**05. 스택(Stack)**

**05-1. 스택의 이해**

: LIFO(Last-In-First-Out)

* 삽입(PUSH), 삭제(POP)

**# 다양한 스택 활용**

* 백 스페이스 키 : 최근에 입력한 글자 삭제
* 최근에 작업한 순으로 취소 : ctrl+z
* 역순 문자열, 진법 변환
* 후위 표기법 : 수식의 괄호 검사, 후위 표기법 변환관 수식 연산
* 시스템 스택(System stack) : 함수의 호출과 복귀 순서를 스택의 LIFO구조를 응용하여 관리

**# 가상 메모리 구조**

* 프로세스 : 운영체제에서 프로세스는 “실행중인 프로그램”
* 프로그램 : 컴퓨터를 실행시키기 위해 차례대로 작성된 “명령어 집합”

텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

**# C++ STL : stack 클래스**

#include <stack>

using namespace std;

stack<DataType> stackName // 빈 스택 생성

void push(const value\_type& val); // 스택에 데이터 항목 추가

void pop(); // 스택에 데이터 항목 삭제

value\_type& top();// 스택의 데이터 항목 반환(top)

bool empty() const; // 스택이 비어 있는지 여부 확인

size\_type size() const; // 스택의 크기 반환

swap(stack1, stack2) // 스택 SWAP

**# 예제 5-1. 스택 활용 – C++STL(stack Class)**

#include <iostream>

#include <stack>

using namespace std;

int main(void) {

stack<int> S;

S.push(10);

S.push(20);

S.push(30);

cout << "stack is empty : " << S.empty() << endl;

cout << "stack size : " << S.size() << endl;

while (!S.empty()) {

cout << "top element : " << S.top() << endl;

S.pop();

}

return 0;

}

**05-2. 스택응용**

**# 다양한 스택활용**

**1. 문자열 역순 출력**

**스크린샷, 텍스트, 라인, 직사각형이(가) 표시된 사진

자동 생성된 설명**

**# 코드**

#include <iostream>

#include <string>

#include <stack>

using namespace std;

int main(void) {

string str;

cout << "문자열 입력: "; getline(cin, str);

stack<char> s;

for (int i = 0; i < str.length(); i++)

s.push(str[i]);

cout << "문자열 역순 출력: ";

while (!s.empty()) {

cout << s.top();

s.pop();

}

cout << endl;

return 0;

}

**2. 십진수를 이진수로 출력**

**텍스트, 도표, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명**

**# 코드**

#include <iostream>

#include <stack>

using namespace std;

int main(void) {

int num;

cout << "십진수 입력: "; cin >> num;

stack<int> s;

while (num) {

s.push(num % 2);

num /= 2;

}

cout << "2진수 변환: ";

while (!s.empty()) {

cout << s.top();

s.pop();

}

cout << endl;

return 0;

}

**3. 수식괄호검사**

**# 코드**

#include <iostream>

#include <string>

#include <stack>

using namespace std;

int testPair(string str);

int main(void) {

string str;

bool state;

cout << "수식 입력: ";

getline(cin, str);

state = testPair(str);

if (state == true)

cout << "수식의 괄호 쌍이 맞습니다." << endl;

else

cout << "수식의 괄호 쌍이 틀립니다." << endl;

return 0;

}

int testPair(string str) {

stack<int> s;

char symbol;

for (int i = 0; i < str.length(); i++) {

//1) 여는 괄호 : 스택에 PUSH

if (str[i] == '(' || str[i] == '{' || str[i] == '[')

s.push(str[i]);

//2) 닫는 괄호 : 스택에서 POP하여 여는 괄호와 짝을 비교

else if (str[i] == ')' || str[i] == '}' || str[i] == ']') {

if (s.empty())

return false;

symbol = s.top();

if (str[i] == ')' && symbol != '('

|| str[i] == '}' && symbol != '{'

|| str[i] == ']' && symbol != ']')

return false;

s.pop();

}

}

int state;

if (s.empty()) state = true;

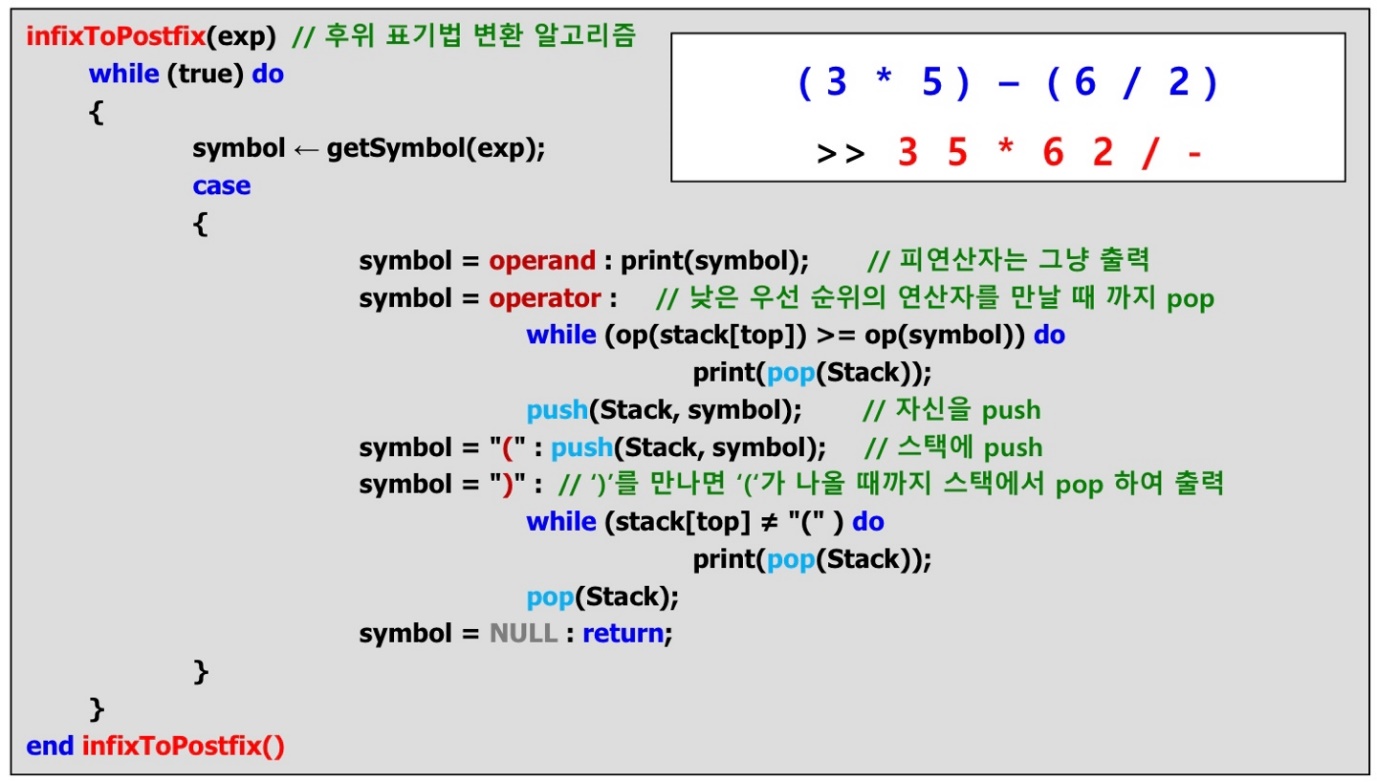
else state = false;

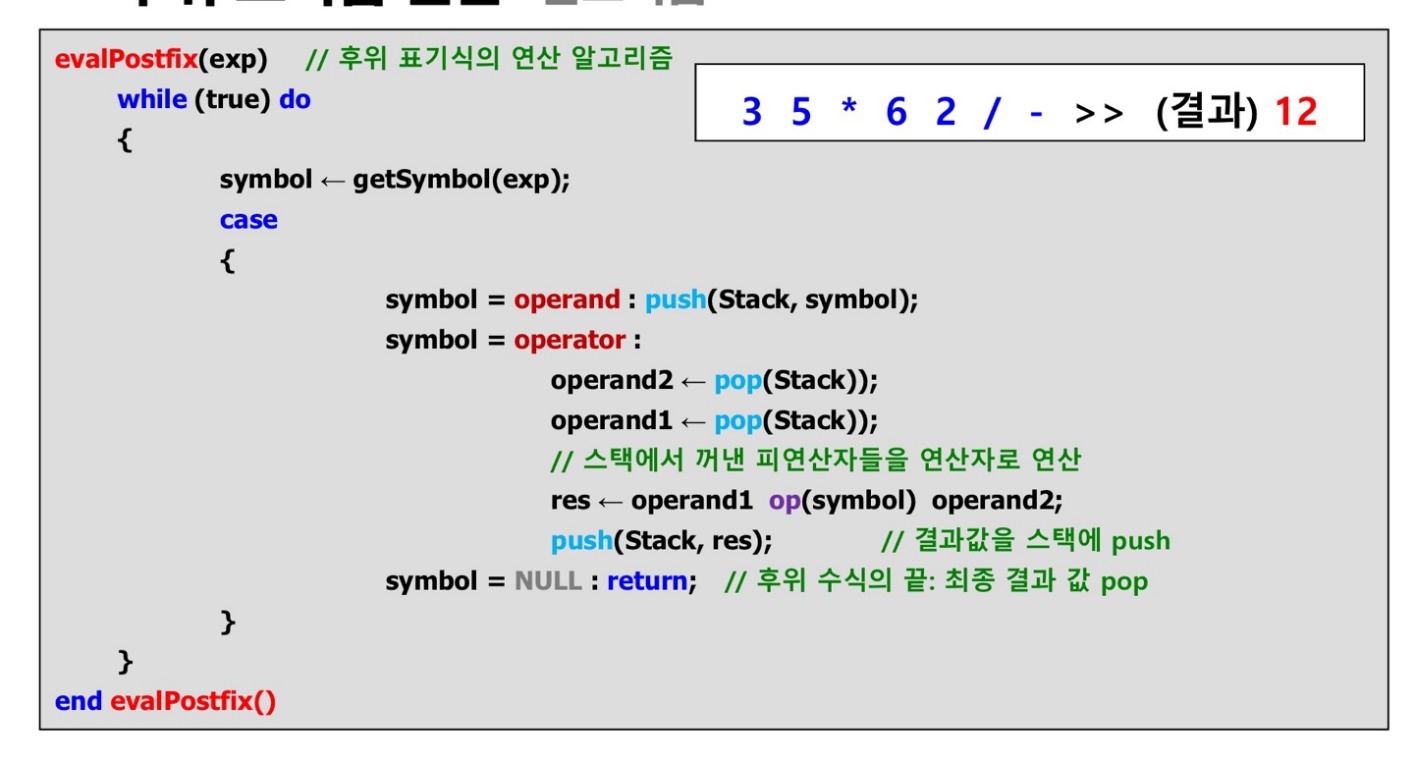
//s.~LinkedStack();

return state;

}

**4. 후위 표기법**

****

****

**# 코드(CPP ver.)**

/\*

스택 활용: 후위 표기법 수식 계산

- evalPostfix : 후위 표기법으로 수식 계산

- InfixToPostfix : 중위 표기법을 후위 표기법으로 변환

중위 표기: (A + B) - C

후위 표기: A B + C -

전위 표기: - + A B C

\*/

#include <iostream>

#include <string>

#include <stack>

using namespace std;

int evalPostfix(string& str);

string InfixToPostfix(string& infix);

int isOperator(int op);

int precedence(int op);

int main(void)

{

int res;

string infixStr, postfixStr;

cout << "수식 입력: ";

getline(cin, infixStr); // cin.getline(infixStr, bufferMAXSIZE);

postfixStr = InfixToPostfix(infixStr);

cout << "\n후위표기법 변환: " << postfixStr << endl;

res = evalPostfix(postfixStr);

cout << "연산 결과: " << res << endl;

return 0;

}

int evalPostfix(string& str) {

stack<char> s;

string temp;

int op1, op2, res = 0;

for (int i = 0; i < str.length(); i++) {

// 피연산자 일 경우...

if (str[i] >= '0' && str[i] <= '9') {

// 한 자리 이상을 처리 하기 위해...

temp.clear();

for (; str[i] >= '0' && str[i] <= '9'; i++)

temp += str[i];

s.push(stoi(temp));

i--;

}

else if (isOperator(str[i])) { // 연산자 일 경우...

op2 = s.top(); s.pop();

op1 = s.top(); s.pop();

switch (str[i]) {

case '+': s.push(op1 + op2); break;

case '-': s.push(op1 - op2); break;

case '\*': s.push(op1 \* op2); break;

case '/': s.push(op1 / op2); break;

}

}

else if (str[i] == ' ') continue;

else {

cout << "잘못된 수식!!!" << endl;

exit(100);

}

}

// 스택에 남은 최종 결과 값

if (!s.empty())

res = s.top();

return res;

}

// 중위 표기법을 후위표기법으로 변환

string InfixToPostfix(string& infix) {

stack<char> s;

string postfix;

for (int i = 0; i < infix.length(); i++) {

// 1) '(' 는 스택에 push

if (infix[i] == '(')

s.push(infix[i]);

// 2) ')'를 만나면 '('가 나올 때까지 pop 한 후에 '('는 버린다.

else if (infix[i] == ')') {

while (s.top() != '(') {

postfix += s.top(); s.pop();

postfix += ' ';

}

s.pop(); // '(' 를 버린다.

}

// 3) 연산자인 경우...

else if (isOperator(infix[i])) {

while (!s.empty() &&

precedence(s.top()) >= precedence(infix[i]))

{ // 자신보다 높은 우선순위의 연산자는 스택에서 pop

postfix += s.top(); s.pop();

postfix += ' ';

}

s.push(infix[i]); // 자신을 push

}

// 4) 피연산자인 경우...

else if (infix[i] >= '0' && infix[i] <= '9') {

for (; infix[i] >= '0' && infix[i] <= '9'; i++)

postfix += infix[i];

postfix += ' ';

i--;

}

else if (infix[i] == ' ') continue;

else {

cout << "잘못된 수식!!!" << endl;

exit(100);

}

}

// 스택에 남은 연산자를 모두 pop 한다.

while (!s.empty()) {

postfix += s.top(); s.pop();

postfix += ' ';

}

return postfix;

}

// 연산자 여부를 판별한다.

int isOperator(int op) {

return op == '+' || op == '-' || op == '\*' || op == '/';

}

// 연산자 우선순위를 수치로 변환

int precedence(int op) {

if (op == '(') return 0;

else if (op == '+' || op == '-') return 1;

else if (op == '\*' || op == '/') return 2;

else return 3;

}

**# 코드(C ver.)**

/\*

스택 활용: 후위 표기법을 이용한 수식 계산

- evalPostfix : 후위 표기법으로 수식 계산

- InfixToPostfix : 중위 표기법을 후위 표기법으로 변환

중위 표기: (A + B) - C

후위 표기: A B + C -

전위 표기: - + A B C

\*/

#include <stdio.h>

#include <stdlib.h> // atoi

#include "LinkedStack.h" // LinkedStack, SNode

// #include "LinkedNode.h" // SNode

#include "Operators.h" // isOperator,precedence

#define bufferMAXSIZE 1024

// 후위 표기법: 수식 계산과 변환(중위표기-> 후위 표기)

element evalPostfix(char\* exp);

void InfixToPostfix(char\* postfix, char\* infix);

int main(void)

{

int res;

char infixStr[bufferMAXSIZE], postfixStr[bufferMAXSIZE];

printf("수식 입력: ");

gets\_s(infixStr, sizeof(infixStr));

InfixToPostfix(postfixStr, infixStr);

printf("\n후위표기법 변환: %s \n", postfixStr);

res = evalPostfix(postfixStr);

printf("연산결과: %d \n", res);

return 0;

}

// 후위 표기법: 수식 계산

element evalPostfix(char\* exp) {

int op1, op2, res;

char temp[bufferMAXSIZE], \*p;

LinkedStack\* S = stackCreate(); // 빈 스택 생성

while (\*exp) {

// 1) 피연산자 일 경우...

if (\*exp >= '0' && \*exp <= '9') {

p = temp; // 한 자리 이상을 처리 하기 위해...

while (\*exp >= '0' && \*exp <= '9')

\*p++ = \*exp++;

\*p = '\0';

push(S, atoi(temp));

}

// 2) 연산자일 경우...

else if (isOperator(\*exp)) {

op2 = top(S); pop(S);

op1 = top(S); pop(S);

switch (\*exp) {

case '+': push(S, op1 + op2); break;

case '-': push(S, op1 - op2); break;

case '\*': push(S, op1 \* op2); break;

case '/': push(S, op1 / op2); break;

}

exp++;

}

else if (\*exp == ' ') exp++;

else {

printf("잘못된 수식!!! \n");

stackDestroy(S);

return 0;

}

}

// 스택에 남은 최종 결과 값

if (!stackEempty(S))

res = top(S); pop(S);

stackDestroy(S);

return res;

}

// 후위표기법 변환(중위 표기 -> 후위 표기)

void InfixToPostfix(char\* postfix, char\* infix) {

LinkedStack\* S = stackCreate();

while (\*infix) {

// 1) '(' 는 스택에 push

if (\*infix == '(')

push(S, \*infix++);

// 2) ')'를 만나면 '('가 나올 때까지 pop 한후에 '('는 버린다.

else if (\*infix == ')') {

while (top(S) != '(') {

\*postfix++ = top(S); pop(S);

\*postfix++ = ' ';

}

pop(S); // '(' 를 버린다.

infix++;

}

// 3) 연산자이면...

else if (isOperator(\*infix)) {

while (!stackEempty(S) &&

precedence(top(S)) >= precedence(\*infix))

{ // 자신보다 높은 우선순위의 연산자는 스택에서 pop

\*postfix++ = top(S); pop(S);

\*postfix++ = ' ';

}

push(S, \*infix++); // 자신을 push

}

// 4) 피연산자인 경우...

else if (\*infix >= '0' && \*infix <= '9') {

while (\*infix >= '0' && \*infix <= '9')

\*postfix++ = \*infix++;

\*postfix++ = ' ';

}

else if (\*infix == ' ') infix++;

else {

printf("잘못된 수식!!! \n");

stackDestroy(S);

return;

}

}

// 스택에 남은 연산자를 모두 pop 한다.

while (!stackEempty(S)) {

\*postfix++ = top(S); pop(S);

\*postfix++ = ' ';

}

postfix--;

\*postfix = '\0';

stackDestroy(S);

return;

}

/\*

#include "Operators.h"

// 연산자 여부를 판단

int isOperator(int op) {

return op == '+' || op == '-' || op == '\*' || op == '/';

}

// 연산자 우선순위를 수치로 변환

int precedence(int op) {

if (op == '(') return 0;

else if (op == '+' || op == '-') return 1;

else if (op == '\*' || op == '/') return 2;

else return 3;

}

\*/

**05-3. 스택 구현**

**0. 스택 구현 : 알고리즘**

- **초기의 빈 스택 생성**

stackCreate()

stack[n]; //크키가 n인 1차원 배열 생성

top 🡨 -1;

end stackCreate()

- **최대로 저장할 수 있는 원소 개수를 크기로 하여 1차원 배열을 선언**

🡪 저장된 원소가 없는 빈 스택이므로 top을 -1로 초기화

- **스택에 데이터 삽입**

push(S, data)

if (top+1=n) then stackFull;

else S(++top) 🡨 data;

end push()

- **스택에서 데이터 삭제**

pop(S)

if(top=-1) then stackEmpty;

else return S(top--);

end pop()

**- 스택에서 데이터 확인 : 맨 마지막에 삽입된 데이터**

stackPeek(S)

if(top=-1) then stackEmpty;

else return S(top);

end stackPeek()

**- 스택의 공백 상태 판단**

stackEmpty(S)

if(top=-1) then return true;

else return false;

end stackEmpty;

**- 스택의 포화 상태 판단**

stackFull(S)

if(top+1=n) then return true;

else return false;

end stackFull()

**1. 순차 자료구조**

* 순차 자료구조인 1차원 배열을 이용하여 구현
  + 스택 크기 : 배열의 크기
  + 스택에 저장된 원소의 순서 : 배열 원소의 첨자
* **변수 top : 스택에 저장된 마지막 원소에 대한 첨자 저장**
  + 공백 상태 : top=-1(초기값)
  + 포화 상태 : top=n-1

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

/\*

스택: 알고리즘 구현 -- 순차자료구조(배열)

- arrayStack : stack, top

\*/

#include <iostream>

#define stackMAXSIZE 100

using namespace std;

inline void error(const char\* message);

// 배열 스택: arrayStack -- 클래스 설계

template <typename E>

class arrayStack {

private:

E stack[stackMAXSIZE];

int top;

public:

arrayStack();

~arrayStack();

void push(const E& e);

E pop(void);

E peek(void) const;

bool isEmpty(void) const;

bool isFull(void) const;

void printStack(void) const;

};

/\*

배열 스택: arrayStack -- 메소드 정의

- 스택 성생 및 삭제: arrayStack, ~arrayStack

- 원소 삽입과 삭제 : push, pop

- 데이터 확인 : peek

- 스택 상태 판단 : isEmpty, isFull

- 전체 원소 출력 : printStack

\*/

// 생성자: 빈 스택 생성

template <typename E>

arrayStack<E>::arrayStack() : top(-1) {}

// 소멸자

template <typename E>

arrayStack<E>::~arrayStack() {}

// PUSH : 스택에 데이터 삽입

template <typename E>

void arrayStack<E>::push(const E& e) {

if (isFull()) error("스택 포화 에러"); // throw "ERROR::STACK IS FULL";

stack[++top] = e;

}

// POP : 스택에서 데이터 삭제

template <typename E>

E arrayStack<E>::pop(void) {

if (isEmpty()) error("스택 공백 에러"); // throw "ERROR::STACK IS EMPTY";

return stack[top--];

}

// PEEK : 스택 맨 위의 원소 확인

template <typename E>

E arrayStack<E>::peek(void) const {

if (isEmpty()) error("스택 공백 에러"); // throw "ERROR::STACK IS EMPTY";

return stack[top];

}

// 스택의 공백 상태 여부 판단

template <typename E>

bool arrayStack<E>::isEmpty(void) const {

return top == -1;

}

// 스택의 포화 상태 여부 판단

template <typename E>

bool arrayStack<E>::isFull(void) const {

return top == stackMAXSIZE - 1;

}

// 스택의 전체 원소 출력

template <typename E>

void arrayStack<E>::printStack(void) const {

cout << "\n STACK [";

for (int i = 0; i <= top; i++) {

cout.width(3);

cout << stack[i];

}

cout << " ]" << endl;

}

inline void error(const char\* message) {

cout << message;

exit(100);

}

int main(void)

{

int num, choice;

arrayStack<int> s = arrayStack<int>(); // 빈 스택 생성

while (true) {

system("cls");

cout << "\n ### 스택 구현: 1차원 배열 ### \n" << endl;

cout << "1) 데이터 삽입: PUSH" << endl;

cout << "2) 데이터 삭제: POP" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

s.push(num);

break;

case 2: cout << "삭제 된 데이터: " << s.pop() << endl;

break;

case 3: s.printStack();

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

}

// s.~ArrayStack();

return 0;

}

**2. 연결 자료구조**

* 단순 연결 리스트를 이용하여 구현
  + 스택의 원소 : 단순 연결 리스트의 노드
    - 스택 원소의 순서 : 노드의 링크 필드로 연결
    - push : 항상 리스트의 첫번째 노드로 삽입
    - pop : 항상 리스트의 마지막 노드를 삭제
* **변수 top : 단순 연결 리스트의 마지막 노드를 가리키는 포인터 변수**
  + 초기 상태 : top=NULL

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

/\*

스택: 알고리즘 구현 -- 단순연결리스트

- main : 데이터 삽입(push).삭제(pop), 전체 원소 출력

\*/

#include <iostream>

#include "LinkedStack.h"

using namespace std;

int main(void)

{

int num, choice;

LinkedStack<int> S; // 빈 스택 생성

while (true) {

system("cls");

cout << "### 스택 구현: 단순 연결 리스트 ###\n" << endl;

cout << "1) 데이터 삽입(push)" << endl;

cout << "2) 데이터 삭제(pop)" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

S.push(num);

break;

case 2: cout << "삭제 된 데이터: " << S.top() << endl;

S.pop();

break;

case 3: S.printStack();

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

}

return 0; // S.~ListStack();

}

/\*

스택: 알고리즘 구현(C++) --단순연결리스트

파일명: LinkedStack.h

클래스: SNode

클래스: LinkedStack

- 생성자.소멸자: LinkedStack, ~LinkedStack

- 데이터 항목 삽입.삭제: push, pop

- 데이터 항목 확인(peek): top

- 빈 스택 여부 판단: empty

- 스택 크기(데이터 항목 개수): size

- 스택의 전체 데이터 항목 출력: printStack

\*/

#include <iostream>

using namespace std;

inline void error(const char\* message) {

cout << message;

exit(100);

}

template <typename E>

class LinkedStack;

// SNode class

template <typename E>

class SNode {

public:

SNode(const E& data);

E getData(void) const;

private:

E \_\_data;

SNode<E>\* \_\_link;

template <typename E> friend class LinkedStack;

};

// SNode class: 멤버 함수 외부 정의

template <typename E>

SNode<E>::SNode(const E& data) :

\_\_data(data), \_\_link(nullptr) {}

template <typename E>

E SNode<E>::getData(void) const { return \_\_data; }

// LinkedStack class

template <typename E>

class LinkedStack {

public:

LinkedStack();

~LinkedStack();

void push(const E& e);

void pop(void);

E& top(void) const;

bool empty(void) const;

int size(void) const;

void printStack(void) const;

private:

SNode<E>\* \_\_top;

int \_\_count;

};

// LinkedStack class: 생성자(소멸자)와 소멸자

template <typename E>

LinkedStack<E>::LinkedStack()

: \_\_top(nullptr), \_\_count(0) {}

template <typename E>

LinkedStack<E>::~LinkedStack() {

SNode<E>\* temp = \_\_top;

while (temp) {

\_\_top = temp->\_\_link;

delete temp;

temp = \_\_top;

}

}

// push: 스택의 데이터 항목 삽입(맨 마지막 원소)

template <typename E>

void LinkedStack<E>::push(const E& e) {

SNode<E>\* newSNode = new SNode<int>(e);

// newSNode->\_\_data = e;

newSNode->\_\_link = \_\_top;

\_\_top = newSNode;

\_\_count++;

}

// pop: 스택에서 맨 마지막에 삽입된 데이터 항목 삭제

template <typename E>

void LinkedStack<E>::pop(void) {

if (empty()) error("ERROR::STACK IS EMPTY"); // throw "ERROR::STACK IS EMPTY";

SNode<int>\* temp = \_\_top;

if (temp) {

\_\_top = temp->\_\_link;

delete temp;

\_\_count--;

}

}

// top(peek): 스택에서 맨 마지막에 삽입된 데이터 항목 확인

template <typename E>

E& LinkedStack<E>::top(void) const {

if (empty()) error("STACK IS EMPTY");// throw "ERROR::STACK IS EMPTY";

return \_\_top->\_\_data;

}

// empty: 스택의 공백 여부 확인

template <typename E>

bool LinkedStack<E>::empty(void) const { return \_\_top == nullptr; }

// size: 스택의 크기(항목의 개수)

template <typename E>

int LinkedStack<E>::size(void) const { return \_\_count; }

// printStack: 스택의 전체 데이터 항목 출력

template <typename E>

void LinkedStack<E>::printStack(void) const {

SNode<E>\* temp = \_\_top;

cout << "\n STACK [";

while (temp) {

cout.width(3);

cout << temp->\_\_data;

temp = temp->\_\_link;

}

cout << " ]" << endl;

}

**05-4. 후위표기법을 이용한 수식 계산\_송**

**# 코드**

#include <iostream>

using namespace std;

#define bufferMAXSIZE 1024

int evalPostfix(char\* str);

void InfixToPostfix(char\* postfix, char\* infixStr);

int isOperator(int op);

int precedence(int op);

inline void error(const char\* message);

template<typename E>

class stackNode {

E data;

stackNode<E>\* link;

template<typename V> friend class linkedStack;

};

template<typename E>

class linkedStack {

stackNode<E>\* top;

public:

//생성자 & 소멸자

linkedStack();

~linkedStack();

//생성

stackNode<E>\* makeStackNode(const E& e);

//조사

bool isEmpty();

//출력

void printStack();

//push,pop,peek

void push(const E& e);

E pop();

E peek();

};

template<typename E>

linkedStack<E>::linkedStack() : top(NULL) {}

template<typename E>

linkedStack<E>::~linkedStack() {

while (!isEmpty()) pop();

}

//생성

template<typename E>

stackNode<E>\* linkedStack<E>::makeStackNode(const E& e) {

stackNode<E>\* nNode = new stackNode<E>;

nNode->data = e;

nNode->link = NULL;

return nNode;

}

//조사

template<typename E>

bool linkedStack<E>::isEmpty() {

return top == NULL;

}

//출력

template<typename E>

void linkedStack<E>::printStack() {

stackNode<E>\* temp = top;

cout << "\n STACK [";

while (temp) {

cout.width(3);

cout << temp->data;

temp = temp->link;

}

cout << " ]" << endl;

}

//push,pop,peek

template<typename E>

void linkedStack<E>::push(const E& e) {

stackNode<E>\* SNode = makeStackNode(e);

SNode->link = top;

top = SNode;

}

template<typename E>

E linkedStack<E>::pop() {

if (isEmpty()) error("스택 공백 에러");

stackNode<E>\* old = top;

E data = old->data; //data 백업

top = old->link;

delete old;

return data;

}

template<typename E>

E linkedStack<E>::peek() {

if (isEmpty()) error("스택 공백 에러");

return top->data;

}

inline void error(const char\* message) {

cout << message << endl;

exit(100);

}

//후위 표기법 함수

int evalPostfix(char\* str) {

int op1, op2, res;

char temp[bufferMAXSIZE], \* p; //임시배열 : 두자리 수 이상도 처리하기 위해

linkedStack<int> s;

while (\*str) {

//case 1 : 피연산자

if (\*str >= '0' && \*str <= '9') {

p = temp;

while (\*str >= '0' && \*str <= '9')

\*p++ = \*str++;

\*p++ = '\0';

s.push(atoi(temp));

}

//case 2: 연산자

else if (isOperator(\*str)) {

op2 = s.pop();

op1 = s.pop();

switch (\*str) {

case '+': s.push(op1 + op2); break;

case '-': s.push(op1 - op2); break;

case '\*': s.push(op1 \* op2); break;

case '/': s.push(op1 / op2); break;

}

str++;

}

else if (\*str == ' ') str++;

else {

cout << "잘못된 수식입니다. " << endl;

return 0;

}

}

if (!s.isEmpty()) res = s.pop();

return res;

}

void InfixToPostfix(char\* postfix, char\* infix) {

linkedStack<int> s;

while (\*infix) {

// case 1 : 괄호 '(' : 스택에 push

if (\*infix == '(') s.push(\*infix++);

// case 2 : 괄호 ')' : '('가 나올 때까지 pop 한 후에 ')'는 버린다.

else if (\*infix == ')') {

while (s.peek() != '(') {

\*postfix++ = s.pop();

\*postfix++ = ' ';

}

s.pop(); // '(' 를 버린다.

infix++;

}

// case 3 : 연산자인 경우, 자신보다 우선순위가 높은 연산자는 스택에서 pop한 후, 자신을 push한다.

else if (isOperator(\*infix)) {

while (!s.isEmpty() &&

precedence(s.peek()) >= precedence(\*infix))

{

\*postfix++ = s.pop();

\*postfix = ' ';

}

s.push(\*infix++);

}

// case 4 : 피연산자

else if (\*infix >= '0' && \*infix <= '9') {

while (\*infix >= '0' && \*infix <= '9')

\*postfix++ = \*infix++;

\*postfix++ = ' ';

}

else if (\*infix == ' ') infix++;

else {

cout << "잘못된 수식입니다. " << endl;

return;

}

}

while (!s.isEmpty()) {

\*postfix++ = s.pop();

\*postfix++ = ' ';

}

postfix--;

\*postfix = '\0';

return;

}

int isOperator(int op) {

return op == '+' || op == '-' || op == '\*' || op == '/';

}

int precedence(int op) {

if (op == '(') return 0;

else if (op == '+' || op == '-') return 1;

else if (op == '\*' || op == '/') return 2;

else return 3;

}

int main() {

int res;

char infixStr[bufferMAXSIZE], postfixStr[bufferMAXSIZE];

cout << "수식 입력: ";

cin.getline(infixStr, bufferMAXSIZE); // getline(cin, infixStr);

cout << "입력된 수식 : " << infixStr << endl;

InfixToPostfix(postfixStr, infixStr); //infixstr의 수식을 postfixstr로 변환

cout << "\n후위표기법 변환: " << postfixStr << endl;

res = evalPostfix(postfixStr);

cout << "연산결과: " << res << endl;

return 0;

}

**# 연구조사**

push

새 노드를 추가하는 함수

1. 새 노드 생성

2. top이랑 연결

pop

노드를 삭제하는 함수

1. old에 삭제할 노드(n)를 백업

2. 노드를 삭제한 후에도, data를 return 할 수 있도록 백업

3. top을 n-1 노드로

4. 노드 삭제

5. 삭제한 노드의 data 리턴

peek

top에 있는 데이터 보여주는 함수

//후위연산자

먼저 중위표기법을 후위표기법으로 변환한 후, 이를 계산한다

evalPostfix

후위표기법 계산하는 함수

세가지 case

case 1: 피연산자

(1) 숫자가 아닌 것을 만날 때까지 임시배열 temp에 숫자를 저장함

(2) (1)이 끝나면 맨 뒤에 NULL문자 추가해주고, atoi를 통해 정수화시켜 push한다.

case 2 : 연산자

(1) 스택에서 두 개의 정수를 pop

(2) switch문 이용, 연산자에 따른 계산을 수행한 후, 스택에 push해둔다.

cass 3: 공백

다음으로 바로 넘어간다

str++에 의해 문자를 하나씩 탐색, 마자막 문자인 NULL문자 만나면 반복문 종료 후 함수 종료

InfixToPOstfix

중위표기법을 후위표기법으로 변환해주는 함수

postfix : 후위표기법 배열

infix : 중위표기법 배열

네 가지의 case를 가진다

case 1: 괄호 '(' : 스택에 바로 push한다.

case 2: 괄호 ')' : '('가 나올 때까지 pop 한 후, ')'는 버린다.

case 3: 연산자 : 자신보다 우선순위 빠른 연산자는 pop 후 자기자신 push

case 4: 피연산자 : 바로 postfix 배열로 넣는다.

스택에는 괄호와 연산자만 들어가며, 피연산자는 바로 후위표기법 배열로 들어간다.

while(\*infix) 종료 후엔, 스택에 남은 연산자나 괄호를 pop하여 postfix에 넣어준다.

필요한 이유 : ex) 3\*(4+5) 의 경우, (4+5)는 처리되나, 스택에 \*가 남는다.

isOperator

어떤 연산자인지 판별한다.

precedence

연산자 우선순위를 숫자로 표현한다.

'('의 우선순위는 반드시 필요하며, 최하위 순위여야 한다.

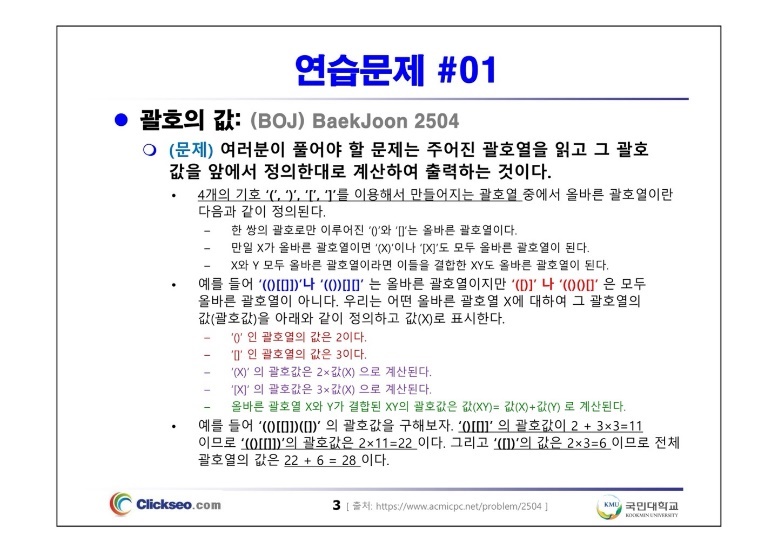
이유 :

'('의 우선순위를 정해두지 않으면, 스택 내에서 '('를 만났을 때, 이를 넘어갈 수 없다.

최하위 순위여야 하는 이유는, '('의 순위가 다른 연산자보다 순위가 높을 경우, pop되어버리기 때문이다.

**05-5. 3차 실습보고서\_연습문제#01\_괄호의 값**

**# 문제**

**텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 친필, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

**# 코드**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

//( : -1, ) : -2, [ : -3, ]: -4

int bracket(string str) {

stack <int> s;

int len = str.length();

int temp = 0;

int result = 0;

for (int i = 0; i < len; i++) {

//열린 괄호인 경우

if (str[i] == '(') s.push(-1);

//닫힌 대괄호인 경우

else if (str[i] == '[') s.push(-3);

//예외1 : 닫힌 괄호가 더 많은 경우

else if (s.empty())

return 0;

else {

//스택의 맨위가 열린 괄호이고, 현재 괄호가 닫힌 괄호인 경우

if (s.top() == -1 && str[i] == ')') {

s.pop();

s.push(2);

}

//스택의 맨위가 열린 대괄호이고, 현재 괄호가 닫힌 대괄호인 경우

else if (s.top() == -3 && str[i] == ']') {

s.pop();

s.push(3);

}

//s.top()이 숫자인 경우, 예외3 : 괄호의 짝이 맞지 않는 경우

else {

temp = 0;

//스택이 비거나, 열린 괄호를 만날 때까지 값을 더함

while (s.top() != -1 && s.top() != -3) {

temp += s.top();

s.pop();

// 예외1 : 닫힌 괄호가 더 많은 경우

if (s.empty())

return 0;

}

//스택의 맨 위가 열린 괄호이고, 현재 괄호가 닫힌 괄호인 경우

if (s.top() == -1 && str[i] == ')') {

temp \*= 2;

s.pop();

}

//스택의 맨 위가 열린 대괄호이고, 현재 괄호가 닫힌 대괄호인 경우

else if (s.top() == -3 && str[i] == ']') {

temp \*= 3;

s.pop();

}

// 예외3 : 괄호의 짝이 맞지 않는 경우

else

return 0;

s.push(temp);

}

}

}

//스택에 남아 있는 값들을 더함

while (!s.empty()) {

// 예외 2 : 열린괄호가 더 많은 경우

if (s.top() == -1 || s.top() == -3)

return 0;

result += s.top();

s.pop();

}

return result;

}

int main() {

string st;

cout << " 20215229 이채훈 " << endl;

cout << "괄호열 :"; cin >> st;

cout << "괄호열의 값(올바르지 않은 괄호열: 0): "<<bracket(st);

}

**# 연구조사**

**\* 문제해법**

EX5 : (()[[]])([])

(

( (

( ( ) -> ( 2

( 2 [

( 2 [ [

( 2 [ [ ] -> ( 2 [ 3

( 2 [ 3 ] -> ( 2 9

( 2 9 ) -> (2 + 9) \* 2 = 22

22 (

22 ( [

22 ( [ ] -> 22 ( 3

22 ( 3 ) -> 22 6

ans= 22 + 6 = 28

0. temp : 괄호열의 부분값을 임시로 저장하는 변수, result : 최종 괄호열의 값

1. 주어진 문자열을 for문으로 순회하면서 열린 괄호일 경우 스택에 해당 괄호의 값을 넣음 (-1 또는 -3).

2. 닫힌 괄호일 경우 스택의 맨 위에 있는 값과 짝이 맞는지 확인하고, 값을 스택에 넣음 (2 또는 3).

3. **스택이 비어있는데 닫힌 괄호가 나오는 경우 예외 처리** (닫힌 괄호가 더 많은 경우).

4. **스택의 맨 위에 있는 값이 숫자인 경우, 괄호의 짝이 맞지 않는 경우 예외 처리.**

5. 최종적으로 스택에 남아 있는 값들을 더하여 괄호열의 전체 값 계산.

**<예외 : 괄호의 짝이 맞지 않는 경우>**

: 예외 처리가 올바르게 되지 않을 경우, 코드가 이상하게 변하여서, 여러가지 테스트를 통해서 예외처리에 대한 경우의 수와 예외처리가 필요해서 예외처리에 집중하였다.

1. 닫힌 괄호왔는데 스택이 비어있는 경우(닫힌 괄호가 더 많은 경우)

2. 반복문이 끝났는데 스택이 비어있지 않는 경우(열린 괄호가 많다)

3. 닫힌괄호와 열린괄호의 짝이 맞지 않는 경우

**05-6. 3차 실습보고서\_연습문제#01\_괄호의 값ver.Ong**

**# 코드**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

int main()

{

stack<char> S;// 문자를 저장하는 스택 선언

string str;

int ans = 0, temp = 1;// 결과값 및 임시 값 초기화

cin >> str;

for (int i = 0; i < str.length(); i++)

{

if (str[i] == '(')

{

temp \*= 2;// 여는 소괄호일 경우, temp 값을 2배로 증가시키고 스택에 추가

S.push('(');

}

else if (str[i] == '[')

{

temp \*= 3;

S.push('[');

}

else if (str[i] == ')')

{

//괄호가 잘 닫히지 않은 경우

if (S.empty() || S.top() != '(')

{

ans = 0; break;

}

// 짝이 맞는 소괄호일 경우, 결과값에 현재 temp 값을 더하고 temp를 나누기 2

if (str[i - 1] == '(') ans += temp;

temp /= 2;

S.pop();

}

else if (str[i] == ']')

{

//괄호가 잘 닫히지 않은 경우

if (S.empty() || S.top() != '[')

{

ans = 0; break;

}

if (str[i - 1] == '[')

ans += temp;

temp /= 3;

S.pop();

}

}

if (!S.empty()) ans = 0;//괄호가 짝이 맞지 않는 경우

cout << ans;// 결과값 출력

return 0;

}

**# 연구조사**

이 문제는 주어진 괄호 문자열에서 올바른 괄호 쌍의 개수를 계산하는 문제이다. 여는 괄호 '('와 '['에 대해 각각 2배와 3배로 가중치를 부여하고, 닫는 괄호가 나올 때 해당 괄호와 짝이 맞으면 결과값에 가중치를 더하고 가중치를 나누어 줄여가며 계산한다. 괄호쌍이 잘 닫혔을 때 최종 결과값, 그렇지 않으면 문제에 나온대로 0을 출력한다. 가중치를 부여하는 부분은 ans에 더하고 빼지는 변수 temp를 이용했다.

**05-7. 3차 실습보고서\_연습문제#02\_스택 구현(순차 자료구조)**

**# 코드**

#include <iostream>

#define stackMAXSIZE 10

using namespace std;

inline void error(const char\* message);

// 배열 스택: arrayStack

template <typename E>

class arrayStack {

private:

E stack[stackMAXSIZE];

int top;

public:

arrayStack();

~arrayStack();

void push(const E& e);

E pop(void);

E peek(void) const;

int size(void) const;

bool Empty(void) const;

bool Full(void) const;

void printStack(void) const;

};

// 생성자: 빈 스택 생성

template <typename E>

arrayStack<E>::arrayStack() : top(-1) {}

// 소멸자

template <typename E>

arrayStack<E>::~arrayStack() {}

// PUSH : 스택에 데이터 삽입

template <typename E>

void arrayStack<E>::push(const E& e) {

if (Full())

error("스택 포화 에러");

stack[++top] = e;

}

// POP : 스택에서 데이터 삭제

template <typename E>

E arrayStack<E>::pop(void) {

if (Empty())

error("스택 공백 에러");

return stack[top--];

// top--;

}

// PEEK : 스택 맨 위의 원소 확인

template <typename E>

E arrayStack<E>::peek(void) const {

if (Empty())

error("스택 공백 에러");

return stack[top];

}

// SIZE : 스택의 크기 값 반환

template <typename E>

int arrayStack<E>::size(void) const {

return top + 1;

}

// 스택의 공백 상태 여부 판단

template <typename E>

bool arrayStack<E>::Empty(void) const {

return top == -1;

}

// 스택의 포화 상태 여부 판단

template <typename E>

bool arrayStack<E>::Full(void) const {

return top == stackMAXSIZE - 1;

}

// 스택의 전체 원소 출력

template <typename E>

void arrayStack<E>::printStack(void) const {

cout << "\nSTACK [";

for (int i = 0; i <= top; i++) {

cout.width(3);

cout << stack[i];

}

cout << " ]" << endl;

}

inline void error(const char\* message) {

cout << message;

exit(100);

}

int main(void)

{

int num, choice;

arrayStack<int> s = arrayStack<int>();// 빈 스택 생성

while (true) {

system("cls");

cout << " 20215229 이채훈 " << endl;

cout << "\n ### 스택 구현: 1차원 배열 ###\n" << endl;

cout << "1) 데이터 삽입: PUSH" << endl;

cout << "2) 데이터 삭제: POP" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

s.push(num);

break;

case 2: cout << "삭제 된 데이터: " << s.pop() << endl;

break;

case 3: s.printStack();

cout << "스택 크기: " << s.size() << endl;

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

}

return 0;

}

**# 연구조사**

**\* 에러 처리 함수 정의**

: `error` 함수는 주어진 메시지를 출력하고 프로그램을 종료시킴.

inline void error(const char\* message) {

cout << message;

exit(100);

}

* error("스택 포화 에러");
* error("스택 공백 에러");

**\* 스택 기본 동작 함수들**

`push`: 스택에 데이터를 삽입하는 함수입

`pop`: 스택에서 데이터를 삭제하고 해당 데이터를 반환하는 함수

`peek`: 스택의 맨 위에 있는 원소를 확인하는 함수

`size`: 스택의 현재 크기를 반환하는 함수

`Empty`: 스택이 비어있는지 확인하는 함수

`Full`: 스택이 꽉 차 있는지 확인하는 함수

`printStack`: 스택의 전체 원소를 출력하는 함수

**05-8. 3차 실습보고서\_연습문제#02\_스택 구현(순차 자료구조)ver.Ong**

**# 코드**

#include <iostream>

#define stackMAXSIZE 10

using namespace std;

template<typename E>

class arrStack

{

public:

arrStack();

~arrStack();

void push(const E& e); //데이터 삽입

void pop(); //데이터 삭제

E peek() const; //top 출력

int size() const; //스택 크기 출력

bool empty() const; //스택 빈 여부 확인

bool full() const; //스택 찬 여부 확인

void printStack() const;//스택 전체 출력

private:

E stack[stackMAXSIZE]; //스택 배열

int top; //스택의 탑(배열의 가장 큰 인덱스)

};

template<typename E>

arrStack<E>::arrStack() : top(-1) {}

template<typename E>

arrStack<E>::~arrStack() {}

template<typename E>

void arrStack<E>::push(const E& e)

{

if (full())

{

cout << "스택 포화 상태로 푸쉬 불가" << endl;

return;

}

stack[++top] = e;

}

template<typename E>

void arrStack<E>::pop()

{

if (empty())

{

cout << "스택이 비어 있음 " << endl;

return;

}

--top;

}

template<typename E>

E arrStack<E>::peek() const

{

if (empty()) return 0;

return stack[top];

}

template<typename E>

int arrStack<E>::size() const

{

if (empty()) return 0;

return (top + 1);

}

template<typename E>

bool arrStack<E>::empty() const

{

if (top == -1) return true;

else return false;

}

template<typename E>

bool arrStack<E>::full() const

{

if (top + 1 == stackMAXSIZE) return true;

else return false;

}

template<typename E>

void arrStack<E>::printStack() const

{

for (int i = 0; i <= top; i++) cout << stack[i] << " ";

}

int main()

{

int menu;

arrStack<int> st; //스택 클래스 선언

cout << "\n### 스택 구현: 1차원 배열 ###\n" << endl;

cout << "1) 데이터 삽입: PUSH" << endl;

cout << "2) 데이터 삭제: POP" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료" << endl << endl;

while (true)

{

cout << "메뉴 선택 : "; cin >> menu;

switch (menu) //선택한 번호에 따라 기능 실행

{

case 1: //1) 데이터 삽입

int input;

cout << "삽입할 데이터 입력: "; cin >> input;

st.push(input); break;

case 2: //2) 데이터 삭제

st.pop(); break;

case 3: //3) 전체 출력

st.printStack(); cout << endl;

break;

case 4: //4) 프로그램 종료

return 0;

}

cout << endl;

}

return 0;

}