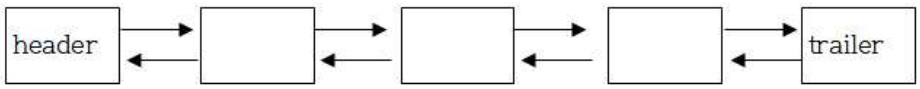
자료구조 과제 1 보고서

201711120 이상목

● Dev-C++ 5.11, TDM-GCC 4.9.2 64-bit Release 환경에서 작성하였다.

문제 1 : 이중연결리스트로 스택 구현

스택을 이중연결리스트 형태로 구현하기 위해서 스택의 한 칸을 노드로 보고 코드를 작성하였다. 문제에서 push, pop 연산의 대상이 문자열 요소임으로, 각 노드가 포함하는 원소의 데이터 타입은 문제 1에서는 string으로 하였다. 구상한 스택의 구조는 다음과 같다.



header는 리스트의 시작(스택의 가장 아래) 노드이고, trailer는 리스트의 마지막(스택의 가장 위) 노드이다. 이중 연결 리스트이므로, 노드 앞뒤로 이동할 수 있도록 prev 포인터와 next 포인터를 사용한다.

본 문제의 구현 방법에 대한 설명은 코드에 대한 직접적인 설명으로 하고자 한다.

```
#include <cstdio>
#include <cstring>
#include <string>
#include <iostream>
#include <sstream>   //
using namespace std;

class Node {
        public:
                string elem;
                Node* prev;
                Node* next;
                friend class Stack;
};

class Stack {
        public:
                Stack();
                ~Stack();
                int size();  //스택의 크기 반환
                bool empty();  //스택이 비었는지 확인
                string& top();   //스택의 최상단요소 반환
                void push(string& e);     //스택의 최상단에 요소 추가
                void pop();               //최상단 요소 제거
                Node* header;   //스택의 가장 아래 노드를 가리킴
                Node* trailer;  //스택의 가장 위 노드를 가리킴
        protected:
                void add(Node *v, string& e);
                void remove(Node *v);
};
```

```
Stack::Stack() {
	header = new Node;
	trailer = new Node;
	header->next = trailer;
	trailer->prev = header;
}  // 스택을 생성하고, head는 다음 노드로 trailer를 가리키고, trailer는 이전 노드로 head를 가리킴

Stack::~Stack(){
	while(!empty())
	{
		pop();
	}
	delete header;
	delete trailer;
}// 스택이 비어 있지 않으면, 원소를 계속 제거. 제거가 끝나면 head와 trailer 삭제

int Stack::size(){
	Node* n = new Node;
	int count;
	n = header->next;
	while(n != trailer)
	{
		n = n->next;
		count++;
	}
	return count;
}// 스택의 처음부터 시작해서 끝노드를 접근할 동안 증가한 수가 노드의 개수

bool Stack::empty() {
	return header->next == trailer;
}// header노드가 다음 노드로 trailer노드를 가리키면, 스택은 비어 있음

string& Stack::top() {
	return trailer->prev->elem;
}//stack의 최 상단 노드의 요소 반환

void Stack::add(Node *v, string& e){
	Node* u = new Node;
	u->elem = e;
	u->next = v;
	u->prev = v->prev;
	v->prev->next = v->prev = u;
}//스택에 어떤 노드를 추가하는 함수, 노드 v앞에 저장한다.

void Stack::remove(Node* v) {
	Node* u = v->prev;
	Node* w = v->next;
	u->next = w;
	w->prev = u;
	delete v;
}//노드 v를 삭제하는 함수
```

```
void Stack::push(string& e) {
	add(trailer,e);
} //stack의 최상단에 노드 추가. add 함수 사용

void Stack::pop(){
	if (header->next == trailer)
	{
	}
	else
		remove(trailer->prev);
}// stack의 최상단 노드 제거. 만약 스택이 비었다면, 아무것도 실행하지 않음

//-------------------LinkedListStack 구현--------------------

string intToString(int n){
		stringstream s;
		s << n;
		return s.str();
}// 정수를 string 타입으로 변환해주는 함수를 구현하였다.

int main(void)
{
	Stack stack;
	string operation;  // 입력받는 명령
	string content;    // 스택의 요소
	string output;     // 출력 결과를 저장할 문자열

	string push = "PUSH" ; string pop = "POP"; string size = "SIZE";
	string empty = "EMPTY" ; string top = "TOP"; string quit = "QUIT";
	cout << "Stack에 할 연산 : PUSH // POP // SIZE // EMPTY // TOP // QUIT" << endl;
	while(1){
		cin >> operation;
		if (operation.compare(push)==0)  // push 연산
		{
			cin >> content;
			stack.push(content);
		}
		else if (operation.compare(pop)==0) // pop 연산
		{
			if (stack.empty()==1)
			{
				output += "ERROR";    //스택이 비어있으면, pop을 실행하지 않고
error 출력
				output += "\n";
			}

			else
				stack.pop();          //스택이 비어있지 않으면 pop실행
		}
		else if (operation.compare(size)==0)  // stack size구하기
```

```
                    {
                              output += intToString(stack.size());  //정수를 string타입으로 형변환하는 함
수. 직접 구현하였다.
                              output += "\n";
                    }
                    else if (operation.compare(empty)==0)  //stack이 비었는지 확인
                    {
                              if(stack.empty()==1){
                                        output += "TRUE";      //스택이 비었으면 출력에 TRUE 추가
                                        output += "\n";
                               }
                              else{

                                        output += "FALSE";      //스택이 비어있지 않으면 출력에 FALSE
추가
                                        output += "\n";
                              }
                    }
                    else if (operation.compare(top)==0)  //stack의 가장 위의 요소 반환
                    {
                              output += stack.top();
                              output += "\n";                    // top(해결),empty(해결),size, pop했을 때
ERROR 출력(해결)
                    }
                    else if (operation.compare(quit)==0)  // 명령어 받기 종료
                    {
                              break;
                    }
                    else
                    {
                              cout << "wrong instruction" << endl;  //등록되지 않는 명령이면 오류 출력
                    }
          }
          cout << endl;
          cout << output << endl;           //명령 실행 결과 출력
}
string intToString(int n){              // 정수 값을 string으로 바꿔주는 함수
                    stringstream s;
                    s << n;
                    return s.str();
}
```

문제 2-1 : 중위표기법을 후위표기법으로 표기

후위표기법으로 표기하는 알고리즘

1) 중위 표기 식을 한 글자씩 읽는다.

2) 읽은 글자가 피연산자(숫자)이면, 결과를 출력할 문자열(후위표기법)에 대입한다.

3-1) 읽은 글자가 왼쪽 괄호이면, 연산자를 저장하는 스택에 넣는다.

3-2) 읽은 글자가 연산순위가 낮은 연산자, + 또는 - 이면, (괄호만 남을 때 까지 연산자 스택에 들어 있던 연산자를 순서대로 후위표기법 문자열에 출력하고, 뽑은 연산자는 삭제한다. 그 후 방금 읽은 연산자를 스택에 넣는다.

3-3) 읽은 글자가 연산순위가 높은 연산자, * 또는 / 이면, 스택의 최상위가 * 또는 / 인 동안 (즉, 연산 순위가 같은 것이 없으면) 연산자 스택에 들어 있던 연산자를 순서대로 후위표기법 문자열에 출력하고, 뽑은 연산자는 삭제한다. 스택의 최상위에 + 나 -가 있으면, *와 /를 연산자 스택의 최상위에 집어넣는다. (연산자의 우선순위 구현)

3-4) 공백을 만나면 건너뛴다.

4) 3-1,3-2,3-3을 반복하며 중위표기식을 모두 읽으면, 스택에는 가장 나중에 연산해야 할 연산자들이 남는다. 이 연산자를 스택의 최상위부터 후위표기식 문자열에 출력한다.

다음은 후위표기법을 구하는 함수를 구현한 코드이다. 본 문제는 스택에 연산자 한 개 씩을 대입해야 하기 때문에 노드 요소의 데이터타입은 문제 2와 다르게 char 타입이다.

```
string GetPostFix(string infixExpression)
{
	string postfixExpression;
	string::iterator i = infixExpression.begin();
	string::iterator start = infixExpression.begin();
	string::iterator end;

	Stack stack;

	for(; i != infixExpression.end(); i++)  //중위연산 읽기
	{
		if(operators.find(*i)==string::npos)  //숫자면 바로 출력
		{
			postfixExpression += *i;
			postfixExpression += " "; //숫자 다음 공백 넣기
			continue;
		}

		switch(*i)			//연산자
		{
			case '(':
				stack.push('(');   //왼쪽괄호는 일단 스택에 넣는다
				break;
			case ')':
				while(stack.top() != '(')    //오른쪽 괄호를 읽으면 왼쪽 괄호가 나올 때 까지 스택에 들어 있던 연산자를 꺼내서 출력
				{
					postfixExpression += stack.top();
```

```
                                        postfixExpression += " ";
                                        stack.pop();  //꺼낸 연산자 제거
                                }
                                stack.pop();  // 만약 스택의 최상위가 오른쪽 괄호면, 출력하지 않고 스택에서 제거
                                break;
                        case '+':
                        case '-':           //연산우선순위가 낮은 연산자
                                while(stack.size() != 0 && stack.top() != '(')  //스택이 비어있지 않고 스택의 최상위 요소가 (가 아닌 동안
                                {
                                        postfixExpression += stack.top();   // 스택에 들어 있던 연산자를  표현식에 출력
                                        postfixExpression += " ";
                                        stack.pop();  //뽑은 연산자는 삭제
                                }
                                stack.push(*i);  //반복문 끝나면 읽은 연산자를 스택에 대입
                                break;
                        case '*':
                        case '/':           //연산우선순위가 높은 연산자
                                while(stack.size()!=0  &&  (stack.top()==  '*'||  stack.top()==  '/'))  //스택이 비어있지 않고 스택 최 상위가 * 또는 /인 동안 (연산순위가 같은 것이 없는 동안)
                                {
                                        postfixExpression += stack.top();    //// 스택에 들어 있던 연산자를  표현식에 출력
                                        postfixExpression += " ";
                                        stack.pop();  //뽑은 연산자는 삭제
                                }
                                stack.push(*i);  //스택의 최상위가 +또는 -인 경우, 그 위에 * 또는 /를 넣음(연산자 우선순위 구현)
                                break;
                        case ' ':  //공백은 건너뜀
                                break;
                        default:   // 오류 처리
                                break;
                }
        }
        for(int i=0; i<stack.size()+1; i++ )
        {
                postfixExpression += stack.top();  // 스택에 남아 있는 연산기호를 넣는 과정
                postfixExpression += " ";
                stack.pop();  //연산기호를 넣고 스택에 들어 있던 연산기호는 제거
        }
        return postfixExpression;
}

int main(void)
{
        int n;
        cin >> n;
        string *infixExpression = new string[n+1];
```

```
	string *postfixExpression = new string[n+1];

	for (int i=0; i<n+1; i++)
	{
		getline(cin, infixExpression[i]);	// 중위연산 n개를 입력받음
	}
	for (int i=1; i<n+1; i++)
	{
		postfixExpression[i] = GetPostFix(infixExpression[i]);	// 중위연산 n개를 후위연산 n
개로 변환
		cout << postfixExpression[i] << endl;	// 후위연산 출력
	}
	return 0;
}
```

문제 2-2 : 후위표기법으로 표현된 식의 계산

후위 표기식을 계산하는 알고리즘

1) 후위 표기식을 왼쪽에서부터 한 글자씩 읽는다.

2) 읽은 글자가 피연산자이면, 피연산자는 스택에 담는다.

3) 읽은 글자가 연산자이면, 스택에서 연산자 2개를 꺼내서(스택에 대해서 2번의 top, pop실행) 연산을 실행하고, 그 연산 결과로 나온 숫자를 다시 스택에 삽입한다.

4) 위 과정을 반복하면 스택에는 최종 연산 결과만 남고, 이를 출력하면 계산 결과를 얻을 수 있다.

다음은 후위 표기법을 계산하는 함수를 구현한 코드이다. 본 연산에서는 숫자를 직접 계산해야 하므로, 문제 1과 달리, 노드 요소의 데이터 타입을 double형으로 지정하였다.

```
double Calculate(const string postfixExpression)
{
	Stack stack;  // 숫자를 저장할 스택
	string token = "후위표기법의 글자 하나를 저장할 변수//일단 임의의 내용으로 선언";

	stringstream temp(postfixExpression);  //temp라는 string에 후위표기법을 넣는다.
	while(!temp.eof())
	{
		temp >> token;		//token에는 후위표기법에서 한 글자씩 읽은 것이 들어간다.
		if (operators.find(token) == string::npos)   // token이 숫자이면
		{
			double num;
			stringstream(token) >> num;	//token을 임시 숫자 변수에 넣는다.
			stack.push(num);		// 임시 숫자 변수의 내용을 stack에 넣는다.
		}
		else				// 읽은 글자가 연산기호이면,
		{
			double op1, op2;
			op2 = stack.top();		// op2 : 숫자stack의 가장 위의 숫자
			stack.pop();
			op1 = stack.top();		// op1 : 숫자stack의 위에서 2번째 숫자. 숫자가 들어있는 스택에서 숫자 2개를 꺼낸다.
			stack.pop();
			switch (operators[operators.find(token)])	// 읽은 연산 기호에 따라서 연산의 종류가 달라진다.
			{
			case '+':
				stack.push(op1 + op2); break;
			case '-':
				stack.push(op1 - op2); break;
			case '*':
				stack.push(op1 * op2); break;
			case '/':
				stack.push(op1 / op2); break;
			}// 연산 결과를 숫자가 들어있는 stack에 넣는다.
		}
	}
```

```
        return stack.top();  // 반복문이 끝나면, 스택에 남아있는 한 개의 요소를 꺼낸다. 이것이 연산의 최종 결과이다.
}


int main()
{
        int n;
    cin >> n;
    string *postfixExpression = new string[n+1];
    int *result = new int[n];
    for (int i=0; i<n+1; i++){
        getline(cin, postfixExpression[i]);          // 공백 포함 입력받기 위해 getline 사용
        }
        for (int i=0; i<n+1; i++){
                result[i] = Calculate(postfixExpression[i]);  // 연산결과를 저장할 배열에 연산 결과를 저장
        }
        for (int i=1; i<n+1; i++){
                cout << result[i] << endl;     //n개의 연산 결과 출력
        }
}
```