과목: 자료구조

교수: 최재현 교수님

자료구조 <프로젝트>

- Calculator 클래스 작성 -

홍지훈

이름: 홍지훈

학과: 소프트웨어학부

분반: 나

학번 : 20201777

0. 과제

수식을 후위표기식으로 변경하고, 계산하는 Calculator 클래스 작성

-- 과제3 에서 작성한 List 클래스를 template 를 추가하여 변경

-- 과제4 에서 작성한 String 클래스를 사용

클래스의 기본 정의는 아래와 같음

class Calculator

{

   private :

      List<String> tokens;

      int errCode;                  // 발생된 오류코드 값 : 0 -> 오류 없음 , 다른 값 -> 오류 있음

      int value;                     // 계산된 값

      String postfix:              // 후위표기식 = 최초 공백으로 초기화

      int makePostFix();         // postfix 로 변경하는 함수 :

                                       // 오류 없는경우, 0, 오류가 있는 경우, 1을 반환

                                       // 변경결과는 postfix 변수에 저장, 오류시 적절한 코드를 errCode 에 저장 (오류코드는 각자가 정의)

      int evaluation();            // postfix 를 계산하는 함수 :

                                       // 계산된 값을 구함, 오류 없는경우 0, 오류가 있는 경우, 1을 반환

                                       // 계산된 값은 value 에 저장, 오류시 적절한 코드를 errCode 에 저장 (오류코드는 각자가 정의)

  public :

      int getErrorCode();                             // 오류코드 반환

      int setExpression(const char\* expr);       // expr에 전달된 수식(중위표기식)을 postfix로 변경하고 계산하는 함수

                                                            //    오류 없는경우, 0, 오류가 있는 경우, 1을 반환

       String getPostFix();                            // 변환된 후위표기식을 반환   --> 오류가 있을경우 최초값인 공백이 리턴

       int getValue();                                   // 수식 오류있음 --> 예외발생

                                                             // 수식 오류없음 --> 결과값 리턴

};

위에 기본정의된 함수외에 각자 필요하다고 생각하다고 하는 함수는 얼마든지 추가하여 구현가능

<< setExpression 예시 >>

   if ( !makePostfix )

        return evaluation();

   else

        return 1;

<< main 함수 예시 >>

char expr[1000];

Calculator c;

cout << "수식을 입력하시오 : ";

cin.getline(expr, 1000);

if ( !c.setExpression(expr) )

{

    String posfix = c.getPostFix();

    try {

       cout << "후위표기식 : " << postfix  << "결과값 : " << c.getValue() << endl;

    } catch (const char\* errmsg)

    {

          cout << errmsg << endl;

     }

}

※ List<String> 은 expression 을 postfix 로 변경하기 expession 전달된 문자열을 토큰리스트로 변경하는 데 사용

    즉, 괄호, 연산자, 피연산자를 구분해서 하나의 단위로 저장

    (토큰 : 공백 또는 의미있는 단위로 잘려진 문자열 )

   예) (3+5)/10 을 List<String>으로 변경 :    tokens[0] = "("

                                                            tokens[1] = "3"

                                                            tokens[2] = "+"

                                                            tokens[3] = "5" ......

※ Postfix 로 변경할때 List<String> 에 저장된 토큰들을 가지고, 강의내용에서 설명한 알고리즘을 적용하여 변경

1. 소스코드

1-1. List.h

#pragma once

#include <iostream>

using namespace std;

template<class T>

class List

{

private:

    T\* items;

    int itemCount;

    int size;

public:

    List();

    ~List();

    T getItem(int index);

    int Length();

    void addItem(T itm);

    void insertItem(int index, int itm);

    T removeAt(int index);

    int removeItem(T itm);

    void concat(List& list);

    void print();

};

template<class T>

List<T>::List() {

    itemCount = 0;

    size = 5;

    items = new T[size];

}

template<class T>

List<T>::~List() {

    delete[] items;

}

template<class T>

T List<T>::getItem(int index) {

    if (index < 0 || index >= itemCount) throw "out of range";

    return items[index];

}

template<class T>

int List<T>::Length() { return itemCount; }

template<class T>

void List<T>::addItem(T itm) {

    if (itemCount < size) {

        items[itemCount] = itm;

        itemCount++;

    }

    else {

        T\* newItem = new T[size \* 2];

        for (int i = 0; i < itemCount; i++)

            newItem[i] = items[i];

        newItem[itemCount] = itm;

        itemCount++;

        delete[] items;

        items = newItem;

        size = size \* 2;

    }

}

template<class T>

void List<T>::insertItem(int index, int itm) {

    if (index >= 0 && index < itemCount) {

        itemCount++;

        if (itemCount >= size) {

            int\* newItem = new T[size \* 2];

            for (int i = 0; i < itemCount - 1; i++)

                newItem[i] = items[i];

            delete[] items;

            items = newItem;

            size = size \* 2;

        }

        for (int i = itemCount; i > index; i--)

            items[i] = items[i - 1];

        items[index] = itm;

    }

    else

        cout << "처리할 수 없습니다. index값은 0~" << itemCount << "사이의 값이여야 합니다." << endl;

}

template<class T>

T List<T>::removeAt(int index) {

    if (index <= 0 || index >= itemCount)

        return -99999;

    int temp = items[index];

    for (int i = index; i < itemCount; i++)

        items[i] = items[i + 1];

    itemCount--;

    return temp;

}

template<class T>

int List<T>::removeItem(T itm) {

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

        if (items[i] == itm) {

            int temp = i;

            for (int j = i; j < itemCount; j++)

                items[j] = items[j + 1];

            itemCount--;

            return temp;

        }

    }

    return -99999;

}

template<class T>

void List<T>::concat(List& list) {

    for (int i = 0; i < list.itemCount; i++) {

        addItem(list.items[i]);

    }

}

template<class T>

void List<T>::print()

{

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

        cout << items[i] << " ";

        if (i == itemCount - 1)

            continue;

        cout << "-> ";

    }

    cout << endl;

}

1-2. Stack.h

#pragma once

#include <iostream>

using namespace std;

//cpp파일을 따로 나누면 오류가 생겨서 헤더파일에 합쳤습니다.

template<class T>

class Stack

{

private:

    T\* stack;

    int top;

    int capacity;

public:

    Stack(int stackCapacity = 10);

    bool IsEmpty() const;

    T& Top() const;

    void Push(const T& item);

    void Pop();

    void print();

};

template <class T>

void ChangeSize1D(T\*& a, const int oldSize, const int newSize)

{

    if (newSize < 0) throw "New length must be >= 0";

    T\* temp = new T[newSize];

    int number = min(oldSize, newSize);

    copy(a, a + number, temp);

    delete[] a;

    a = temp;

}

template<class T>

Stack<T>::Stack(int stackCapacity) : capacity(stackCapacity)

{

    if (capacity < 1) throw "Stack capacity must be > 0";

    stack = new T[capacity];

    top = -1;

}

template<class T>

inline bool Stack<T>::IsEmpty() const { return top == -1; }

template<class T>

inline T& Stack<T>::Top() const

{

    if (IsEmpty()) throw "Stack is empty";

    return stack[top];

}

template<class T>

void Stack<T>::Push(const T& x)

{

    if (top == capacity - 1)

    {

        ChangeSize1D(stack, capacity, 2 \* capacity);

        capacity \*= 2;

    }

    stack[++top] = x;

}

template<class T>

void Stack<T>::Pop()

{

    if (IsEmpty()) throw "Stack is empty. Cannot delete.";

    stack[top--].~T();

}

template<class T>

void Stack<T>::print()

{

    for (int i = 0; i <= top; i++) {

        cout << stack[i] << " ";

        if (i == top)

            continue;

        cout << "-> ";

    }

    cout << endl;

}

1-3. String.h

#pragma once

#include <iostream>

using namespace std;

class String

{

private:

    char\* buffer;

    int length;

    int size;

    String(int m);

    friend ostream& operator<<(ostream& os, String& s);

public:

    String();

    String(String&); //복사생성자

    String(char\* init, int m); //길이m string init 초기화

    String(char init); //char to string

    ~String();

    String Concat(String t);

    String& operator=(const String&); // 대입

    String& operator=(const char\* s);

    bool operator==(String t); // 동일한지 비교

    bool operator!(); // 공백이면 true 아니면 false

    String& itos(const int s); //int to string

    int Length(); // 문자수 반환

    String Substr(int i, int j); // i~j사이의 string 반환

    int Find(String pat); //string에서 pat스트링을 찾아서 위치를 반환 없으면 -1

    char At(int pos);

    int to\_int(); // 숫자만 뽑아서 int로 반환

    void print();

};

1-4. String.cpp

#include "String.h"

String::String() : String(10) {}

String::String(int m)

{

    size = m;

    buffer = new char[m];

    length = 0;

}

String::~String()

{

    delete[] buffer;

    size = 0;

    length = 0;

}

String::String(String& s) : String(s.length + 1)

{

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

        buffer[i] = s.buffer[i];

    buffer[s.length] = '\0';

    length = s.length;

}

String::String(char\* init, int m) : String(m + 1)

{

    for (int i = 0; i < m; i++)

        buffer[i] = init[i];

    buffer[m] = '\0';

    length = m;

}

String::String(char init) : String(2)

{

    buffer[0] = init;

    if (init == '\0')

        length = 0;

    else {

        buffer[1] = '\0';

        length = 1;

    }

}

String String::Concat(String t)

{

    String result(length + t.length + 1);

    for (int i = 0; i < length; i++)

        result.buffer[i] = buffer[i];

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

        result.buffer[length + i] = t.buffer[i];

    result.buffer[length + t.length] = '\0';

    result.length = length + t.length;

    return result;

}

String& String::operator=(const String& s)

{

    if(size > 0)

        delete[] buffer;

    size = 0;

    while (s.buffer[size++] != '\0');

    buffer = new char[s.length + 1];

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

        buffer[i] = s.buffer[i];

    buffer[s.length] = '\0';

    length = s.length;

    return \*this;

}

String& String::operator=(const char\* s)

{

    if (size > 0)

        delete[] buffer;

    size = 0;

    while (s[size++] != '\0');

    buffer = new char[size];

    this->size = size;

    for (int i = 0; i < size; i++) {

        if (s[i] == '\0') length = i;

        buffer[i] = s[i];

    }

    return \*this;

}

bool String::operator==(String t)

{

    if (length != t.length) return false;

    for (int i = 0; i < length; i++)

        if (buffer[i] != t.buffer[i])

            return false;

    return true;

}

bool String::operator!()

{

    if (length == 0)

        return true;

    return false;

}

String& String::itos(const int s)

{

    if (size > 0)

        delete[] buffer;

    size = 0;

    int cnt = 1;

    int check = 10;

    while (check < s) {

        check \*= 10;

        cnt++;

    }

    size = cnt + 1;

    buffer = new char[size];

    this->size = size;

    for (int i = 0; i < cnt; i++) {

        check /= 10;

        buffer[i] = (s / check % 10) + '0';

    }

    buffer[cnt] = '\0';

    length = cnt;

    return \*this;

}

int String::Length()

{

    return length;

}

String String::Substr(int i, int j)

{

    String result(j - i + 2);

    result.length = j - i + 2;

    for (int k = 0; k < j - i + 1; k++)

        result.buffer[k] = buffer[k + i];

    result.buffer[j - i + 1] = '\0';

    return result;

}

int String::Find(String pat)

{

    for (int i = 0; i + pat.length + 1 < length; i++) {

        bool suc = true;

        for (int j = 0; j < pat.length; j++) {

            if (i + j >= length || buffer[i + j] != pat.buffer[j]) {

                suc = false;

                break;

            }

        }

        if (suc) return i;

    }

    return -1;

}

int String::to\_int()

{

    int value = 0;

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

        if (buffer[i] < '0' || buffer[i] > '9')

            continue;

        value \*= 10;

        value += buffer[i] - '0';

    }

    return value;

}

char String::At(int pos) {

    if (pos <= length)

        return buffer[pos];

    throw "Out of buffer";

}

void String::print()

{

    for (int i = 0; i < length; i++)

        cout << buffer[i];

    cout << endl;

}

1-5. Calculator.h

#pragma once

#include <iostream>

#include "List.h"

#include "String.h"

#include "Stack.h"

class Calculator

{

private:

    List<String> tokens;

    int errCode;                  // 발생된 오류코드 값 : 0 -> 오류 없음 , 1 -> 공백오류 2 -> 구문오류 3 -> 잘못된 부호

    int value;                     // 계산된 값

    String postfixStr;              // 후위표기식 = 최초 공백으로 초기화

    List<String> postfix;        // 후위표기식 리스트

    int makePostFix();         // postfix 로 변경하는 함수 :

                                     // 오류 없는경우, 0, 오류가 있는 경우, 1을 반환

                                     // 변경결과는 postfix 변수에 저장, 오류시 적절한 코드를 errCode 에 저장 (오류코드는 각자가 정의)

    int evaluation();            // postfix 를 계산하는 함수 :

                                     // 계산된 값을 구함, 오류 없는경우 0, 오류가 있는 경우, 1을 반환

                                     // 계산된 값은 value 에 저장, 오류시 적절한 코드를 errCode 에 저장 (오류코드는 각자가 정의)

public:

    Calculator();

    int getErrorCode();                             // 오류코드 반환

    int setExpression(const char\* expr);       // expr에 전달된 수식(중위표기식)을 postfix로 변경하고 계산하는 함수 ,오류 없는경우, 0, 오류가 있는 경우, 1을 반환

    String getPostFix();                            // 변환된 후위표기식을 반환   --> 오류가 있을경우 최초값인 공백이 리턴

    int getValue();                                   // 수식 오류있음 --> 예외발생

                                                          // 수식 오류없음 --> 결과값 리턴

};

1-6. Calculator.cpp

#include "Calculator.h"

#include "Stack.h"

#include "String.h"

Calculator::Calculator() {

    errCode = 0;

    value = 0;

    postfixStr = " ";

}

int Calculator::makePostFix() {

    Stack<String> stack;

    String temp;

    for (int i = 0; i < tokens.Length(); i++) {

        temp = tokens.getItem(i);

        if(temp.At(0) >= '0' && temp.At(0) <= '9') {

            postfix.addItem(temp);

            postfixStr = postfixStr.Concat(temp);

            postfixStr = postfixStr.Concat(' ');

        }

        else {

            switch (temp.At(0)) {

                case '(':

                    if (i > 0 && tokens.getItem(i - 1).At(0) >= '0' && tokens.getItem(i - 1).At(0) <= '9') {

                        errCode = 2;

                        return 2;

                    }

                    stack.Push(temp);

                    break;

                case ')':

                    while (!stack.IsEmpty() && stack.Top().At(0) != '(') {

                        postfix.addItem(stack.Top());

                        postfixStr = postfixStr.Concat(stack.Top());

                        postfixStr = postfixStr.Concat(' ');

                        stack.Pop();

                    }

                    if (stack.IsEmpty()) {

                        errCode = 2;

                        return 2;

                    }

                        stack.Pop();

                    break;

                case '+':

                case '-':

                    while (!stack.IsEmpty() && (stack.Top() == '+' || stack.Top() == '-' || stack.Top() == '\*' || stack.Top() == '/')) {

                        postfix.addItem(stack.Top());

                        postfixStr = postfixStr.Concat(stack.Top());

                        postfixStr = postfixStr.Concat(' ');

                        stack.Pop();

                    }

                    stack.Push(temp);

                    break;

                case '\*':

                case '/':

                    while (!stack.IsEmpty() && (stack.Top() == '\*' || stack.Top() == '/')) {

                        postfix.addItem(stack.Top());

                        postfixStr = postfixStr.Concat(stack.Top());

                        postfixStr = postfixStr.Concat(' ');

                        stack.Pop();

                    }

                    stack.Push(temp);

                    break;

                default:

                    errCode = 3;

                    return 3;

                    break;

            }

        }

    }

    while (!stack.IsEmpty()) {

        if (stack.Top() == '(') {

            errCode = 2;

            return 2;

        }

        postfix.addItem(stack.Top());

        postfixStr = postfixStr.Concat(stack.Top());

        postfixStr = postfixStr.Concat(' ');

        stack.Pop();

    }

    return 0;

}

int Calculator::evaluation() {

    Stack<String> stack;

    String n1, n2;

    int res = 0;

    for (int i = 0; i < postfix.Length(); i++) {

        if (postfix.getItem(i).At(0) == '+'

            || postfix.getItem(i).At(0) == '-'

            || postfix.getItem(i).At(0) == '\*'

            || postfix.getItem(i).At(0) == '/') {

            if (stack.IsEmpty()) {

                errCode = 2;

                return 2;

            }

            n1 = stack.Top();

            stack.Pop();

            if (stack.IsEmpty()) {

                errCode = 2;

                return 2;

            }

            n2 = stack.Top();

            stack.Pop();

            int res = 0;

            switch (postfix.getItem(i).At(0))

            {

            case '+':

                res = n2.to\_int() + n1.to\_int();

                break;

            case '-':

                res = n2.to\_int() - n1.to\_int();

                break;

            case '\*':

                res = n2.to\_int() \* n1.to\_int();

                break;

            case '/':

                res = n2.to\_int() / n1.to\_int();

                break;

            default:

                errCode = 2;

                return 2;

                break;

            }

            n2.itos(res);

            stack.Push(n2);

        }

        else {

            stack.Push(postfix.getItem(i));

        }

    }

    n1 = stack.Top();

    stack.Pop();

    if (!stack.IsEmpty()) {

        errCode = 2;

        return 2;

    }

    value = n1.to\_int();

    return 0;

}

int Calculator::getErrorCode() { return errCode; }

int Calculator::setExpression(const char\* expr){      // expr에 전달된 수식(중위표기식)을 postfix로 변경하고 계산하는 함수       //    오류 없는경우, 0, 오류가 있는 경우, 1을 반환

    int size = 0;

    String word = '\0';

    while (expr[size++] != '\0') {

        if (expr[size - 1] == ' ') continue;

        if (!word == false) {

            if (expr[size - 1] >= '0' && expr[size - 1] <= '9') {

                if (expr[size - 2] < '0' || expr[size - 2] > '9') {

                    errCode = 1;

                    return 1;

                }

                word = word.Concat(expr[size - 1]);

            }

            else {

                tokens.addItem(word);

                word = '\0';

            }

        }

        if (!word == true) {

            if (expr[size - 1] >= '0' && expr[size - 1] <= '9') {

                word = expr[size - 1];

            }

            else tokens.addItem(expr[size - 1]);

        }

    }

    if(!word == false)

        tokens.addItem(word);

    if (!makePostFix())

        return evaluation();

    else return 1;

}

String Calculator::getPostFix() {

    if (errCode)

        return ' ';

    return postfixStr;

}

int Calculator::getValue() {

    if (errCode)

        throw "error";

    else

        return value;

}

1-7. Main.cpp

#include "Calculator.h"

#include "String.h"

using namespace std;

ostream& operator<<(ostream& os, String& s) {

    os << s.buffer;

    return os;

}

int main(void) {

    char expr[1000];

    Calculator cal;

    String postfix;

    cout << "수식을 입력하시오 : ";

    cin.getline(expr, 1000);

    cal.setExpression(expr);

    postfix = cal.getPostFix();

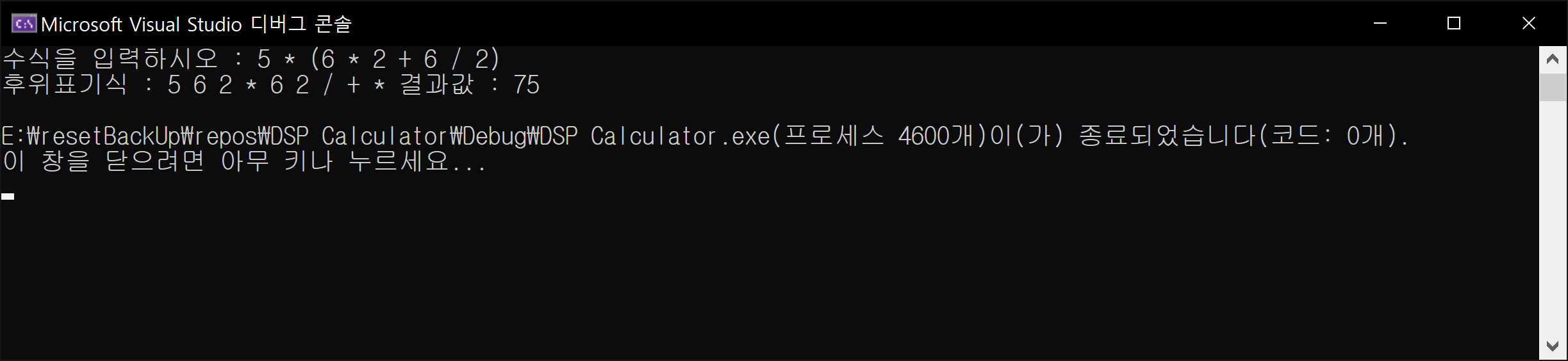
    cout << "후위표기식 :" << postfix << "결과값 : " << cal.getValue() << endl;

    return 0;

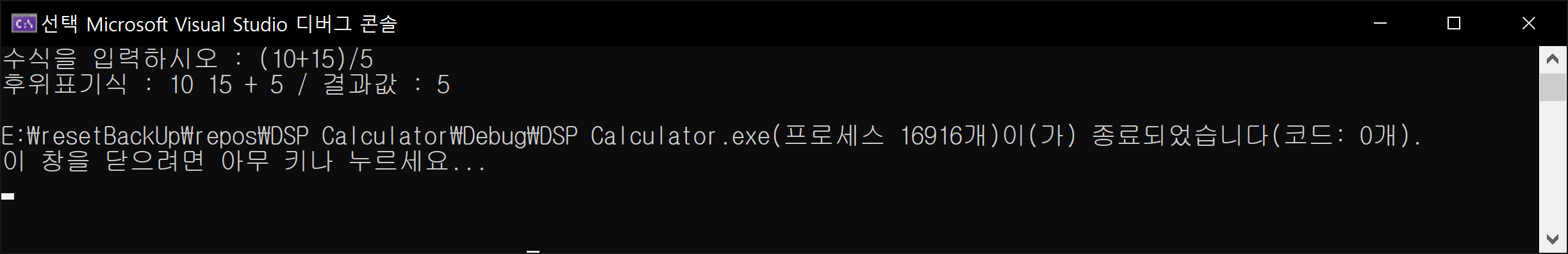
}

2. 실행 화면

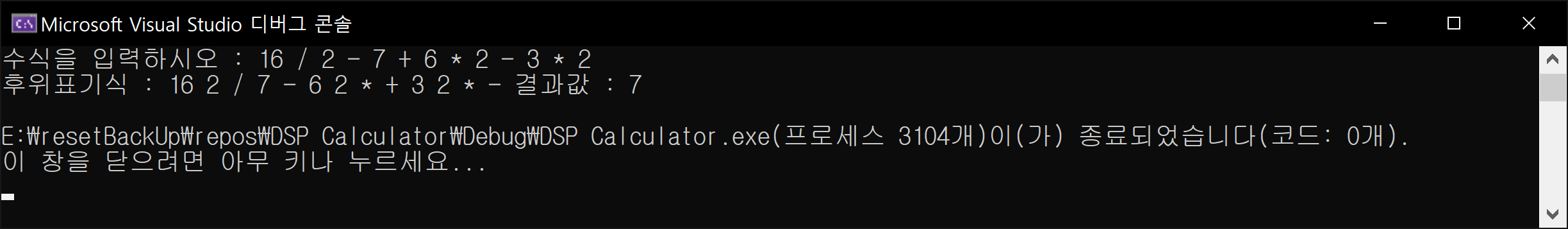
(1) 5\*(6\*2+6/2)



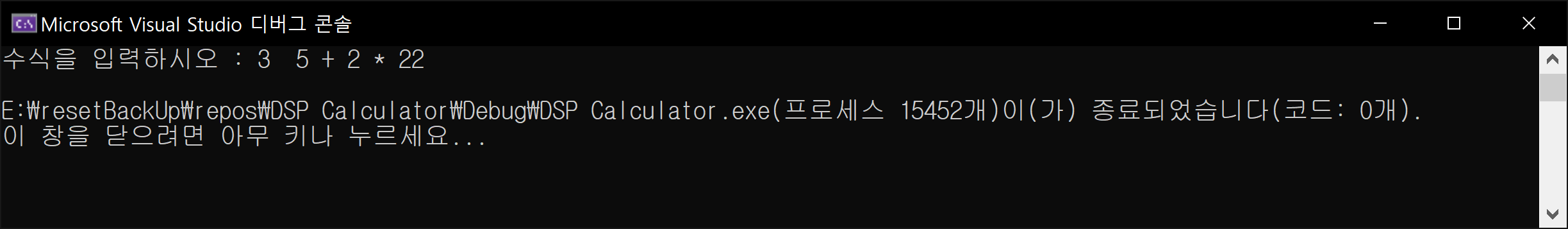
(2)(3+5)/10



(3) 16 / 2 - 7 + 6 \* 2 - 3 \* 2



(4) err1



3. 특이 사항

소수점과 관련된 내용에 딱히 언급이 없으셔서 정수만 처리하도록 하였습니다.