[프로젝트 과제]

템플릿 MyStack을 이용한 사칙연산 구현

2017156034 전상민

**프로젝트 과제 #1**

è10장에서 학습한 ‘템플릿’에서 MyStack (스택)을 이용한 사칙연산 계산기를 작성해 제출하시오.( 본 과제는 리포트 양식에 맞춰 구현 로직, 구조, 소스설명, 결과, 소스)를 도출해 제출해 주시길 바랍니다.

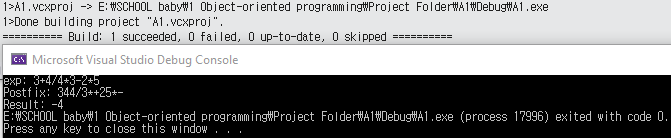
è조건 1: 9장에 추상 클래스와 이에 따른 파생클래스를 이용해 사칙연산(+, -, \*, / ) 되도록 구현할 것

è조건 2: 후위표기식으로 수식(ex, (3\*4)+(4/4) )을 표현해 MyStack을 구성해 계산할 것 (아래 링크 확인)  
<https://repl.it/@yangkim/2020-Data-Structure#DataStrcuture/Chap.4/postfix_stack.c>  
후위표기식:  (1+2)\*7  -> 12+7\*

è조건 3: 우선순위(괄호안에 수식 먼저 계산, \*, /의 경우 계산함) 기반의 수식 계산이 되게 할 것 (괄호는 제외하고 그냥 사칙연산 \*, /, +, - 만으로 구성해도 됨

**수식: 3+4/4\*3-2\*5 의 결과값을 과제 리포트에 결과로 제시할 것**

# 결과



# 구조

//virtual class and derived class

class Calculator {

è조건 1: 9장에 추상 클래스와 이에 따른 파생클래스를 이용해 사칙연산(+, -, \*, / ) 되도록 구현할 것

//template stack

template <class T>

class MyStack {

è10장에서 학습한 ‘템플릿’에서 MyStack (스택)을 이용한 사칙연산 계산기를 작성해 제출하시오.

//infix to postfix to calculation

int isOperand(char ch)

int Prec(char ch)

void InfixToPostfix(char\* exp)

int PostfixCalc(char\* exp)

è조건 2: 후위표기식으로 수식(ex, (3\*4)+(4/4) )을 표현해 MyStack을 구성해 계산할 것 (아래 링크 확인)  
<https://repl.it/@yangkim/2020-Data-Structure#DataStrcuture/Chap.4/postfix_stack.c>  
후위표기식:  (1+2)\*7  -> 12+7\*

è조건 3: 우선순위(괄호안에 수식 먼저 계산, \*, /의 경우 계산함) 기반의 수식 계산이 되게 할 것 (괄호는 제외하고 그냥 사칙연산 \*, /, +, - 만으로 구성해도 됨

int main()

{

1. 수식 입력.

char exp[] = "3+4/4\*3-2\*5";

cout << "exp: " << exp << endl;

1. 수식을 후위표기식 꼴로 변환.

cout << "Postfix: "; InfixToPostfix(exp);

1. 후위표기식 꼴의 수식을 계산.

cout << "Evaluation: " << PostfixCalc(exp);

}

# 알고리즘

## InfixToPostfix(char\* exp)

중위표기에서 후위표기로 변환하는 함수.

1. 중위표기식을 읽는다.
2. 읽은 문자가 숫자면 출력한다.
3. 읽은 문자가 연산자면…
   1. 연산자의 우선순위가 스택에 이미 있는 연산자보다 높다면 push한다.
   2. 연산자의 우선순위가 스택에 이미 있는 연산자보다 낮다면 이미 있는 연산자부터 그와 같은 우선순위를 갖는 연산자까지 pop한다. 그 다음 읽은 연산자를 push한다. 만약 ‘(‘가 있다면 거기서 멈추고 읽은 연산자를 push한다.
4. 읽은 문자가 ‘(‘이면 push한다.
5. 읽은 문자가 ‘)’이면 ‘(‘까지 pop하고 괄호를 없앤다.
6. 2~6단계를 반복한다.
7. 위와 스택에 남아있는 것까지 모두 출력한다.

## PostfixCalc(char\* exp)

후위표기식을 게산하는 함수.

1. 후위표기식을 읽는다.
2. 읽은 문자가 숫자면 push 한다.
3. 읽은 문자가 연산자이면 스택에 있는 두 숫자를 pop하고, 그 연산자에 맞게 계산을 한다. 그 후 그 결과를 push한다.
4. 문자를 모두 읽으면 최종 결과가 남는다.

예를 들어, “(1+2)\*7”이 변환된 “12+7\*”이면,

1. 스택에 1을 push한다.
2. 스택에 2를 push한다.
3. +를 스캔하여 스택에 있던 1과 2를 pop하고 +한다. 그 결과값인 3을 push한다.
4. 스택에 7를 push한다.
5. \*를 스캔하여 스택에 있던 3과 7를 pop하고 \*한다. 그 결과값인 21을 push한다.
6. 읽을 문자가 없다. 남아있는 값 21이 최종 결과이다.

# 소스 설명

**//virtual class and derived class**

## class Calculator {

추상 클래스와 파생 클래스를 이용한 계산기 클래스 생성.

protected: -추상 클래스

int x, y;

virtual int calc(int x, int y) = 0; -순수 가상 함수

public:

int run(int x, int y) {

return calc(x, y);

}

};

-파생 클래스, 순수 가상 함수 구현

class Adder : public Calculator {

protected:

int calc(int a, int b) {

return a + b;

}

};

class Subtractor : public Calculator {

protected:

int calc(int a, int b) {

return a - b;

}

};

class Multiplier : public Calculator {

protected:

int calc(int a, int b) {

return a \* b;

}

};

class Divider : public Calculator {

protected:

int calc(int a, int b) {

return a / b;

}

};

**//template stack**

## template <class T> class MyStack {

템플릿 스택 호출.

int tos; - 맨 위 요소

T data[MAX]; - 스택 크기

public:

MyStack(); - 생성자

void push(T element); - 스택 요소 삽입

T pop(); - 스택 요소 삭제

T peek(); - 맨 위 요소 반환

int isEmpty(); - 비어있는지 검사

### template <class T> MyStack<T>::MyStack() {

생성자.

tos = -1; - 스택을 비운다.

}

### template <class T> void MyStack<T>::push(T element) {

스택 요소 삽입.

if (tos == MAX - 1) { - 스택이 차 있는지를 체크하고

cout << "stack full";

return;

}

tos++; - 스택 높이 증가

data[tos] = element; - 스택 요소 삽입

### template <class T> T MyStack<T>::pop() {

스택 요소 삭제 및 반환.

T retData;

if (tos == -1) { - 스택이 비었는지를 체크하고

cout << "stack empty";

return 0;

}

retData = data[tos--]; - 스택 높이 감소 및 값 정의

return retData; - 값 반환

}

### template <class T> T MyStack<T>::peek() {

스택 요소 반환.

T retData;

if (tos == -1) { - 스택이 비었는지를 체크하고

cout << "stack empty";

return 0;

}

retData = data[tos]; - 값 정의

return retData; - 값 반환

}

### template <class T> int MyStack<T>::isEmpty() {

스택이 비어있는지를 체크.

return tos == -1;

}

**//infix to postfix to calculation**

## int isOperand(char ch) {

연산자인지 아닌지 구분하는 함수.

return (ch >= '0' && ch <= '9'); - 숫자인지 문자인지 구분한다.

}

## int Prec(char ch) {

우선순위를 구분하기 위한 함수.

switch (ch)

{

case '+':

case '-':

return 1; - 우선순위 1

case '\*':

case '/':

return 2; - 우선순위 2

}

return -1;

우선순위 값이 높을수록 먼저 계산한다.

‘^’는 왜 포함되어 있었는지 모르겠으나 사용하지 않으므로 제외함.

물론 하나의 우선순위 그룹을 더 만들고 파생 클래스를 하나 더 만들어 ‘^’의 기능까지 포함시킬 수 있다.

## void InfixToPostfix(char\* exp) {

중위표기에서 후위표기로 변환하는 함수.

MyStack<int> stack; - 스택 생성

int i, j; - i: front (앞쪽), j: rear (뒤쪽)

for (i = 0, j = -1; exp[i]; ++i)

{

if (isOperand(exp[i])) - 숫자면 출력하고 출력 스택에 push

exp[++j] = exp[i];

else if (exp[i] == '(') - ‘(’이면 push

stack.push(exp[i]);

else if (exp[i] == ')') - ‘)’이면 pop.

{ - 만약 괄호안에 괄호가 있으면 그 때까지 pop

while (!stack.isEmpty() && stack.peek() != '(')

exp[++j] = stack.pop();

if (stack.isEmpty() || !stack.peek() != '(')

stack.pop();

}

else - 연산자면 우선순위에 맞게 pop 하고 push

{

while (!stack.isEmpty() && Prec(exp[i]) <= Prec(stack.peek()))

exp[++j] = stack.pop();

stack.push(exp[i]);

}

}

while (!stack.isEmpty()) - 남은 요소들을 모두 pop.

exp[++j] = stack.pop();

exp[++j] = '\0'; - 출력 스택 마무리

cout << exp << endl; - 출력

}

## int PostfixCalc(char\* exp) {

후위표기식을 게산하는 함수.

MyStack<int> stack; - 스택 생성

Adder Add; - 클래스 호출

Subtractor Sub;

Multiplier Mul;

Divider Div;

int i;

for (i = 0; exp[i]; ++i)

{

if (isdigit(exp[i])) - 숫자면 push

stack.push(exp[i] - '0');

else { - 연산자면 숫자 2개를 꺼내 연산자에 맞게 계산하고 push

int val1 = stack.pop();

int val2 = stack.pop();

switch (exp[i]) { - 파생 클래스를 이용한 계산기 함수 사용

case '+': stack.push(Add.run(val1, val2)); break;

case '-': stack.push(Sub.run(val2, val1)); break;

case '\*': stack.push(Mul.run(val1, val2)); break;

case '/': stack.push(Div.run(val2, val1)); break;

}

}

}

return stack.pop(); - 최종 값 반환

}

## int main() {

char exp[] = "3+4/4\*3-2\*5";

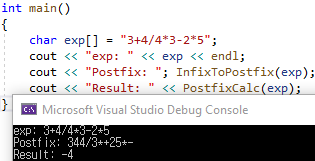
cout << "exp: " << exp << endl; - 계산 식 입력

cout << "Postfix: "; InfixToPostfix(exp); - 중위표기식을 후위표기식으로 변환

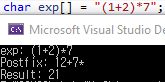
cout << "Evaluation: " << PostfixCalc(exp); - 후위표기식 형식의 수식 계산

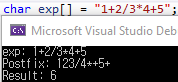
}

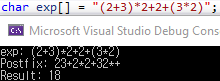
# 결과



## Extra







# 소스

#include <iostream>

#include <string.h>

using namespace std;

#define MAX 100

//virtual class and derived class

class Calculator {

protected:

int x, y;

virtual int calc(int x, int y) = 0;

public:

int run(int x, int y) {

return calc(x, y);

}

};

class Adder : public Calculator {

protected:

int calc(int a, int b) {

return a + b;

}

};

class Subtractor : public Calculator {

protected:

int calc(int a, int b) {

return a - b;

}

};

class Multiplier : public Calculator {

protected:

int calc(int a, int b) {

return a \* b;

}

};

class Divider : public Calculator {

protected:

int calc(int a, int b) {

return a / b;

}

};

//template stack

template <class T>

class MyStack {

int tos;

T data[MAX];

public:

MyStack();

void push(T element);

T pop();

T peek();

int isEmpty();

};

template <class T>

MyStack<T>::MyStack() {

tos = -1;

}

template <class T>

void MyStack<T>::push(T element) {

if (tos == MAX-1) {

cout << "stack full";

return;

}

tos++;

data[tos] = element;

}

template <class T>

T MyStack<T>::pop() {

T retData;

if (tos == -1) {

cout << "stack empty";

return 0;

}

retData = data[tos--];

return retData;

}

template <class T>

T MyStack<T>::peek() {

T retData;

if (tos == -1) {

cout << "stack empty";

return 0;

}

retData = data[tos];

return retData;

}

template <class T>

int MyStack<T>::isEmpty() {

return tos == -1;

}

//infix to postfix to calculation

int isOperand(char ch)

{

return (ch >= '0' && ch <= '9');

}

int Prec(char ch)

{

switch (ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

}

return -1;

}

void InfixToPostfix(char\* exp)

{

MyStack<int> stack;

int i, j;

for (i = 0, j = -1; exp[i]; ++i)

{

if (isOperand(exp[i]))

exp[++j] = exp[i];

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

stack.push(exp[i]);

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

{

while (!stack.isEmpty() && stack.peek() != '(')

exp[++j] = stack.pop();

if (stack.isEmpty() || !stack.peek() != '(')

stack.pop();

}

else

{

while (!stack.isEmpty() && Prec(exp[i]) <= Prec(stack.peek()))

exp[++j] = stack.pop();

stack.push(exp[i]);

}

}

while (!stack.isEmpty())

exp[++j] = stack.pop();

exp[++j] = '\0';

cout << exp << endl;

}

int PostfixCalc(char\* exp)

{

MyStack<int> stack;

Adder Add;

Subtractor Sub;

Multiplier Mul;

Divider Div;

int i;

for (i = 0; exp[i]; ++i)

{

if (isdigit(exp[i]))

stack.push(exp[i] - '0');

else {

int val1 = stack.pop();

int val2 = stack.pop();

switch (exp[i]) {

case '+': stack.push(Add.run(val1, val2)); break;

case '-': stack.push(Sub.run(val2, val1)); break;

case '\*': stack.push(Mul.run(val1, val2)); break;

case '/': stack.push(Div.run(val2, val1)); break;

}

}

}

return stack.pop();

}

int main()

{

char exp[] = "3+4/4\*3-2\*5";

cout << "exp: " << exp << endl;

cout << "Postfix: "; InfixToPostfix(exp);

cout << "Result: " << PostfixCalc(exp);

}