7

제출일 : 2022-11-05

KWANGWOON UNIVERSITY

담당교수	김용혁 교수님	학 번	2021203034
학 과	소프트웨어학부	이 름	허찬영

KWANGWOON UNIVERSITY

my environment : Visual Studio 2022 (Windows)

■ Begin with a summary of my results

I write a C++ program that inputs general real-numerical expressions line by line and then outputs their results. “EOI” means the end of input.

■ For each mission, explain how I completed it

○ main

```
while (true) {
    getline(cin, infix);

    if (!infix.compare(EOI))
        break;

    if (!checkPARENTHESIS(infix)) {
        cout << "Error!: unbalanced parentheses\n";
        continue;
    }

    postfix = inToPos(infix);
    ans = computePostfix(postfix);
    if (checkDividedByZero == true) {
        cout << "Error!: divided by zero\n";
        checkDividedByZero = false;
        continue;
    }
    else {
        cout << fixed;
        cout.precision(3);
        cout << ans << '\n';
    }
}
```

위 사진은 main 함수 안에 구현된 while문이다. getline()함수를 호출하여 중위 표기식인 string 타입의 infix에 문자열을 저장한다. 만약 EOI가 입력될 경우 반복문을 빠져 나오고 프로그램을 종료한다. checkPARENTHESIS()함수는 괄호가 정상적으로 입력되었는지 확인하는 함수로 괄호가 정상적으로 입력되지 않을 경우 false를 반환하고 오류 메시지를 출력한다. 입력이 정상적으로 될 경우 inToPos() 함수를 통해 중위 표기식으로 입력된 문자열을 후위 표기식으로 변환한 후 저장한다. 그 후, computePostfix() 함수를 호출하여 후위 표기식의 연산을 진행한 후 ans에 저장한다. 만약 0으로 나누는 연산이 발생해 checkDividedByZero가 true가 되면 오류 메시지를 출력한다. 연산이 정상적으로 진행될 경우 소수점 네 번째 자리에서 반올림하여 출력한다.

```
string inToPos(const string);
bool getOrder(char, char);
bool checkPARENTHESIS(const string);
double computePostfix(const string);
vector<double> stringToVector(const string);
```

계산기의 중요한 기능을 구현하는 5가지 함수가 있다.

```
bool checkPARENTHESIS(const string infix)
```

```
for (int i = 0; i < infix.size(); i++) {
    if (infix[i] == LEFT_PARENTHESIS || infix[i] == LEFT_BRACE || infix[i] == LEFT_BRACKET) {
        s.push(infix[i]);
    }

    else if (infix[i] == RIGHT_PARENTHESIS) {
        if (s.empty())
            return false;
        if (s.top() == LEFT_PARENTHESIS)
            s.pop();
    }

    else if (infix[i] == RIGHT_BRACE) {
        if (s.empty())
            return false;
        if (s.top() == LEFT_BRACE)
            s.pop();
    }

    else if (infix[i] == RIGHT_BRACKET) {
        if (s.empty())
            return false;
        if (s.top() == LEFT_BRACKET)
            s.pop();
    }
}
```

```
if (s.empty())
    return true;
else
    return false;
```

먼저 checkPARENTHESIS 함수이다. 이 함수는 중위 표기식 문자열을 매개변수로 받아 괄호가 정상적으로 입력되면 true, 아니면 false를 반환하는 함수다. 반복문을 통해 문자열의 모든 인덱스를 점검하고 여는 괄호가 입력되면 stack에 push를 한다. 그리고 닫는 괄호가 입력될 때 stack이 비어 있다면 false를 반환, 아니면 stack에 가장 위에 있는 데이터가 해당 닫는 괄호의 여는 괄호일 경우 pop을 한다. 반복문이 끝나고도 stack에 데이터가 남아 있으면 false를 반환한다.

```
string inToPos(const string infix)
```

inToPos는 중위 표기식 문자열을 매개변수로 받아 후위 표기식 문자열로 변환하고 반환하는 함수이다.

```
while (infix[cnt] && infix[cnt] != '\0') {
    if (isdigit(infix[cnt]) || infix[cnt] == DECIMAL) {
        postfix.push_back(infix[cnt]);
        if (isdigit(infix[cnt + 1]) == 0 && infix[cnt + 1] != DECIMAL)
            postfix.push_back(' ');
        else if (infix[cnt + 1] == NULL)
            postfix.push_back(' ');
    }

    else if (infix[cnt] == LEFT_PARENTHESIS || infix[cnt] == LEFT_BRACE || infix[cnt] == LEFT_BRACKET) {
        s.push(infix[cnt]);
    }

    else if (infix[cnt] == '*') {
        if (s.empty()) {
            s.push('*');
        }
        else {
            if (getOrder(s.top(), infix[cnt])) {
                postfix.push_back(s.top());
                s.pop();
                postfix.push_back(' ');
            }
            s.push('*');
        }
    }
```

```
else if (infix[cnt] == RIGHT_PARENTHESIS) {
    while (s.top() != LEFT_PARENTHESIS) {
        postfix.push_back(s.top());
        s.pop();
        postfix.push_back(' ');
    }
    s.pop();
}
```

반복문을 통해 infix의 모든 문자열을 확인하고 숫자나 '.' 기호가 입력될 경우 후위 표기식 문자열에 입력한다. 만약 infix의 다음 문자가 숫자나 DECIMAL 기호가 아니거나 NULL일 경우 후위 표기식 문자열에 공백을 입력한다. 괄호가 입력되면 무조건 push한다. 연산자가 입력될 경우 stack이 비어 있으면 무조건 push하고 그렇지 않다면 getOrder() 함수를 호출하여 연산자의 우선순위를 확인하고 stack에 있는 연산자가 현재 연산자보다 같거나 높을 경우 후위 표기식 문자열에 입력하고 pop을 한다. 닫는 괄호가 입력되면 해당 닫는 괄호의 여는 괄호가 나올 때까지 후위 표기식 문자열에 입력하고 pop을 한다. 그리고 여는 괄호가 나오면 여는 괄호를 pop한다.

```
bool getOrder(char oldStack, char newStack) {
    if (oldStack == LEFT_PARENTHESIS || oldStack == LEFT_BRACE || oldStack == LEFT_BRACKET)
        return false;

    if (newStack == '+' || newStack == '-')
        return true;
    else {
        if (oldStack == '*' || oldStack == '/')
            return true;
        else
            return false;
    }
}
```

다음은 getOrder함수이다. getOrder 함수는 stack에 있는 연산자가 현재 연산자보다 같거나 높을 경우 true를 반환하고 낮을 경우 false를 반환한다. 또한 여는 괄호가 나오면 false를 반환한다.

```
double computePostfix(const string postfix) {
    string tmp;
    double result = 0.0;
    int cnt = 0;
    vector<double> operand = stringToVector(postfix);
    stack<double> numbers;
```

```
for (int i = 0; i < postfix.length(); i++) {
    if (strchr("+-/*", postfix[i]) != NULL)
    {
        double tmp;
        double num2 = numbers.top();
        numbers.pop();
        double num1 = numbers.top();
        numbers.pop();

        switch (postfix[i]) {
        case '+':
            tmp = num1 + num2;
            break;
        case '-':
            tmp = num1 - num2;
            break;
        case '*':
            tmp = num1 * num2;
            break;
        case '/':
            if (num2 == 0) {
                checkDividedByZero = true;
                return -1;
            }
            tmp = num1 / num2;
            break;
```

```
    }

    else if ((isdigit(postfix[i]) || postfix[i] == DECIMAL) && postfix[i + 1] == ' ') {
        numbers.push(operand[cnt++]);
    }
}
result = numbers.top();
numbers.pop();

return result;
```

computePostfix는 후위 표기식을 문자열을 매개 변수로 받고 후위 표기식을 연산한 후 결과를 반환하는 함수이다. stringToVector(문자열로 되어 있는 피연산자를 double형으로 바꾸는 함수)의 값을 저장하는 double형 vector와 double형 데이터를 저장하는 stack을 이용하여 구현했다.

반복문을 통해 후위 표기식의 모든 문자열을 탐색하고 해당 인덱스에 숫자가 입력되고 다음 인덱스의 공백 문자가 입력될 경우 vector에 있는 데이터를 stack에 push한다. (후위 표기식 문자열을 저장할 때 숫자나 연산자가 끝나는 인덱스의 다음 인덱스에 공백을 입력했기 때문이다.) 만약 연산자가 입력될 경우 stack에서 두 개의 문자를 pop하고 해당 연산자에 맞는 연산을 한 후 결과를 stack에 push 한다. /의 경우 0으로 나눌 경우 에러메시지를 출력하고 리턴 한다. 그리고 모든 연산이 끝나면 결과값을 리턴한다.

```
vector<double> stringToVector(const string postfix) {
    vector<double> operand;
    string tmp = "";

    for (int i = 0; i < postfix.length(); i++) {
        if (postfix[i] == ' ') {
            if (tmp.size() == 0);
            else {
                operand.push_back(stod(tmp));
                tmp.clear();
            }
        }
        else if (isdigit(postfix[i]) || postfix[i] == DECIMAL) {
            tmp.push_back(postfix[i]);
        }
    }

    return operand;
}
```

마지막으로 후위 표기식 문자열을 매개 변수로 받고 double형 vector를 반환하는 stringToVector 함수다. 이 함수는 반복문을 통해 후위 표기식의 모든 문자열을 탐색하고 숫자나 DECIMAL이 나올 경우 임시 문자열에 저장을 한다. 그리고 공백 문자가 나올 경우 임시 문자열에 저장한 데이터를 vector에 저장한다. 모든 문자열을 읽으면 double형 vector를 반환한다.

■ Conclude with some comments on my work

간단한 계산기 프로그램을 만들어보면서 STL에서 제공하는 stack에 대해서 더 잘 알 수 있었다. 그리고 stack이 유용한 자료구조라는 것을 이번 과제를 통해 알게 되었다. 또한 예전에 배웠던 string과 vector 자료구조를 사용해 볼 수 있었다.