**[자료구조1분반] 숙제1. 스택**

**성공코드**

A. 괄호 짝이 맞는지 검사(C++)

|  |
| --- |
| 괄호 짝이 맞는지 검사 입력 되는 문자열에 대한 아래 요건을 만족하는 괄호 검사 프로그램을 만드시오.  - 소괄호“()”, 중괄호“{}”, 대괄호“[]” 의 짝이 맞는지 확인   INPUT ”EOF“를 입력 받을 때까지 문자열을 입력 OUTPUT 괄호 검사 성공 여부에 따라 출력문은 아래와 같고 오류 시 오류가 생긴 문장의 라인,문자까지만 인정 1) 성공 : OK, Line\_count : **(문장 수)**, bracket\_count : **(괄호 짝 수)** 2) 오류 : Error, Line\_count : **(문장 수)**, bracket\_count : **(괄호 짝 수)**  - 아래 main 함수를 참조하여 **checkMatching()** 함수를 만드시오. int main(){     std::string Str,temp;     while(true){         std::getline(std::cin,temp);         if(temp=="EOF")break;         Str.append(temp);         Str.append("\n");         std::cin.clear();     }     checkMatching(Str);      return 0; } Sample Input **[ { } [** **] ( )** **] EOF** Sample Output OK, Line\_count : 3, bracket\_count : 4   Sample Input **{ } [** **] ( )** **EOF** Sample Output   OK, Line\_count : 2, bracket\_count : 3   Sample Input **{** **EOF** Sample Output Error, Line\_count : 1, bracket\_count : 0 |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <cstdio>  #include <stdio.h>  #include <cstdlib>  #include <string>  #include <iostream>  #include <cstring>  #include <fstream>  **using** **namespace** std;  **const** **int** MAX\_STACK\_SIZE = 20;      //오류 처리 함수  **inline** **void** error(**char**\* message)  {  **printf**("%s\n", message);  **exit**(1);  }    **class** ArrayStack  {  **int** top;  **char** data[MAX\_STACK\_SIZE] = { };    **public**:      ArrayStack() { top = -1; } //스택 생성자: create()      ~ArrayStack() {} //스택 소멸자    **bool** isEmpty() { **return** top == -1; }    **bool** isFull() { **return** top == MAX\_STACK\_SIZE - 1; }    **void** push(**int** e)      {  **if** (isFull()) { error("스택 포화 에러"); }          data[++top] = e;      }    **int** pop()      {  **if** (isEmpty()) { error("스택 공백 에러"); }  **return** data[top--];      }    **int** peek() //삭제하지 않고 요소 반환      {  **if** (isEmpty()) { error("스택 공백 에러"); }  **return** data[top];      }    **void** display()      {  **printf**("[스택 항목의 수 = %2d] ==> ", top + 1);  **for** (**int** i = 0; i <= top; i++)          {  **printf**("<%2d>", data[i]);          }  **printf**("\n");      }  };      **bool** checkMatching(**const** **char**\* filename)  {  **int** fp = **strlen**(filename);    **if** (fp == 0)      {          error("Error: 파일 존재하지 않습니다.\n");      }    **int** nLine = 0;  **int** nChar = 0;        ArrayStack stack;  **char** ch;  **for** (**int** i = 0; i < fp; i++)      {          ch = filename[i];    **if** (ch == '\n')              {                  nLine++;              }      **if** (ch == '[' || ch == '(' || ch == '{')              {                  stack.push(ch);              }  **else** **if** (ch == ']' || ch == ')' || ch == '}')              {  **int** prev = stack.pop();  **if** ((ch == ']' && prev != '[') || (ch == ')' && prev != '(') || (ch == '}' && prev != '{'))                  {    **break**;                  }                  nChar++;              }      }      **if** (!stack.isEmpty())      {  **printf**("Error, Line\_count : %d, bracket\_count : %d\n\n", nLine, nChar);      }  **else**      {  **printf**("OK, Line\_count : %d, bracket\_count : %d\n\n", nLine, nChar);      }  **return** stack.isEmpty();  }    **int** main() {      std::string Str, temp;  **while** (**true**) {          std::getline(std::cin, temp);  **if** (temp == "EOF")**break**;          Str.append(temp);          Str.append("\n");          std::cin.clear();      }      checkMatching(Str.data());  **return** 0;  } |

B. 작은 따옴표 처리(C++)

|  |
| --- |
| 작은 따옴표 처리 입력 되는 문자열에 대한 아래 요건을 만족하는 프로그램을 만드시오.  - 작은 따옴표 사이에 대한 문자열 무시  - 작은 따옴표 처리 중 이스케이프 시퀀스 문자('\n','\\','\t' 등)에 대해서 정상적인 처리   INPUT ”EOF“를 입력 받을 때까지 문자열을 입력 OUTPUT 프로그램 검사 성공 여부에 따라 출력문은 아래와 같고 오류 시 오류가 생긴 문장의 라인,문자까지만 인정 1) 성공 : OK, Line\_count : **(문장 수)**,  quotes\_count : **(작은 따옴표 짝 수)** 2) 오류 : Error, Line\_count : **(문장 수)**, quotes\_count : **(작은 따옴표 짝 수)**  - 아래 main 함수를 참조하여 **checkMatching()** 함수를 만드시오. int main(){     std::string Str,temp;     while(true){         std::getline(std::cin,temp);         if(temp=="EOF")break;         Str.append(temp);         Str.append("\n");         std::cin.clear();     }     checkMatching(Str);      return 0; } Sample Input ' { { ' **EOF** Sample Output **OK, Line\_count : 1, quotes\_count : 1** Sample Input printf('left quotes\_count\n'); printf('right quotes\_count'); **EOF** Sample Output **OK, Line\_count : 2, quotes\_count : 2**   Sample Input printf('escape\_count'\n'); **EOF** Sample Output **Error, Line\_count : 1, quotes\_count : 1** |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <cstdio>  #include <stdio.h>  #include <cstdlib>  #include <string>  #include <iostream>  #include <cstring>  #include <fstream>  **using** **namespace** std;  **const** **int** MAX\_STACK\_SIZE = 20;      //오류 처리 함수  **inline** **void** error(**char**\* message)  {  **printf**("%s\n", message);  **exit**(1);  }    **class** ArrayStack  {  **int** top;  **char** data[MAX\_STACK\_SIZE] = { };    **public**:      ArrayStack() { top = -1; } //스택 생성자: create()      ~ArrayStack() {} //스택 소멸자    **bool** isEmpty() { **return** top == -1; }    **bool** isFull() { **return** top == MAX\_STACK\_SIZE - 1; }    **void** push(**int** e)      {  **if** (isFull()) { error("스택 포화 에러"); }          data[++top] = e;      }    **int** pop()      {  **if** (isEmpty()) { error("스택 공백 에러"); }  **return** data[top--];      }    **int** peek() //삭제하지 않고 요소 반환      {  **if** (isEmpty()) { error("스택 공백 에러"); }  **return** data[top];      }    **void** display()      {  **printf**("[스택 항목의 수 = %2d] ==> ", top + 1);  **for** (**int** i = 0; i <= top; i++)          {  **printf**("<%2d>", data[i]);          }  **printf**("\n");      }  };      **bool** checkMatching(**const** **char**\* filename)  {  **int** fp = **strlen**(filename);    **if** (fp == 0)      {          error("Error: 파일 존재하지 않습니다.\n");      }    **int** nLine = 0;  **int** nChar = 0;        ArrayStack stack;  **char** ch;  **bool** s\_q = **false**; //작은따옴표      //bool b\_q = false; //큰따옴표    **for** (**int** i = 0; i < fp; i++)      {          ch = filename[i];    **if** (ch == '\n')          {              nLine++;          }    **if** (ch == '\'' && s\_q == **false**)          {              s\_q = **true**;              stack.push(ch);          }  **else** **if** (ch == '\'')          {              s\_q = **false**;              stack.pop();  **if** (ch != '\'')              {  **break**;              }              nChar++;          }      }      **if** (!stack.isEmpty())      {  **printf**("Error, Line\_count : %d, quotes\_count : %d\n\n", nLine, nChar);      }  **else**      {  **printf**("OK, Line\_count : %d, quotes\_count : %d\n\n", nLine, nChar);      }  **return** stack.isEmpty();  }    **int** main() {      std::string Str, temp;  **while** (**true**) {          std::getline(std::cin, temp);  **if** (temp == "EOF")**break**;          Str.append(temp);          Str.append("\n");          std::cin.clear();      }      checkMatching(Str.data());  **return** 0;  } |

C. 큰 따옴표 처리(C++)

|  |
| --- |
| 큰 따옴표 처리 입력 되는 문자열에 대한 아래 요건을 만족하는 괄호 검사 프로그램을 만드시오.  - 큰 따옴표 사이에 대한 문자열 무시  - 괄호 닫기에 대해 문제가 없어야 함 INPUT ”EOF“를 입력 받을 때까지 문자열을 입력 OUTPUT 괄호 검사 성공 여부에 따라 출력문은 아래와 같고 오류 시 오류가 생긴 문장의 라인,문자까지만 인정 1) 성공 : OK, Line\_count : **(문장 수)**, bracket\_count : **(괄호 짝 수)** 2) 오류 : Error, Line\_count : **(문장 수)**, bracket\_count : **(괄호 짝 수)**  - 아래 main 함수를 참조하여 **checkMatching()** 함수를 만드시오. int main(){     std::string Str,temp;     while(true){         std::getline(std::cin,temp);         if(temp=="EOF")break;         Str.append(temp);         Str.append("\n");         std::cin.clear();     }     checkMatching(Str);      return 0; }   Sample Input **printf(" left bracket = (((((( "); printf(" right bracket = }}}}}} ");** **EOF** Sample Output   OK, Line\_count : 2, bracket\_count : 2 Sample Input **printf(" left bracket = (((((( ); printf(" right bracket = }}}}}} ");** **EOF** Sample Output Error, Line\_count : 2, bracket\_count : 0 |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <cstdio>  #include <stdio.h>  #include <cstdlib>  #include <string>  #include <iostream>  #include <cstring>  #include <fstream>  **using** **namespace** std;  **const** **int** MAX\_STACK\_SIZE = 20;      //오류 처리 함수  **inline** **void** error(**char**\* message)  {  **printf**("%s\n", message);  **exit**(1);  }    **class** ArrayStack  {  **int** top;  **char** data[MAX\_STACK\_SIZE] = { };    **public**:      ArrayStack() { top = -1; } //스택 생성자: create()      ~ArrayStack() {} //스택 소멸자    **bool** isEmpty() { **return** top == -1; }    **bool** isFull() { **return** top == MAX\_STACK\_SIZE - 1; }    **void** push(**int** e)      {  **if** (isFull()) { error("스택 포화 에러"); }          data[++top] = e;      }    **int** pop()      {  **if** (isEmpty()) { error("스택 공백 에러"); }  **return** data[top--];      }    **int** peek() //삭제하지 않고 요소 반환      {  **if** (isEmpty()) { error("스택 공백 에러"); }  **return** data[top];      }    **void** display()      {  **printf**("[스택 항목의 수 = %2d] ==> ", top + 1);  **for** (**int** i = 0; i <= top; i++)          {  **printf**("<%2d>", data[i]);          }  **printf**("\n");      }  };      **bool** checkMatching(**const** **char**\* filename)  {  **int** fp = **strlen**(filename);    **if** (fp == 0)      {          error("Error: 파일 존재하지 않습니다.\n");      }    **int** nLine = 0;  **int** nChar = 0;  **bool** b\_q = **false**;        ArrayStack stack;  **char** ch;      **for** (**int** i = 0; i < fp; i++)      {          ch = filename[i];    **if** (ch == '\n')          {  **if** (b\_q == **true**)              {                  nLine++;              }              nLine++;          }    **if** (ch == '\"' && b\_q == **false**)          {              b\_q = **true**;          }  **else** **if** (ch == '\"')          {              b\_q = **false**;          }    **if** (ch == '[' || ch == '(' || ch == '{')          {  **if** (b\_q == **false**)              {                  stack.push(ch);              }          }  **else** **if** (ch == ']' || ch == ')' || ch == '}')          {  **if** (b\_q == **false**)              {  **int** prev = stack.pop();  **if** ((ch == ']' && prev != '[') || (ch == ')' && prev != '(') || (ch == '}' && prev != '{'))                  {                      stack.push(ch);  **break**;                  }                  nChar++;              }          }      }      **if** (!stack.isEmpty())      {  **printf**("Error, Line\_count : %d, bracket\_count : %d\n\n", nLine, nChar);      }  **else**      {  **printf**("OK, Line\_count : %d, bracket\_count : %d\n\n", nLine, nChar);      }  **return** stack.isEmpty();  }    **int** main() {      std::string Str, temp;  **while** (**true**) {          std::getline(std::cin, temp);  **if** (temp == "EOF")**break**;          Str.append(temp);          Str.append("\n");          std::cin.clear();      }      checkMatching(Str.data());  **return** 0;  } |

D. 주석 처리에서 괄호검사 안함(C++)

|  |
| --- |
| 주석 처리에서 괄호 검사 안함 입력 되는 문자열에 대한 아래 요건을 만족하는 주석 검사 프로그램을 만드시오.  - 주석(한 줄 주석”//“, 여러줄 주석”/\*\*/“)에 대한 괄호 무시 처리   INPUT ”EOF“를 입력 받을 때까지 문자열을 입력 OUTPUT 괄호 검사 성공 여부에 따라 출력문은 아래와 같고 오류 시 오류가 생긴 문장의 라인,문자까지만 인정 1) 성공 : OK, Line\_count : **(문장 수)**, bracket\_count : **(괄호 짝 수)** 2) 오류 : Error, Line\_count : **(문장 수)**, bracket\_count : **(괄호 짝 수)**  - 아래 main 함수를 참조하여 **checkMatching()** 함수를 만드시오. int main(){     std::string Str,temp;     while(true){         std::getline(std::cin,temp);         if(temp=="EOF")break;         Str.append(temp);         Str.append("\n");         std::cin.clear();     }     checkMatching(Str);      return 0; } Sample Input printf(" left bracket = (((((( ");    // remark check printf(" right bracket = }}}}} "); **EOF** Sample Output OK, Line\_count : 2, bracket\_count : 2 Sample Input printf(" right bracket = }}}}} ");   /\* remark check  \*/ **EOF** Sample Output   OK, Line\_count : 2, bracket\_count : 1   Sample Input printf(" right bracket = }}}}} //");  printf(" right bracket = }}}}} "//);  **EOF** Sample Output Error, Line\_count : 2, bracket\_count : 1 |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <cstdio>  #include <stdio.h>  #include <cstdlib>  #include <string>  #include <iostream>  #include <cstring>  #include <fstream>  **using** **namespace** std;  **const** **int** MAX\_STACK\_SIZE = 20;      //오류 처리 함수  **inline** **void** error(**char**\* message)  {  **printf**("%s\n", message);  **exit**(1);  }    **class** ArrayStack  {  **int** top;  **char** data[MAX\_STACK\_SIZE] = { };    **public**:      ArrayStack() { top = -1; } //스택 생성자: create()      ~ArrayStack() {} //스택 소멸자    **bool** isEmpty() { **return** top == -1; }    **bool** isFull() { **return** top == MAX\_STACK\_SIZE - 1; }    **void** push(**int** e)      {  **if** (isFull()) { error("스택 포화 에러"); }          data[++top] = e;      }    **int** pop()      {  **if** (isEmpty()) { error("스택 공백 에러"); }  **return** data[top--];      }    **int** peek() //삭제하지 않고 요소 반환      {  **if** (isEmpty()) { error("스택 공백 에러"); }  **return** data[top];      }    **void** display()      {  **printf**("[스택 항목의 수 = %2d] ==> ", top + 1);  **for** (**int** i = 0; i <= top; i++)          {  **printf**("<%2d>", data[i]);          }  **printf**("\n");      }  };      **bool** checkMatching(**const** **char**\* filename)  {  **int** fp = **strlen**(filename);    **if** (fp == 0)      {          error("Error: 파일 존재하지 않습니다.\n");      }    **int** nLine = 0;  **int** nChar = 0;          ArrayStack stack;  **char** ch;  **bool** b\_q = **false**;  **bool** com\_pre = **false**; //주석 준비  **bool** com = **false**; //한줄주석  **bool** com\_star = **false**; //\* 주석  **bool** ex\_com\_star\_pre = **false**; //\* 주석 해제준비    **for** (**int** i = 0; i < fp; i++)      {          ch = filename[i];            //주석 체크  **if** (com\_pre == **false** && ch == '/' && b\_q == **false**)          {              com\_pre = **true**;          }  **if** (com\_star == **true** && ch == '\*')          {              ex\_com\_star\_pre = **true**;          }            //주석인지 일반 문자인지 확인  **if** (com\_pre == **true** && ch == '/')          {              com = **true**;          }  **else** **if**(com\_pre == **true** && ch == '\*')          {              com\_star = **true**;          }  **else**          {              com\_pre = **false**;          }            //\*별 주석 해제인지 일반 문자였는지 확인  **if** (ex\_com\_star\_pre == **true** && ch == '/')          {              com\_star = **false**;          }  **else**          {              ex\_com\_star\_pre = **false**;          }      **if** (ch == '\n')          {  **if** (b\_q == **true**)              {                  nLine++;              }              nLine++;              com = **false**;          }    **if** (ch == '\"' && b\_q == **false**)          {              b\_q = **true**;          }  **else** **if** (ch == '\"')          {              b\_q = **false**;          }    **if** (ch == '[' || ch == '(' || ch == '{')          {  **if** (b\_q == **false** && com == **false** && com\_star == **false**)              {                  stack.push(ch);              }          }  **else** **if** (ch == ']' || ch == ')' || ch == '}')          {  **if** (b\_q == **false** && com == **false** && com\_star == **false**)              {  **char** prev = stack.pop();  **if** ((ch == ']' && prev != '[') || (ch == ')' && prev != '(') || (ch == '}' && prev != '{'))                  {                      stack.push(ch);  **break**;                  }                  nChar++;              }          }      }      **if** (!stack.isEmpty())      {  **printf**("Error, Line\_count : %d, bracket\_count : %d\n\n", nLine, nChar);      }  **else**      {  **printf**("OK, Line\_count : %d, bracket\_count : %d\n\n", nLine, nChar);      }  **return** stack.isEmpty();  }    **int** main() {      std::string Str, temp;  **while** (**true**) {          std::getline(std::cin, temp);  **if** (temp == "EOF")**break**;          Str.append(temp);          Str.append("\n");          std::cin.clear();      }      checkMatching(Str.data());  **return** 0;  } |

E. 스택 – 수식 계산 프로그램(C)

|  |
| --- |
| **수식 계산 프로그램**  중위 수식을 입력받아 후위 수식 표기로 변환하여 계산 결과를 출력하는 프로그램을 작성하시오. 입력 수식은 실수이고 연산자는 +,-,\*,/ 이다. 수식에는 괄호도 포함된다. 입력 수식에 오류가 없다고 가정한다. 단, 예외는 0으로 나누는 경우만 고려하고 그 때 오류는 "Error : zero division error"를 출력하시오.  INPUT  띄어쓰기로 구분된 수식을 입력  OUTPUT  결과는 실수형(double)으로 소수 둘째자리로 출력    **Sample Input**  ( 6 / 3 ) + 4 \* 0.4  **Sample Output**  3.60  **Sample Input**  6 / 3 + 4 \* 0.4 / 0  **Sample Output**  Error : zero division error  **Sample Input**  13 / 2 + 4 \* 4 – 2 / 3  **Sample Output**  21.83 |
| #include <stdio.h>  #include <ctype.h>  #include <string.h>  #include <stdlib.h>    #define MAX\_SIZE 100    **double** stack[MAX\_SIZE];  **int** top = -1;    **void** push(**double** x) {  **if** (top == MAX\_SIZE - 1) {  **printf**("Stack overflow\n");  **return**;      }      stack[++top] = x;  }    **double** pop() {  **if** (top == -1) {  **printf**("Stack underflow\n");  **return** -1;      }  **return** stack[top--];  }    **int** is\_empty() {  **return** top == -1;  }    **int** peek() {  **if** (!is\_empty()) {  **return** stack[top];      }  **else** {  **return** -1;      }  }    **int** precedence(**char** op) {  **if** (op == '+' || op == '-') {  **return** 1;      }  **else** **if** (op == '\*' || op == '/') {  **return** 2;      }  **else** **if** (op == '^') {  **return** 3;      }  **else** {  **return** 0;      }  }    **void** check\_error(**char** postfix[])  {  **int** len = **strlen**(postfix);        // error:  연산자 후 0이 오면 예외처리  **for** (**int** i = 0; i < len; i++)      {  **if** (**strstr**(postfix, "/0") != NULL || **strstr**(postfix, "/ 0") != NULL)          {  **printf**("Error : zero division error\n");  **exit**(1);          }      }  }    **void** infix\_to\_postfix(**char**\* infix, **char**\* postfix) {        // 입력받은 표기식 에러 검사      check\_error(infix);    **int** i, j = 0;    **for** (i = 0; i < **strlen**(infix); i++) {    **if** (infix[i] == ' ')          {  **continue**;          }            // If the scanned character is an operand          // add it to the postfix expression  **if** (**isdigit**(infix[i]) || **isalpha**(infix[i])) {    **while** (**isdigit**(infix[i]) || infix[i] == '.') {                  postfix[j++] = infix[i++];              }  **if** (infix[i + 1] == NULL)              {                  infix[i++] = ' ';              }              postfix[j++] = ' ';              i--;                // If the scanned character is an opening parenthesis,              // push it to the stack          }  **else** **if** (infix[i] == '(') {                push(infix[i]);                // If the scanned character is a closing parenthesis,              // pop and output from the stack until an opening parenthesis is encountered          }  **else** **if** (infix[i] == ')') {    **while** (!is\_empty() && peek() != '(') {                    postfix[j++] = pop();                  postfix[j++] = ' ';                }                pop();                // If an operator is scanned          }  **else** {    **while** (!is\_empty() && precedence(peek()) >= precedence(infix[i])) {                    postfix[j++] = pop();                  postfix[j++] = ' ';                }                push(infix[i]);            }        }    **while** (!is\_empty()) {            postfix[j++] = pop();          postfix[j++] = ' ';        }        postfix[--j] = '\0';    }      **double** evaluate\_postfix(**char**\* postfix) {    **int** i;  **double** op1, op2;    **for** (i = 0; i < **strlen**(postfix); i++) {            // If the scanned character is an operand          // push it to the stack  **if** (**isdigit**(postfix[i])) {    **char** num[10];  **int** j = 0;    **while** (**isdigit**(postfix[i]) || postfix[i] == '.') {                  num[j++] = postfix[i++];              }                num[j] = '\0';              push(**atof**(num));                // If the scanned character is an operator,              // pop two elements from the stack,              // perform the operation and push the result back to the stack          }  **else** **if** (postfix[i] != ' ') {                op2 = pop();              op1 = pop();    **switch** (postfix[i]) {  **case** '+':                  push(op1 + op2);  **break**;  **case** '-':                  push(op1 - op2);  **break**;  **case** '\*':                  push(op1 \* op2);  **break**;  **case** '/':                  push(op1 / op2);  **break**;  **default**:  **printf**("Invalid operator\n");              }            }        }    **return** pop();    }        **int** main() {    **char** infix[MAX\_SIZE], postfix[MAX\_SIZE];    **fgets**(infix, MAX\_SIZE, stdin);      infix[**strlen**(infix) - 1] = '\0';        infix\_to\_postfix(infix, postfix);    **printf**("%.2f\n", evaluate\_postfix(postfix));    **return** 0;    } |