1. Running Environment

Window11, Visual Studio 2019에서 Debug x86으로 실행

2. Code Snippets

전체적인 알고리즘은 infix notation(중위표기식)을 입력 받은 후, postfix notation (후위표기식)으로 바꾸고, 이를 연산하여 결과를 출력하는 순서로 이루어져 있다.
먼저, infix를 postfix로 바꿔야 하고, 이 함수를 in2post라고 하였다.

```
12  int priority(char c)
13  // precondition: input char must be an operator(+-*/)
14  // postcondition: returns priority of specific operator
15  {
16      switch (c)
17      {
18      case '(':
19      case '{':
20      case '[':
21          return 0;
22      case '+':
23      case '-':
24          return 1;
25      case '*':
26      case '/':
27          return 2;
28      default:
29          break;
30      }
31  }
```

함수를 보기에 앞서, 함수에서 사용될 부호별 우선순위를 switch-case를 통하여 따로 지정해주었다.

```
45  string in2post(string infix)
46  // precondition: needs a infix notation
47  // postcondition: returns postfix notation
48  {
49      string postfix;
50      stack<char> operations;
51
52      for (int i = 0; i < infix.length(); i++) {
53          // left parentheses
54          if (infix[i] == '(' || infix[i] == '{' || infix[i] == '[') {
55              operations.push(infix[i]);
56          }
57          // number or decimal point(operand)
58          else if ((infix[i] >= '0' && infix[i] <= '9') || infix[i] == '.') {
59              postfix += infix[i];
60          }
```

in2post에서는 변환할 문자열 infix를 입력 받고, 문자열 원소를 하나씩 살펴보며 postfix로

변환할 것이다.

변환한 후위표기식을 저장할 문자열을 postfix, 부호를 담을 stack<char>를 operations라고 선언하였다. in2post 함수는 크게 4가지 경우로 나뉘어진다. 첫 번째는 왼쪽 괄호가 입력되었을 때(괄호끼리의 우선순위는 존재하지 않는다.)이고, 이 경우 스택에 괄호를 저장한다. 숫자가 들어올 경우에는 출력인 postfix에 추가한다.

```
// operator(+-*/)
else if (strchr("+-*/", infix[i]) != NULL) {
    postfix += ' ';
    if (!operations.empty()) {
        if (operations.top() != '(' && operations.top() != '{' && operations.top() != '[') {
            if (priority(operations.top()) >= priority(infix[i])) {
                while (operations.top() != '(' && operations.top() != '{' && operations.top() != '[') {
                    postfix += operations.top();
                    postfix += ' ';
                    operations.pop();
                    if (operations.empty()) break;
                }
                operations.push(infix[i]);
            }
            else if (priority(operations.top()) < priority(infix[i])) {
                operations.push(infix[i]);
            }
        }
        else operations.push(infix[i]);
    }
    else operations.push(infix[i]);
}
```

3번째는 부호가 들어올 경우인데, strchr를 통하여 현재 문자열 인덱스가 부호인지 파악하고, 공백을 추가한다. postfix를 후에 연산할 때, 숫자끼리 붙어있으면 구분할 수가 없으므로, 구분자인 공백(‘ ’)을 추가해준다.

stack이 비어있지 않을 경우, operations.top()이 좌측 괄호 중 하나일 경우, 현재 인덱스의 연산자와 operations.top() 연산자의 우선순위를 비교하여 그에 따라 처리하고, operations.top()이 좌측 괄호일 경우, 현재 인덱스의 부호를 stack에 push한다. stack이 비어있을 경우에는 빈 stack에다 그냥 현재 부호를 push한다.

```
// right parentheses
else if (infix[i] == ')' || infix[i] == '}' || infix[i] == ']') {
    while (true) {
        // Cases for unbalanced parentheses
        if (infix[i] == ')') {
            if (operations.top() == '{' || operations.top() == ']') {
                return "err";
            }
            else if (operations.top() == '(') break;
            else {
                postfix += ' ';
                postfix += operations.top();
                operations.pop();
            }
        }
        else if (infix[i] == '}') {
            if (operations.top() == '(' || operations.top() == ']') {
                return "err";
            }
            else if (operations.top() == '{') break;
            else {
                postfix += ' ';
                postfix += operations.top();
                operations.pop();
            }
        }
        else if (infix[i] == ']') {
            if (operations.top() == '(' || operations.top() == '{') {
                return "err";
            }
            else if (operations.top() == '[') break;
            else {
                postfix += ' ';
                postfix += operations.top();
                operations.pop();
            }
        }
    }
    operations.pop();
}
```

다음으로, infix의 현재 값이 오른쪽 괄호일 경우, unbalanced parentheses 문제를 고려해 주어야 한다. 현재 괄호의 쌍이 맞지 않다면 “err”문자열을 반환하고, 같은 쌍의 괄호가 나올 때까지는 공백 추가 및 operations.top()을 postfix에 추가 및 pop()한다. 같은 쌍의 괄호가 나오면, while문을 탈출하고, 그 같은 쌍의 괄호를 stack에서 pop()한다.

```
121                 operations.pop();
122             }
123         }
124         // if there is something still left in stack, add to postfix and pop
125         if (!operations.empty()) {
126             postfix += ' ';
127             postfix += operations.top();
128             operations.pop();
129         }
130         return postfix;
131     }
```

위에서 문자열을 훑어보는 for문을 탈출한 후에 만약 stack에 뭐라도 남아있다면, postfix에 공백 추가 및 operations.top()을 추가, pop()을 하고, postfix를 반환한다.

다음으로, 반환받은 postfix를 가지고 그 식을 계산하는 post_eval함수에 대해 설명한다.

```
33  double evaluate(char oper, double opr1, double opr2)
34  // precondition: opr1, opr2 should be positive real number, oper must be an operator
35  // postcondition: returns the result of operation using parameters
36  {
37      switch (oper) {
38      case '+': return (opr1 + opr2);
39      case '-': return (opr1 - opr2);
40      case '*': return (opr1 * opr2);
41      case '/': return (opr1 / opr2);
42      }
43  }
```

그 전에 post_eval에서 쓸 evaluate 함수를 만든다. 2개의 피연산자와 1개의 연산자를 입력받고, 이를 연산하는 간단한 함수이다.

```
133 double post_eval(string postfix)
134 // precondition: before starting this method, you need a postfix notation of your input.
135 // postcondition: After this method, you'll get the evaluate of the postfix.
136 {
137     double answer;
138     string temp;
139     stack<double> operands;
140     for (int i = 0; i < postfix.length(); i++) {
141         if (postfix[i] != ' ') {
142             // if the current element is operand
143             if ((postfix[i] >= '0' && postfix[i] <= '9') || postfix[i] == '.') {
144                 temp += postfix[i];
145             }
146             // if the current element is operator
147             else if (strchr("+-*/", postfix[i]) != NULL) {
148                 double opr2 = operands.top();
149                 operands.pop();
150                 double opr1 = operands.top();
151                 operands.pop();
152                 operands.push(evaluate(postfix[i], opr1, opr2));
153             }
154         }
155         else if (postfix[i] == ' ') {
156             if (temp != "") { // if not an empty string
157                 operands.push(stod(temp));
158             }
159             temp = ""; // reinitialize temp to empty string
160         }
161     }
162     if (!operands.empty()) {
163         answer = operands.top();
164         operands.pop();
165         return answer;
166     }
167     exit(0);
168 }
```

post_eval 함수에서는 postfix 문자열을 입력으로 받고, 이 문자열을 하나씩 보면서 진행한다. 피연산자를 담을 stack<double> operands를 선언해주었고, temp에는 현재 숫자를 저장할 것이다.

위 in2post 함수에서는 숫자와 부호 사이에 각각 공백을 넣었는데, post_eval에서는 이를 기준으로 연산을 한다. 만약 현재 문자가 공백이 아니라면, 현재 원소가 숫자일 경우 공백이 나올 때까지 계속 temp에 추가한 후, 공백을 마주했을 때 그 값을 double로 바꿔서(stod) operands에 push해주게 된다. 만약 공백이 아닐 때 현재 문자가 연산자라면, stack의 위에서부터 2개의 연산자를 뽑아서 앞에서 선언한 evaluate 함수를 통해 연산을 진행 후 다시 stack에 집어넣는다.

```
170  void Calculator()
171  // postcondition: After completing this method, you will get teh answer of your infix input.
172  //                Works as a Calculator.
173  {
174      string infix;   // input of calculator
175      double answer;
176
177      while (true) {
178          cin >> infix;
179          if (infix == "EOI") break; // Break when EOI
180          string postfix = in2post(infix);
181          if (postfix == "err") cout << "Error!: unbalanced parentheses" << endl;
182          else {
183              cout << postfix << endl;
184              answer = post_eval(postfix);
185              if (!isfinite(answer)) cout << "Error!: dvide by zero" << endl;
186              else cout << round(answer * 1000) / 1000 << endl;   // round off to the nearest thousandth
187          }
188          cout << endl;
189      }
190  }
```

위에서 만든 두 함수 in2post와 post_eval을 합쳐서 최종 함수인 Calculator()를 만들게 되고, 이때 EOI가 입력되면 break;로 while문을 탈출하여 프로그램이 종료되고, 그렇지 않다면 in2post를 진행하게 된다. 이때, “err”이 출력된다면 unbalanced parentheses에 대한 경고가 출력되며 그 외의 경우에는 post_eval 실행 및 그 결과를 소수점 넷째 자리에서 반올림한 값을 출력하게 된다.

```
192  int main() {
193      Calculator();
194      return 0;
195  }
```

마지막으로 main문에서는 Calculator함수를 실행해주기만 한다.

실행 결과는 다음과 같다.

HW2 ppt자료에 나온 값들을 출력해 보았고, 마지막에는 그냥 임의의 값을 넣어서 그 값을 계산해보았다. 알아보기 쉽게하기 위해, 변환된 postfix 수식도 출력해주었다. EOI가 입력되면 프로그램이 종료되는 것을 확인할 수 있다.

3. Discussion(고찰)

이번 과제에서는 괄호가 섞인 사칙연산 계산기를 만들어 보았다. 수치식에서 나올 수 있는 경우의 수들을 직접 생각하면서 코드를 작성하느라 시간이 조금 오래 걸리긴 하였으나 수식이 길고 복잡하더라도 잘 작동하는 것을 확인할 수 있었다.

C/C++을 처음 배울 때 만들어보는 괄호가 없는 사칙연산 계산기는 switch-case 문 또는 다른 것들을 활용해서 간단하게 구현이 가능하였다. 하지만 괄호치기의 처리 및 후위 표기식으로 바꾸는 과정에 있어서 스택이 필요하였고, 또 이 후위 표기식을 계산하는 과정에 있어서도 스택을 사용하였다.

코드를 작성하고, 여러 오류에 대해 많은 디버깅을 해본 결과 스택을 사용하면 오버플로우 혹은 언더플로우로 인해 에러가 발생하는 횟수가 잦아서 상당히 귀찮음을 느꼈다. 또한 .top()을 통해서 스택의 최상위 원소에 접근할 수는 있었지만, 개별 원소를 인덱스로 접근할 수 없어서 연산 도중 이전의 부호나 괄호 정보를 알아볼 수 없다는 점이 굉장히 큰 단점으로 다가왔다. 이 때문에 제대로 된 디버깅이 불가능하다는 것도 매우 큰 단점이었다. 결국에 스택은 가독성은 좋지만, 디버깅이 불가능하다는 큰 단점이 있으므로, 계산기 뿐만 아니라 다른 문제를 푸는 데에 있어서도 이를 사용하는 것이 크게 좋지는 않다고 생각하였다.

이를 대체할 방법으로, 스택과 동일하게 사용이 가능하며 개별원소를 접근할 수 있는 벡터를 사용하는 것이 좋은 대안이 될수 있을 것이라고 생각하였다.