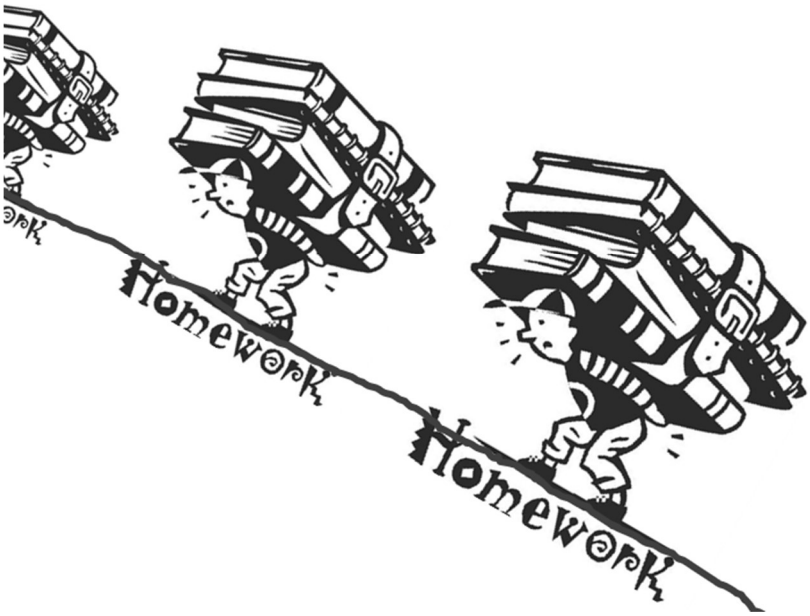
Data Structures 2020



Data Structures 2020

HW2

2016726028

이민재 | 자료구조 | 2020/11/09

과제 개요

- Write a C++ program that inputs numerical expressions line by line and then outputs their results. "EOI₩n" means the end of input.
  - Only 'a', 'b', 'c' can be input as operands.
  - Only 3 binary operators can be used as operators:
    - '@', '#', and '&'
    - '&' has higher precedence than '@' and '#'.
    - Consecutive operations with the same precedence must be computed from left.
  - The result is always 'a', 'b', or 'c'.
  - Operation table is given as a text file named "operations.txt"
  - When exceptions such as unbalanced parenthesis (or brace, bracket) occur, print "Error!₩n" and process next input lines.

컴파일 환경

Microsoft Visual Studio Enterprise 2019 (버전 16.7.7)

Microsoft Visual C ++ 2019

Amd ryzen Cpu(3700x)

Test in (x32) Debug mod

구현

Only 'a Operation table is given as a text file named "operations.txt"

'a', 'b', 'c' can be input as operands.

VOID OPEN_FILE

메인 함수에서 가장 먼저 호출되는 함수로, 실행파일과 동일한 디렉토리 상의 **"operations.txt"**를 읽어온다. 연산 정답 시트의 형식이 동일하다고 가정한 후, 프로그래밍 했기 때문에 **txt 의 결과 표 입력순은 (@-#-&)** 순이 되어야 한다. Open_file 함수는 파일 입출력과 getline 함수를 통해 텍스트 파일을 한줄씩 읽고, 해당하는 연산 결과값을 각각 **AtSign, HashTag, Ampersand 의 3X3 char 형배열에 저장한다.**

```
void open_file()
{
	ifstream fd;
	int count = 0;
	fd.open("operations.txt");

	if (fd.is_open())
	{
		while (!fd.eof())
		{
			char tmp1[256];
			char tmp[256];
			fd.getline(tmp1, 256);
			string str(tmp1); //char 배열을 문자열로
			str.erase(std::remove(str.begin(), str.end(), ' '),
str.end()); // 읽어온 라인 공백 제거
			//문자열을 char 배열로
			strcpy_s(tmp, str.c_str());
			switch (count)
			{
			case 1:
				AtSign[0][0] = tmp[0];
				AtSign[0][1] = tmp[1];
				AtSign[0][2] = tmp[2];
				break;


...
```

The result is always 'a', 'b', or 'c'.

VOID EVALUATE_STACK

Operand 간의 연산 결과를 push 하고 사용된 operations 를 pop 해주는 함수로, open_file 함수에서 저장한 연산 결과를 바탕으로 operations 와 각 operand 를 기준삼아 결과값(a/b/c)을 push 해준다.

```
switch (operations.top())
        {
        case '@':
                if (operand1 == 'a' && operand2 == 'a')
                        numbers.push(AtSign[0][0]);
                else if (operand1 == 'a' && operand2 == 'b')
                        numbers.push(AtSign[0][1]);

…

operations.pop();
```

위의 '@' case 의 경우 'a @ a' 연산에 대한 결과로 AtSign[0][0] 에 저장된 값을 numbers 에 push 하는 것을 알 수 있다.

Only 3 binary operators can be used as operators:

'@', '#', and '&'

'&' has higher precedence than '@' and '#'.

Consecutive operations with the same precedence must be computed from left.

When exceptions such as unbalanced parenthesis (or brace, bracket) occur, print "Error!₩n" and process next input lines.

CHAR READ_AND_EVALUATE(ISTREAM& INS)

강의 자료에 있는 read_and_evaluate 와 구조가 상당부분 동일하나, '@#&' 기호와 괄호 연산의 검사를 위해 다음과 같이 수정하였다.

먼저 사칙연산 대신 '@#&'의 기호를 새로운 연산자로 받아야 하고, 이중 & 연산자의 우선순위가 @#보다 높아야 하기 때문에, 각 연산자의 입력을 구분한다. **높은 우선순위의 연산자**일 경우, 동일 연산자가 stack 에 없다면 stack 에 쌓는다., **낮은 우선순위의 연산자**일 경우, 선행된 연산자와 우선순위가 같거나 낮기 때문에 앞선 계산을 먼저 진행한 뒤, stack 에 쌓는다. 이를 구현하면 다음과 같다.

```
else if (strchr("@#", ins.peek()) != NULL)
{
        if (operations.empty() == false && strchr("@#&", operations.top()) != NULL)
                        evaluate_stack(numbers, operations);
                ins >> symbol;
                operations.push(symbol);
        }
else if (strchr("&", ins.peek()) != NULL) {
                ins >> symbol;
        if (operations.empty() == false && strchr("&", operations.top()) != NULL)
                        evaluate_stack(numbers, operations);

                operations.push(symbol);

        }
```

다음으로, 강의자료와는 다르게 **소/중/대 괄호를 모두 사용**가능해야 하며, **각 괄호쌍 끼리의 묶음**이 정상적으로 되었는지를 검출하여 만약 틀리다면, **에러메시지**를 출력해야하기 때문에 이를 구현해 보았다.

왼쪽 괄호의 경우, 종류에 상관없이 무조건 push 되어야한다. 이와 동시에, 괄호쌍 묶음의 검사를 위하여 parenthesis stack 을 생성하여 추가적으로 push 해주었다.

```
else if (strchr("([{", ins.peek()) != NULL) { // 왼쪽 괄호 push
                ins >> symbol;
                operations.push(symbol);
                parenthesis.push(symbol);

        }
```

오른쪽 괄호의 경우, parenthesis 에 top 에 위치한 왼쪽 괄호의 알맞은 쌍이 입력되어야 하며, 그렇지 않다면 에러 메시지를 출력해야 한다. 알맞은 쌍이

```
else if (ins.peek() == RIGHT_PARENTHESIS || ins.peek() == RIGHT_CURLY_BRACKET||
ins.peek() == RIGHT_SQUARE_BRACKET)
        {
                ins >> symbol;
                switch (symbol)
                {
                case RIGHT_PARENTHESIS:
                        if (parenthesis.empty() == true)
                                throw - 1;
                        if (parenthesis.top() == '(')
                        {
                                parenthesis.pop();
                                while (operations.top() != '(')
                                        evaluate_stack(numbers, operations);
                                operations.pop();
                        }
                        else    throw - 1;
```

들어왔다면 operations stack 에 해당 괄호기호에 도달할 때까지 저장된 operations 와 numbers 를 사용하여 evaluate_stack 을 실행한다. 위 코드는 오른쪽 소괄호가 입력 되었을 때를 가정한 것으로, '(' 가 parenthesis 의 top 인지 먼저 확인하고, 일치한다면 stack 에 저장된 요소들을 이용하여 연산하고 pop 한다. **이때, 괄호 끼리 불일치 하는 경우, parenthesis 의 size 가 0 인경우 (왼쪽괄호 없음) 등을 상정하여 예외를 던지도록 설정**하였다.

"EOI₩n" means the end of input

```
else
                {
                        if (ins.peek() == 'E')
                        {
                                char tmp[3];
                                char eoi[] = "EOI";
                                ins >> tmp;
                                if (strcmp(tmp, eoi) == 0)
                                        exit(0);
                        }
                        ins.ignore();
                }

        }

        while (!operations.empty()) // 괄호 없을경우 계산
                evaluate_stack(numbers, operations);

        return numbers.top();
```

마지막으로 종료조건인 EOI 입력을 받기 위하여, ins.peek() 한 char 이 'E' 라면 전체 단어가 "EOI"인지를 검사하고 맞다면 프로그램을 종료한다. 최 하단의 while 반복문은 괄호가 없는 수식의 경우를 상정한 것으로써, 오른쪽 괄호를 읽음으로써 트리거 되는 계산이 없기 때문에, **어떠한 괄호도 수식에 없을 경우** 저장된 계산을 모두 수행하도록 설정하였다. 끝으로 연산결과가 남은 numbers 의 top 을 return 하여 메인 함수에 전달한다.

```
try
{
    cout << read_and_evaluate([&]cin) << endl;
    cin.ignore(); // '₩n' 무시
}
catch (int expn) // 예외에 따라 처리한다
{

    if (expn == -1) {
        cout << "Error!: unbalanced parenthesis" << endl;
    } // 올바르지 않은 괄호문일 때 예외 처리
```

메인 함수는 try-catch 문으로 설정하여 , 예외 처리를 하도록 구현하였다.

예시

C@B&C(괄호가 없는 연산)

b&c = c 의 연산이 우선되고 ,이후 c@c 의 연산으로 b 의 결과가 도출된다.

(c@b 우선의 경우 a&c 의 결과값인 a 가 출력될 것이다.)

```
c@b&c
b
```

(B@C#C)&A@C (괄호와 괄호 밖의 연산)

괄호 안의 b@c#c 가 먼저 연산되고 , 이후, b&a@c 의 연산으로 **결과값인 c 가 출력된다**

```
(b@c#c)&a@c
c
```

(A#C&(B@A#C)) (중첩된 한 종류 괄호의 연산)

(b@a#c) 의 연산이 먼저 이루어지고 이후 a#c&(c) 의 연산의 결과로써 **c 의 결과가 도출된다.**

```
(a#c&(b@a#c))
c
```

(A&B#(A@C@B))#[(C&B)@{C&A#C}@B] (중첩된 여러 종류 괄호의 연산)

슬라이드에 포함된 수식으로, **결과는 c 가 출력된다**

```
(a&b#(a@c@b))#[(c&b)@{c&a#c}@b]
c
```

{A@B)&C (짝이 맞지 않는 괄호의 연산)

{ 기호와 짝이되는 } 기호 대신에) 기호가 입력되었고, 따라서 **에러 메시지가 출력된다.**

```
{a@b)&c
Error!: unbalanced parenthesis
```

B@C#A)#{A&B} (왼쪽 괄호가 없는 연산)

Parenthesis 스택에 아무 괄호가 없는 상황에서 ')' 기호가 입력되었고, **에러메시지가 출력된다.**

```
b@c#a)#{a&b}
Error!: unbalanced parenthesis
```

B@A# (입력값이 부족한 연산)

#연산자 뒤에 올 operand 가 입력되지 않았고, **따라서 에러메시지가 출력된다.**

```
b@a#
Error!: missing operands
```

EOI (프로그램 종료)

EOI 를 입력하면 프로그램이 종료된다.

```
EOI

F:₩2020_DataStructure₩HW2₩Debug₩abc.exe(프로세스 15888개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

고찰

이번 과제는 stack 을 활용한 사칙연산 계산기에서 더 나아가, 사용자가 임의로 정한 연산자를, 그 결과값 시트를 활용하여 새로운 계산기를 만들어 보는 과제였다. 과제 수행을 위해 stack 의 pop, push 를 활용하며 원하는 조건 하에서 저장된 요소들을 사용, 또는 새로운 요소를 저장 하는 방법을 복습하였다. 또한, 파일 입출력과 예외처리 등을 복습 할 수 있었던 유용한 과제였다.