자료구조

HOMEWORK 1
STACK

Prof. 조성현
TA. 지서연
201711205. 황정평

< 참고사항 >

■입출력방식: 표준 입출력(Standard I/O)

■코드에 대한 설명을 각주가 달린 코드로 설명합니다. 각주를 꼭 읽어주세요!

■노드 == Node

■스택에 담는다. == 스택에 쌓는다. == 스택에 넣어준다.

■이전의 노드 == 앞 노드 == prev

■후의 노드 == 뒷 노드 == next

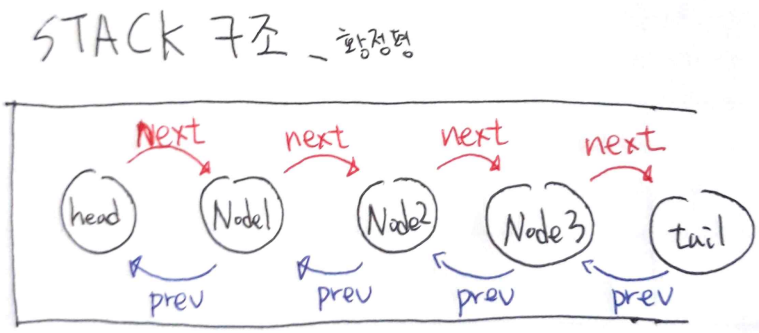
■int main(){ doit(); } 꼴로 작성되어 있기에 입출력 부분은 doit() 함수가 담당한다.

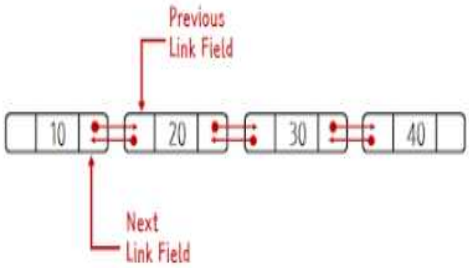
prob1.cpp

알고리즘1. 스택 구조

스택 구조는 간단히 말하면 긴 박스에 공을 넣는 것과 같다. 제일 먼저 들어간 공은 제일 늦게 나올 수 있다. 제일 위에 있는 공만 뺄 수 있으며, 확인할 수 있다.

이러한 스택 구조를 C++에서 DoubleLinkedList로 어떻게 구현할 수 있을까?





■STACK은 생성되며 head Node와 tail Node를 갖는다. 각 Node는 뒷 Node를 가리키는 next 포인터가 있고, 앞 Node를 가리키는 prev 포인터가 있다. 즉, 각 Node들은 연결되어 있다고 말할 수 있다.
**head와 tail Node는 스택 구조를 쉽게 구현하기 위한 헬퍼일 뿐이다. 위 그림을 예로들면, 현재 스택 원소의 개수는 3개이고, 현재 스택의 top은 Node3이며, 현재 스택에 원소를 추가한다면 Node3 뒤에 추가되며, 현재 스택에 원소를 제거한다면 Node3이 제거 될 것이다.

■PUSH를 구현 할 때는, tail 앞에 새로운 Node를 형성하여 넣어준다. 이 때, 원래 tail 앞에 있던 Node의 next는 새로운 Node를 가리키게 하고, tail의 prev도 새로운 Node를 가리키게 한다. 새롭게 생성되는 Node의 prev를 원래 tail 앞에 있던 Node를 가리키게 하고, next는 tail을 가리키게 한다.

■POP을 구현 할 때는, 마지막 노드 앞의 노드의 next가 tail을 가리키게 하고, tail의 prev가 마지막 노드 앞의 노드를 가리키게 한다. 그리고 마지막 노드를 삭제한다. throw catch문을 이용하여, STACK이 비어있다면 ERROR를 출력하도록 한다.

■SIZE를 구현 할 때는, tail부터 출발하여 prev를 이용하여 계속 다음 주소로 이동하며 head가 나오기 전까지 이동한다. 이 때 이동한 횟수를 출력한다.

■EMPTY를 구현 할 때는, tail의 prev가 head라면 true를, head가 아니라면 false를 반환하게 만든다.

■TOP을 구현 할 때는, tail의 prev의 data의 참조 값을 반환하도록 한다. throw catch문을 이용하여, STACK이 비어있다면 ERROR를 출력하도록 한다.

다음 Node클래스가 어떻게 짜여져 있는지 살펴보자.

```
template<class E>
class Node {    // stack 구조 안에 들어가는 클래스
public:
    E data;     // 각 Node는 data 값을 가진다.
    Node* next;     // 다음 Node를 가리키는  포인터
    Node* prev;     // 이전 Node를 가리키는 포인터
    Node() {    // 기본 생성자
        next = NULL;    // 기본 생성자의 next 포인터 값은 NULL
        prev = NULL;    // 기본 생성자의 prev 포인터 값은 NULL
        data = '0';     // 기본 생성자의 data 값은 '0'
    }
    Node(E e, Node* ptr) { // 포인터ptr이 가리키는 Node 뒤에 새로운 Node를 추가
        data = e;   // 역시 data 값을 가지며
        prev = ptr;     // 새로운 Node의 앞 Node의 주소는 ptr 이며
        next = ptr->next;   // 새로운 Node의 뒤 Node의 주소는, 본디 앞주소가 가리키던 다음 주소이다.
        prev->next = this;  // 앞 Node의 next 주소를 새로운 Node의 주소로 변경
        next->prev = this;  // 뒷 Node의 prev 주소를 새로운 Node의 주소로 변경
    }
    void selvDelete() {     // Node 스스로 사라지게 하는 멤버 함수
        prev->next = next;  // 앞 Node의 next 주소를 현재 Node 다음 주소로 변경
        next->prev = prev;  // 뒷 Node의 prev 주소를 현재 Node 이전 주소로 변경
        delete this;    // 스스로를 삭제한다.
    }
};
```

간단히 말하면 Node 클래스는, 앞 노드 주소와 뒷 노드 주소를 가리키는 멤버 변수가 있으며, 자신의 data값인 멤버변수도 있고, 특정 Node 뒤에 Node를 추가하는 멤버 함수와, 양 옆 Node를 이어주고 스스로 소멸하는 멤버 함수도 있다.

스택 클래스 구현을 살펴보자.

```
template<class E>
class Stack {   // 스택 구조 클래스
public:
    Node<E>* head;
    Node<E>* tail;  // 기본적으로 head와 tail 노드를 생성해준다.
    Stack() {   // 기본 생성자
        head = new Node<E>();
        tail = new Node<E>();   // head와 tail 노드는 빈껍데기와 비슷하다. 스택 구조 구현을 편리하게 하기 위함이다.
        head->next = tail;
        tail->prev = head;  // head와 tail을 서로 연결해준다.
    }
    ~Stack() {  // 소멸자
        while (head->next != tail) {    // head 와 tail만 남을 때까지 모든 Node를 삭제한다.
            head->next->selvDelete();
        }
        delete tail;
        delete head;    // 그 후 모든 Node를 삭제한다.
    }
    size_t size() const {   // 스택에 들어있는 Node수를 return 한다.
        size_t count = 0;
        Node<E>* temp = tail;
        while (temp->prev != head) {    // head 에서 출발하여 head와 tail 노드를 제외한 노드의 수를 센다.
            count++;
            temp = temp->prev;
        }
        return count;
    }
```

```
    bool empty() const {    // 스택이 비어 있으면 true, 스택이 비어 있지않으면 false 값을 반환한다.
        if (tail->prev == head) {
            return true;
        }
        else {
            return false;
        }
    }
    const E& top() const throw(StackEmpty) {    // 스택의 젤 마지막 노드 참조 값을 return 한다.
        try {
            if (head->next == tail) {
                throw StackEmpty("ERROR");
            }
            return tail->prev->data;
        }
        catch (StackEmpty& h) {
            cout << h.what() << endl;   // 만약 스택이 비어 있다면 ERROR를 출력한다.
        }
    }
    void push(const E& e) {     // 파라미터 값을 data로 갖는 Node를 제일 맨뒤에 생성한다.
        new Node<E>(e, tail->prev);     // tail 앞 노드의 뒤에 새로운 Node 생성
    }
```

```
    void pop() throw(StackEmpty) {  // 제일 마지막 Node를 제거한다.
        try {
            if (head->next == tail) {
                throw StackEmpty("ERROR");
            }
            tail->prev->selvDelete();   // 스스로 사라지게 한다.
            return;
        }
        catch (StackEmpty& h) {
            cout << h.what() << endl;   // 만약 스택이 비어 있다면 ERROR를 출력한다.
        }

    }
};
```

try, throw, catch 문에 대해 더 설명하면, try에서 if절을 만족하면 throw를 하게 되는데, 그럼 밑에 문들을 다 무시하고 catch 문으로 바로 간다.

```
class StackEmpty : public std::runtime_error {
public:
    explicit StackEmpty(const std::string& msg) : runtime_error(msg) {
        string what = msg;
    }
    explicit StackEmpty(const char* msg) : runtime_error(msg) {

    }
};
```

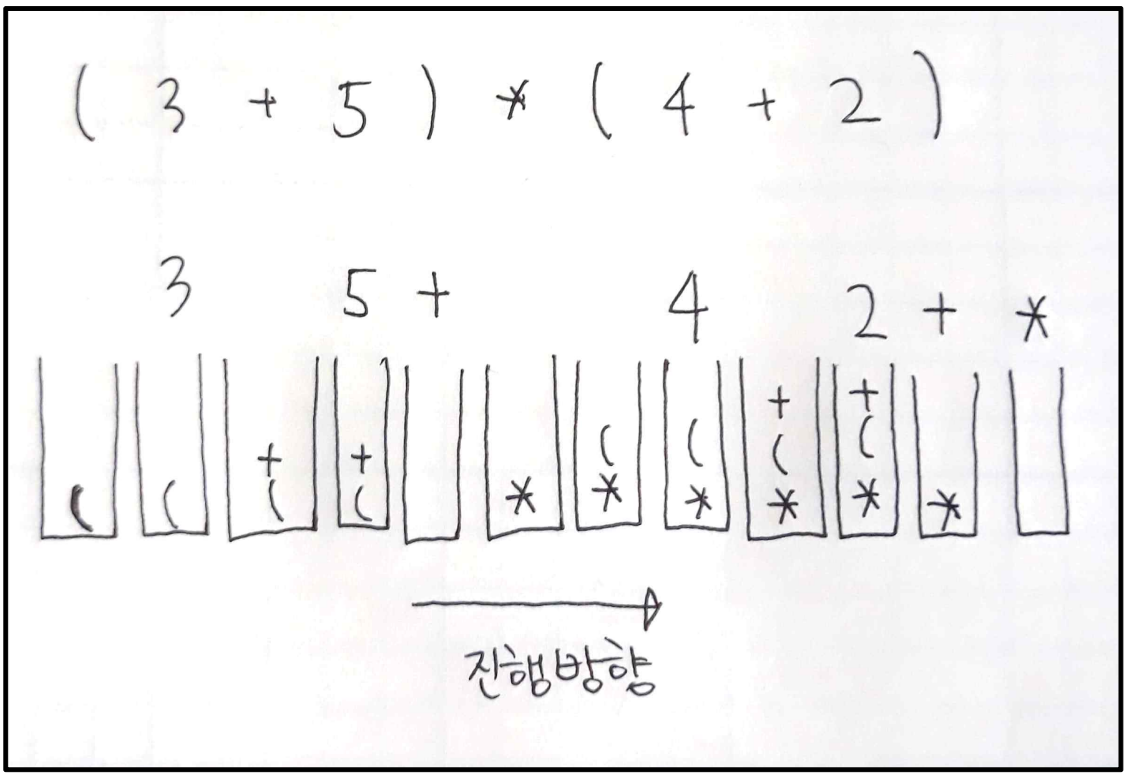
StackEmpty 클래스는 위와 같이 구현되어 있다. catch 문에서 파라미터를 h로 받고 h.what은 그 파라미터를 출력한다.

prob1입출력은 다음과 같이 구현하였다.

```
void doit() {   // 입출력을 받고 그에 맞게 함수들을 실행하는 함수이다.
    Stack<string>* stack = new Stack<string>();     // 스택을 생성해준다.
    string input;
    while (input != "QUIT") {   // 밑에 제어문들은 쉽게 이해할 수 있다.
        getline(cin, input);

        if (input.substr(0, 4) == "PUSH") {
            stack->push(input.substr(5));
        }

        else if (input.substr(0, 3) == "POP") {
            stack->pop();
        }

        else if (input.substr(0, 4) == "SIZE") {
            cout << stack->size() << endl;
        }

        else if (input.substr(0, 5) == "EMPTY") {
            if (stack->empty()) {
                cout << "TRUE" << endl;
            }
            else {
                cout << "FALSE" << endl;
            }
        }

        else if (input.substr(0, 3) == "TOP") {
            if (stack->empty()) {
                cout << "ERROR" << endl;
                continue;
            }
            cout << stack->top() << endl;
        }

        else if (input.substr(0, 4) == "QUIT") {
            break;
        }

        else {
            cout << "INPUT ERROR" << endl;  // 그 외 입력이 들어오면 INPUT ERROR를 출력한다.
        }
    }
}
```

prob2-1.cpp

알고리즘2. 중위표기법 -> 후위표기법



- 중위표기법 수식을 왼쪽에서부터 오른쪽 순으로 스캔한다.
- 숫자가 오면 출력한다.
- 숫자가 아닌 연산자는 stack 에 쌓는다.
- Stack 의 top 에 있는 연산자가 새로 추가되는 연산자의 우선순위보다 높거나 같다면 스택의 top 에 있는 연산자 역시 스택에서 빼서 출력한다. 스택에 더 이상 우선순위가 높거나 같은 연산자가 없을 때까지 이를 반복한다.
- "("이 나온다면 스택에 쌓는다.
- ")"은 스택에서 "("가 나올 때까지 모든 연산자를 스택에서 빼서 출력한다.

Node 클래스와, Stack 클래스는 prob1 과 동일하다.

prob2-1 입출력 부분을 살펴보면 다음과 같다.

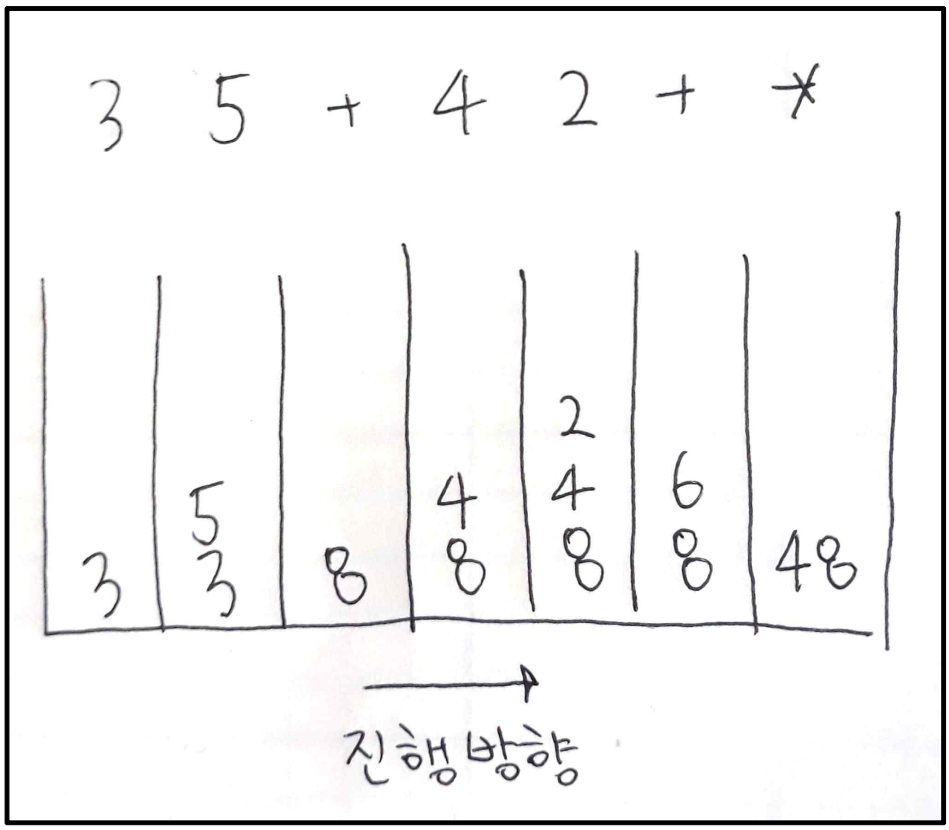
```
void doit() {
    Stack<string>* stack = new Stack<string>();     // stack 생성
    string num_str;
    getline(cin, num_str);  // 우선 getline으로 몇번이나 돌릴건지 입력받는다.
    int num = stoi(num_str);    // int형으로 변환하여 넣어준다.
    string* ans = new string[num];  // 각 줄에 맞는 정답을 넣어주기 위해 배열을 생성한다.
    for (int i = 0; i < num; i++) {
        string input_getline;
        getline(cin, input_getline);    // 우선 getline으로 식을 입력 받는다.
        stringstream input_ss(input_getline);   // 그 다음 stringstream을 이용하여 공백을 기준으로 값을 받는다.
        string input;
        while (input_ss >> input) {     // 공백을 기준으로 값을 input으로 계속 넣어준다.
            /*이제 공백을 기준으로 연산자 또는 숫자를 입력 받는다.
            연산자를 받을 때, 스택이 비어 있다면 그대로 스택에 넣어주고,
            스택이 비어 있지 않다면 스택에, 지금 들어가려는 연산자의 우선순위보다 낮은 연산자만 남을때까지, 연산자를 스택에서 꺼내고 출력한다.
            낮은 연산자가 비로소 남을 때 스택에 넣는다.*/
            /*ans[i] 에 대해 설명하면, 각 줄을 ans[i]에 집어 넣고, 나중에 ans[]를 for문을 돌려 출력한다.*/
            if (input == "+") {
                if (stack->empty()) {
                    stack->push("+");
                }
                else {
                    while (stack->top() == "-" || stack->top() == "*" || stack->top() == "/") {
                        ans[i] += stack->top() + ' ';
                        stack->pop();
                        if (stack->empty()) {
                            break;
                        }
                    }
                    stack->push("+");
                }
            }
```

가운데 생략된 -, *, / 연산자에 대한 if절은 + if절과 비슷하다.

```
            else if (input == "(") {    // 일단 ( 연산자가 들어오면 stack에 쌓는다.
                stack->push("(");
            }
            else if (input == ")") {    // 비로소 ) 연산자가 들어오면 스택에서 원소를 꺼내는데, ( 연산자가 안나올때까지 빼준다.
                if (!stack->empty()) {
                    while (stack->top() != "(") {
                        ans[i] += stack->top() + ' ';
                        stack->pop();
                    }
                    stack->pop(); // 마지막으로 ( 을 빼주는 것
                }
            }
            else {
            ans[i] += input + ' ';  // 연산자가 아니면 숫자이므로 그대로 출력한다. (말이 출력이지 ans[i]에 넣는 것이다.)
            }
        }
        while (!stack->empty()) {   // 스택이 빌때까지 stack에 남아있던 연산자를 빼내 출력한다.
            ans[i] += stack->top() + ' ';
            stack->pop();
        }
    }
    for (int i = 0; i < num; i++) {     // 이제 ans[]에 쌓아왔던 답들을 순서대로 출력한다.
        cout << ans[i] << endl;
    }


}
```

prob2-2.cpp

알고리즘3. 후위표기법->결과값



- 후위표기법 수식을 왼쪽에서부터 오른쪽 순으로 스캔한다.
- 숫자가 오면 스택에 넣는다.
- 숫자가 아닌 연산자는 스택 맨 위 수와 그 아래 수를 연산한다.
- 모든 입력이 끝나면 스택에 남은 원소를 출력한다.
- 이번 과제에서는 비교연산자에 대해 구현되어 있지 않다.

Node 클래스와, Stack 클래스는 prob1 과 동일하다.

prob2-2 입출력 부분을 살펴보면 다음과 같다.

```
void doit() {   // prob2-1에 입력값을 받는 방법에 대해 설명되어 있다.
    Stack<string>* stack = new Stack<string>();
    string num_str;
    getline(cin, num_str);
    int num = stoi(num_str);
    string* ans = new string[num];
    for (int i = 0; i < num; i++) {
        string input_getline;
        getline(cin, input_getline);
        stringstream input_ss(input_getline);
        string input;
        while (input_ss >> input) {
            /*후위표기법->값출력
            이번에는 숫자들을 바로 출력하지 않고 스택에 담는다.
            연산자가 들어오면 바로 스택에 제일 위 두 원소에 대해 연산을 수행한다.
            그 연산한 값은 다시 스택에 담는다.*/
            if (input == "+") {
                string temp = to_string((stoi(stack->tail->prev->prev->data) + stoi(stack->tail->prev->data)));
                stack->pop();
                stack->pop();
                stack->push(temp);
            }
            else if (input == "-") {
                string temp = to_string((stoi(stack->tail->prev->prev->data) - stoi(stack->tail->prev->data)));
                stack->pop();
                stack->pop();
                stack->push(temp);
            }
            else if (input == "*") {
                string temp = to_string((stoi(stack->tail->prev->prev->data) * stoi(stack->tail->prev->data)));
                stack->pop();
                stack->pop();
                stack->push(temp);
            }
            /* 5/2 = 2 가 -5/2 = -3 이 나오도록 해야한다.
            앞 피연산자와 뒷 피연산자의 부호가 동일하다면 / 연산만 수행하였고,
            앞 피연산자와 뒷 피연산자의 부호가 다르다면 / 연산을 한 후, - 1을 해주었다.*/
            else if (input == "/") {
                int first = stoi(stack->tail->prev->prev->data);
                int seconde = stoi(stack->tail->prev->data);
                if (first * seconde >= 0) {
                    string temp = to_string(first / seconde);
                    stack->pop();
                    stack->pop();
                    stack->push(temp);
                }
                else if (first * seconde < 0) {
                    string temp = to_string(first / seconde - 1);
                    stack->pop();
                    stack->pop();
                    stack->push(temp);
                }

            }
            else {  // 연산자가 아닌 숫자는 그대로 스택에 넣어준다.
            stack->push(input);
            }
        }
        ans[i] = stack->top();
    }
    for (int i = 0; i < num; i++) {
        cout << ans[i] << endl;
    }
}
```