#include <string>

#include <sstream>

#include "stdlib.h"

using namespace std;

char separator = ' ';

/\* clears unused chars \*/

string clear(string equation)/\* clearing the string form any unsued chars like spaces \*/{

string AllstringwedChars = "1234567890.+-\*/()";

string tmp = "";

for ( int i =0 ; i<equation.length() ; i++ ){

size\_t found=AllstringwedChars.find(equation[i]);

if (found != string::npos )

tmp += equation[i];

}

return tmp;

}//end clear

/\* checks if the string contains number \*/

bool isNum(string str){

string numbers = "0123456789.";

string tmp = "";

for( int i = 0 ; i < str.length() ; i ++){

size\_t found = numbers.find(str[i]);

if(found != string::npos)

return true;

}

return false;

}//end isNum

/\* converts form double to string \*/

double string\_To\_Double(string str){

char\* arr = new char[str.length()];

for(int i = 0 ; i < str.length() ; i ++){

arr[i] = str[i];

}

return atof(arr);

}//end string\_To\_Double

/\* converts from a double to string \*/

string double\_To\_String(double d){

std::ostringstream sstream;

sstream << d;

std::string varAsString = sstream.str();

return varAsString;

}//end double\_To\_String

/\* calculates the tokens in the string \*/

string tokens(string equation){

equation = clear(equation);

string result = "";

string numTmp = "";

/\* extracting the tokens \*/

for(int i = 0 ; i < equation.length() ; i ++ ){

if(( (int)equation[i] >= 48 && (int)equation[i] <= 57 ) || equation[i] == '.')/\* double \*/{

numTmp += equation[i];

/\* the last loop, attaching the last num to the string of tokens \*/

if(i == equation.length()-1){

if(result != ""){

result += separator;

}

result += numTmp;

}

}else/\* for [ +, -, \*, /, (, and ) ] \*/{

/\* the separator shouldn't come at first \*/

if(result.length() > 0){

result += separator;

}

if(numTmp != ""){

result += numTmp;

result += separator;

}

result += equation[i];

numTmp = "";

}

}//end for

return result;

}//end tokens

/\* takes the tokens string and calculates the spaces + 1 \*/

int num(string tokens){

int separatorCount = 0;

for(int i = 0 ; i < tokens.length() ; i ++ ){

if(tokens[i] == separator){

separatorCount ++;

}

}

return ++separatorCount;

}//end num

/\* returning the last (char) of the string \*/

char getLast(string stack){

if(stack.length() != 0){

return stack[stack.length()-1];

}else{

return NULL;

}

}//end getLast

/\* returning a string without the last char \*/

string deleteLastChar(string stack){

string tmp = "";

for(int i = 0 ; i < stack.length()-1 ; i ++){

tmp += stack[i];

}

return tmp;

}//end deleteLastChar

/\* returns a string without the last token in given RPN \*/

string deleteLastToken(string RPN){

int separatorCount = 0;

string result = "";

if(num(RPN) == 1){

return "";

}

for(int i = 0 ; i < RPN.length() ; i ++){

if(RPN[i] == separator){

separatorCount ++;

}

if(separatorCount == num(RPN)-1){

break;

}

result += RPN[i];

}//end for

return result;

}//end deleteLastToken

/\* returns the dependence of the operator

according to the rule (sorting with according to there precedences):

"Please Excuse My Dear Aunt Sally": P E M D A S;

Parentheses, Exponentiation (roots and powers), Multiplication, Division, Addition, Subtraction \*/

int precedence(char c){

if(c == '-')return 1;/\* least \*/

else if(c == '+')return 2;

else if(c == '/')return 3;

else if(c == '\*')return 4;

else return 0;/\* error \*/

}//end precedence(char)

int precedence(string c){

if(c == "-")return 1;/\* least \*/

else if(c == "+")return 2;

else if(c == "/")return 3;

else if(c == "\*")return 4;

else return 0;/\* error \*/

}//end precedence(string)

/\* returns the token with this index \*/

string token(int index, string tokens){

if(index > num(tokens))return NULL;

if(num(tokens) == 1){

return tokens;

}

int separatorCount = 0;

string tmp = "";

for(int i = 0 ; i < tokens.length() ; i ++){

if(tokens[i] != separator){

tmp += tokens[i];

}else{

if(++separatorCount == index){

return tmp;

}else{

tmp = "";

}//end else

}//end else

}//end for

/\* if last token \*/

return tmp;

}//end token

/\* generating an RPN code from the tokens string \*/

string RPN(string tokens){

string result = "";/\* the generated RPN \*/

string stack = "";/\* to save the operators in it like a stack \*/

string tmp = "";

for(int i = 0 ; i < num(tokens) ; i ++){

tmp = token(i+1,tokens);

if(tmp.compare("(") == 0){

stack += tmp;

}else if(isNum(tmp)){

result += tmp;

result += separator;

}else if(tmp == "\*" || tmp == "/" || tmp == "+" || tmp == "-"){

if(stack.length() == 0 || getLast(stack) == '(' || precedence(getLast(stack)) < precedence(tmp)){

stack += tmp;

}else{

do{

result += getLast(stack);

result += separator;

stack = deleteLastChar(stack);

}while(stack.length() != 0 && getLast(stack) != '(' && precedence(tmp) < precedence(getLast(stack)));

stack += tmp;

}//end nested else

}else if(tmp.compare(")") == 0){

while(getLast(stack) != '('){

result += getLast(stack);

result += separator;

stack = deleteLastChar(stack);

}//end while

stack = deleteLastChar(stack);

}else{

return NULL;

}//end else

}//end for

while(stack.length() != 0){

result += getLast(stack);

result += separator;

stack = deleteLastChar(stack);

}

result = deleteLastChar(result);

return result;

}//end RPN

/\* returns the proccessed string for RPN (it's solution) \*/

string processRPN(string RPN){

string process = "";

string tmp = "";

double d1,d2;

for(int i = 0 ; i < num(RPN) ; i ++){

tmp = token(i+1,RPN);

if(isNum(tmp)){

if(process != ""){

process += separator;

}

process += tmp;

}else{

int length = num(process);

d2 = string\_To\_Double(token(length,process));

d1 = string\_To\_Double(token(length-1,process));

process = deleteLastToken(process);

process = deleteLastToken(process);

if(process != ""){

process += separator;

}

if(tmp == "\*"){

process += double\_To\_String(d1\*d2);

}else if(tmp == "/"){

process += double\_To\_String(d1/d2);

}else if(tmp == "+"){

process += double\_To\_String(d1+d2);

}else if(tmp == "-"){

process += double\_To\_String(d1-d2);

}

}//end else

}//end for

return process;

}//end processRPN

/\* just solve the equation \*/

string solveEquation(string equation){

return processRPN(RPN(tokens(equation)));

}

/\* link: http://www.seas.gwu.edu/~csci133/fall04/133f04toRPN.html \*/