s.is\_empty()){

balanced=**false**;

errortype = 1;

errorline = line;

}

**else**{

open\_symbol=s.pop();

balanced = (open\_symbol == '(' && symbol ==')');

**if** (balanced){

l.pop();

}

**else**{

errorline=line;

errortype=1;

}

}

}

/\*

\* 닫는 괄호일 경우

\* ㄱ) 스택이 비어있다면 balanced는 false이며 각각의 괄호에 해당하는 errortype을 저장하고 현재 line을 errorline에 저장

\* ㄴ) 비어있지 않다면 s에서 pop해 짝이맞는지 검사함

\* ㄷ) 짝이 맞다면 l에서 pop (짝이 맞는 괄호는 이미 pop해서 지워진 상태이기 때문에 그 괄호에 맞는 line도 pop해줘야함)

\* ㄹ) 짝이 맞지 않다면 errorline과 errortype을 저장함

\*/

**else** **if**(symbol=='}'){

**if**(s.is\_empty()){

balanced=**false**;

errortype = 2;

errorline = line;

}

**else**{

open\_symbol=s.pop();

balanced = (open\_symbol =='{' && symbol=='}');

**if** (balanced){

l.pop();

}

**else**{

errorline=line;

errortype=2;

}

}

}

**else** **if**(symbol==']'){

**if**(s.is\_empty()){

balanced=**false**;

errortype =3;

errorline = line;

}

**else**{

open\_symbol=s.pop();

balanced = (open\_symbol == '[' && symbol == ']');

**if**(balanced){

l.pop();

}

**else**{

errorline =line;

errortype= 3;

}

}

}

}

**if**(!balanced){ //오른쪽괄호에 의해 unbalanced일 경우, errorline을 이용해 어디서 에러가 났는지 출력

cout<<"error "<<errortype<<" at line "<<errorline<<**endl**;

}

**else** **if**(s.is\_empty()){ //balanced인 경우

cout<<"1"<<**endl**;

}

**else**{ //왼쪽 괄호에 의한 unbalanced

errorcheck = s.pop(); //s에 남아있는 여는 괄호를 pop해 errortype을 정한다.

**if** (errorcheck == '('){

errortype = 4;

}

**else** **if** (errorcheck == '{'){

errortype = 5;

}

**else** **if** (errorcheck == '['){

errortype = 6;

}

cout<<"error "<<errortype<<" at line "<<l.pop()<<**endl**; //l에 남아있는 괄호에대한 line정보를 pop한다.

}

**return** 0;

}

**Stack::Stack**(){

top = -1;

}

**Stack::~Stack**(){}

**bool** **Stack::is\_empty**(){

**return** top==-1; //top 이 -1 일때 true 반환

}

**bool** **Stack::is\_full**(){

**return** top==MAXS -1;//top이 MAXS-1일때 true 반환

}

**void** **Stack::push**(ItemType item){

**if**(is\_full()){

cout<<"스택 포화 오류"<<**endl**;

**exit**(1);

}

top++;

c\_data[top]=item;

}

ItemType **Stack::pop**(){

**if**(is\_empty()){

cout<<"스택 비어있음 오류" <<**endl**;

**exit**(1);

}

ItemType item;

item = c\_data[top];

top--;

**return** item;

}