

실습과제 5.2

[과제 설명]

계산기는 스택을 활용한 대표적인 사례이다. 프로그램을 만들 때나 계산기를 사용할 때 보통 사용자는 중위 표기법을 사용한다.

중위 표기법은 연산자(+, - 등)가 피연산자(숫자) 사이에 위치하는데, 연산 순서에 대한 정보가 포함되어 있지 않다. 그렇기 때문에 연산자 우선 순위에 따라서 컴퓨터는 계속 비교해서 처리해야 하기 때문에 리소스가 많이 필요하게 된다. 하지만, 전위/후위 표기법은 연산의 순서에 대한 정보가 포함되어 있어서 괄호가 필요하지 않고 단방향으로 연산을 하면 된다. 따라서 스택을 이용해서 먼저 주어진 중위식을 후위식으로 변환하고, 변환된 후위식을 계산하는 프로그램을 만든다.

이론상으로 중위식을 후위식을 변환하는 방법은 이렇다.

1. 연산 우선순위에 맞게 모든 연산에 괄호를 친다
2. 연산 우선순위가 높은 연산부터 괄호 안에 있는 연산자를 괄호 뒤로 뺀다.
3. 연산자가 앞으로 빠진 괄호는 삭제한다.
4. 괄호가 다 사라질 때까지 반복한다.

이것을 프로그램으로 구현할 때는 문자열로 수식을 받기 때문에 숫자와 연산자를 구별하는 함수, 연산자의 우선순위를 따로 판별해주는 함수를 만들고, 괄호에 따라서 연산자를 임시 저장할 때 스택을 사용한다.

또, 이 스택은 변환된 후위식을 계산할 때도 사용된다. 후위식을 계산할 때는 숫자인지 확인하고 숫자이면 스택에 push하기 전에 아스키코드에서 정수값으로 반드시 변환해주어야 한다(실제로 연산할 때는 정수값으로 해야하는데 아스키코드로 하면 '1'+2'가 되고 95가 되어서 논리 오류가 발생하기 때문이다). 숫자가 아니라면 연산자인데, 연산자인 경우 기존에 push된 숫자를 꺼내서 연산하고 다시 스택에 집어넣는다. 이것을 스택에 결과값만 남을 때까지 반복한다.

[코드]

메인 함수와 header 파일은 교안에서 미리 구현된 것을 사용했으므로 보고서에는 생략하고 구현된 함수 세부 내용만 첨부한다.

```
#include "MyArrayStack.hpp"

//중위식->후위식 변환
void Infix2Postfix(const char* infix_exp, char* postfix_exp) {
    ArrStack<char> stack;
    stack.Stack_Init(); //스택 초기화

    int len = strlen(infix_exp);

    int j = 0;

    for (int i = 0; i < len; i++) {
        char token = infix_exp[i];

        if (IsDigit(token)) { //token이 숫자라면 postfix에 저장한다.
            postfix_exp[j] = token;
            j++;
        }
        else if (IsOperator(token)) { //token이 연산자라면
            while (!stack.Stack_IsEmpty() && Priority(token) <= Priority(stack.Stack_Peek())) {
                postfix_exp[j] = stack.Stack_Pop();
                j++;
            }
            stack.Stack_Push(token);
            //스택에 저장되어 있는 연산자 우선순위가 같거나 작을 때까지 POP해서
            //postfix에 저장한 뒤, 마지막으로 token을 넣는다.
        }
        else if (token == '(') { //만약 여는 괄호라면
            stack.Stack_Push(token);
        }
        else if (token == ')') { //만약 닫는 괄호라면 여는 괄호가 나올 때까지 POP해서 postfix에 저장한다.
            while (stack.Stack_Peek() != '(') {
                postfix_exp[j] = stack.Stack_Pop();
                j++;
            }
            stack.Stack_Pop();
            //여는 괄호는 POP하기만 하고 postfix에 저장하지는 않는다.
        }
    }

    while (!stack.Stack_IsEmpty()) {
        postfix_exp[j] = stack.Stack_Pop();
        j++;
    } //스택이 빌 때까지 남아있는 연산자를 postfix에 넣는다.

    postfix_exp[j] = '\0';
    //문자열이기 때문에 끝에 NULL 문자를 넣지 않으면 오류난다.
}

//숫자인지 확인
bool IsDigit(char token) {
```

```

    if (token >= '0' && token <= '9')
        //if(token >= 48 && token <= 57)과 동일
        return true;
    else
        return false;
}

//연산자인지 확인
bool IsOperator(char token) {
    switch (token)
    {
        case '+':
        case '-':
        case '/':
        case '*':
            return true;
        default:
            return false;
    }
    //아스키코드가 연달아 있지 않기 때문에 케이스를 다 따로 적었다.
}

//우선순위 확인
int Priority(char op) {
    int pri;

    switch (op)
    {
        case '*':
        case '/':
            pri = 3;
            break;

        case '+':
        case '-':
            pri = 2;
            break;

        case '(':
            pri = 1;
            break;

        default:
            pri = -1;
            break;
    }

    return pri;
}

//후위식 계산 및 결과
int Eval_Postfix(char* postfix_exp) {
    ArrStack<int> stack;
    stack.Stack_Init();

    int len = strlen(postfix_exp);

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

```

```

char token = postfix_exp[i];

if (IsDigit(token)) { //일단 숫자면 문자에서 정수형으로 값을 변경 후 스택에 push한다.
    stack.Stack_Push(token - '0');
}
else if (IsOperator(token)) {
    int op2 = stack.Stack_Pop();
    int op1 = stack.Stack_Pop();
    //두 피연산자를 스택에서 역순으로 POP한다.
    int result = 0;

    switch (token)
    {
    case '+':
        result = op1 + op2;
        break;

    case '-':
        result = op1 - op2;
        break;

    case '*':
        result = op1 * op2;
        break;

    case '/':
        result = op1 / op2;
        break;

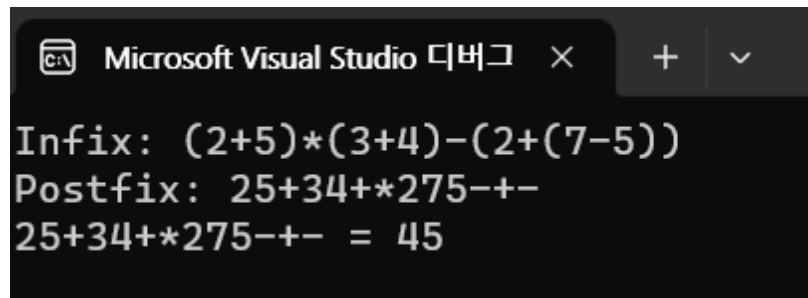
    default:
        break;
    }

    stack.Stack_Push(result);
    //연산을 하고 난 뒤에 다시 스택에 push한다.
}
}

return stack.Stack_Pop();
}

```

[실행 결과]



```
Microsoft Visual Studio 디버그 × + ▾  
Infix: (2+5)*(3+4)-(2+(7-5))  
Postfix: 25+34+*275-+-  
25+34+*275-+- = 45
```

[과제에 대한 고찰]

후위식은 중위식에 비해서 많이 생소한 개념이라 프로그램을 짜기 전에 손으로 직접 계산할 때도 처음에는 익숙하지 않아서 다소 버벅댔던 것 같다. 그래서 코드를 짜기 전에 의사 코드를 보고 알고리즘을 따라서 하나씩 천천히 계산해보았던 것 같다.

이 코드는 한 자리수에 대해서만 작동하는 단점이 있는데, 두 자리수 이상에 대해서도 기존의 알고리즘을 조금만 수정해도 가능할 것 같았다. (뒤에 오는 문자가 수라면 n자리수로 인식한다던지) 나중에 더 공부할 기회가 온다면 이렇게도 만들어보고 싶다.