REPORT #3

|  |
| --- |
| **제 출 일** : 2019년 4월 12일 |
| **과 목 명** : 자료구조및실습 |
| **담당교수 : 김희철 교수님** |
| **학 과** : 컴퓨터전자시스템공학부 |
| **학 번** : 201803818 |
| **성 명 : 한아영** |

**HANKUK UNIVERSITY OF FOREIGN STUDIES**



**Problem Definition**

연결리스트로 구현한 스택을 활용한 괄호 검사 프로그램

* 괄호 검사 프로그램은 괄호를 검사하여 짝이 맞을 경우 1을, 맞지 않을 경우 -1을 출력함
* 괄호들을 스택에 저장할 것인데 이때 스택은 연결리스트로 구현되어 있음

**Solution**

1. 연결리스트를 구현하기 위해 Node가 필요합니다. 이때 Node를 구조체로 선언하였습니다. 구조체 내에서는 데이터(괄호)를 저장할 ItemType형 변수 info, 다음 Node를 가리킬 Node형 포인터 next가 선언되었습니다.
2. main함수 (괄호 검사 프로그램이 구현되는 부분)는 과제 3-1가 과 같습니다.
3. Stack 클래스에서는 top위치에 있는 node를 가리키기 위한 Node형 포인터 topPtr을 선언하였습니다. 연결리스트에서는 스택이 포화상태인지 검사할 필요가 없기에 3-1과 달리 is\_full()함수를 제외하였습니다.

생성자에서는 topPtr을 NULL로 초기화해주었습니다.

소멸자에서는 동적으로 할당한 메모리들을 delete하기 위해 while문으로 topPtr이 가리키는 게 없을 때까지 loop을 돌게 됩니다. 이때 Node형 포인터 tempPtr이 topPtr이 가리키던 것을 가리키면 topPtr은 원래 가리키건 것의 next가 가리키고 있던 것을 가리키게 됩니다. 이것이 NULL이라면 다음에 while문은 종료됩니다. tempPtr이 가리키던 것을 delete합니다.

topPtr이 NULL이라면 스택은 비어있다는 의미입니다. 이것이 is\_empty()함수입니다.

스택에 push하기 위해서는

새로운 node를 생성하기 위해 Node형 포인터 newNode를 선언합니다.

이 포인터로 새로운 node를 동적할당 합니다.

할당한 node의 info에 스택에 쌓을 데이터를 저장합니다.

할당한 node의 next는 topPtr이 가리키던 node를 가리키게 됩니다.

새로 할당한 node는 topPtr이 가리키게 됩니다.

스택에서 pop하기 위해서는

반환할 노드 속 데이터를 저장할 ItemType형 변수 item을 선언합니다.

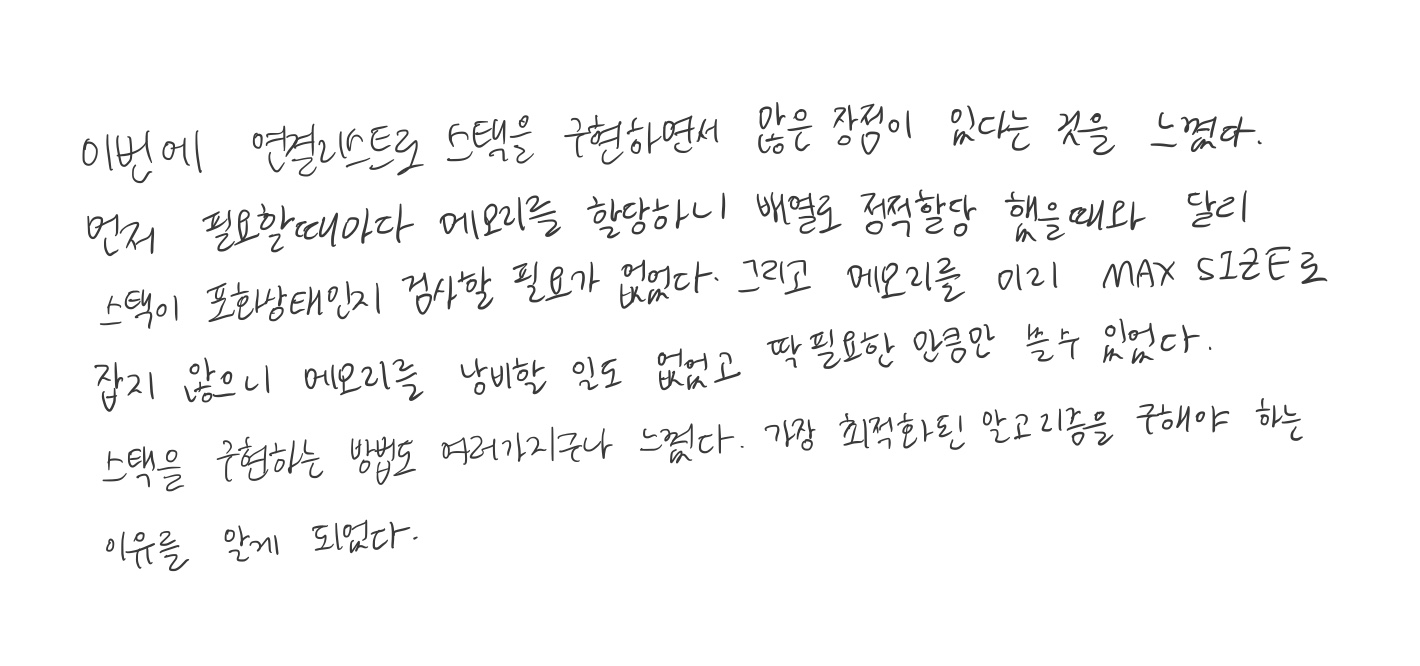
Node형 포인터 tempPtr을 선언합니다.

tempPtr은 topPtr이 가리키던 node를 가리킵니다.

topPtr은 원래 가리키던 node의 다음 node를 가리킵니다.

tempPtr이 가리키던 node를 delete하고 저장해두었던 item을 return합니다.

**Conclusion**



**Source Code**

#include<iostream>

#include<cstdlib>

using namespace std;

typedef char ItemType;

struct Node{

ItemType info; //데이터를 저장

Node\* next; // 다음 노드를 가리킬 포인터

};

class Stack{

private:

Node\* topPtr;//구현할 스택에서 가장 마지막에 저장 된 원소의 index

public:

Stack();

~Stack();

bool is\_empty(); //스택이 비어있는지 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(){

topPtr = NULL;

}

Stack::~Stack(){

Node \*tempPtr;

while (topPtr != NULL){ //topPtr이 가리키는 것이 NULL이 아닐 동안 하나씩 node를 delete함

tempPtr = topPtr;

topPtr = topPtr->next;

delete tempPtr;

}

}

bool Stack::is\_empty(){

return topPtr == NULL;

}

void Stack::push(ItemType item){

Node \*newNode;

newNode = new Node; //동적메모리 할당

newNode->info = item; //할당한 node의 info에 데이터를 저장

newNode->next = topPtr; //할당한 node의 next가 원래 topPtr이 가리키던 node를 가리킴

topPtr = newNode; //새로 할당한 node를 topPtr이 가리킴

}

ItemType Stack::pop(){

ItemType item;

Node \*tempPtr;

item = topPtr->info; //delete하고 난 후에 값을 반환해야 하므로 미리 저장함

tempPtr = topPtr;

topPtr = topPtr ->next;

delete tempPtr;

return item;

}