namespace MyExcel1

{

//клас, що виключає помилки для анализатора

class ParserException : ApplicationException

{

public ParserException(string str) : base(str) { }

public override string ToString()

{ return Message; }

}

class Parser

{

enum Types { NONE, DELIMITER, VARIABLE, NUMBER}; //лексеми.

enum Errors { SYNTAX, UNBALPARENS, NOEXP, DIVBYZERO }; // помилки.

string exp; // рядок виразу

int expIdx; // поточний індекс у виразі

string token; // поточна лексема

Types tokType; // тип лексеми

// Масив для змінних

double[] vars = new double[26];

public Parser()

{

// Ініціалізуємо змінні нульовими значеннями.

for (int i = 0; i < vars.Length; i++)

vars[i] = 0.0;

}

/\* Точка входу аналізатора. \*/

public double Evaluate(string expstr)

{

double result;

exp = expstr;

expIdx = 0;

try

{

GetToken();

if (token == "")

{

SyntaxErr(Errors.NOEXP); // Вираз відсутній

return 0.0;

}

EvalExp1(out result);

if (token != "") /\* остання лексема повинна бути нульова \*/

SyntaxErr(Errors.SYNTAX);

return result;

}

catch (ParserException exc)

{

Console.WriteLine(exc);

return 0.0;

}

}

// Обробка присвоювання

void EvalExp1(out double result)

{

int varIdx;

Types ttokType;

string temptoken;

if (tokType == Types.VARIABLE)

{

// зберегти стару лексему

temptoken = String.Copy(token);

ttokType = tokType;

// обчислити індекс змінної

varIdx = Char.ToUpper(token[0]) - 'A';

GetToken();

if (token != "=")

{

PutBack();// повернути поточну лексему в поток

//відновити стару лексему - це не присвоювання

token = String.Copy(temptoken);

tokType = ttokType;

}

else

{

GetToken();// отримати наступну частину виразу

EvalExp2(out result);

vars[varIdx] = result;

return;

}

}

EvalExp2(out result);

}

// Додавання або віднімання двох доданків

void EvalExp2(out double result)

{

string op;

double partialResult;

EvalExp3(out result);

while ((op = token) == "+" || op == "-")

{

GetToken();

EvalExp3(out partialResult);

switch (op)

{

case "-":

result = result - partialResult;

break;

case "+":

result = result + partialResult;

break;

}

}

}

// Виконуємо множення чи ділення доух множників

void EvalExp3(out double result)

{

string op;

double partialResult = 0.0;

EvalExp4(out result);

while ((op = token) == "\*" || op == "/" || op == "%")

{

GetToken();

EvalExp4(out partialResult);

switch (op)

{

case "\*":

result = result \* partialResult;

break;

case "/":

if (partialResult == 0.0)

SyntaxErr(Errors.DIVBYZERO);

result = result / partialResult;

break;

case "%":

if (partialResult == 0.0)

SyntaxErr(Errors.DIVBYZERO);

result = (int)result % (int)partialResult;

break;

}

}

}

// Піднесення до степені

void EvalExp4(out double result)

{

double partialResult, ex;

int t;

EvalExp5(out result);

if (token == "^")

{

GetToken();

EvalExp4(out partialResult);

ex = result;

if (partialResult == 0.0)

{

result = 1.0;

return;

}

for (t = (int)partialResult - 1; t > 0; t--)

result = result \* (double)ex;

}

}

// Множення унарних операторів + й -.

void EvalExp5(out double result)

{

string op;

op = "";

if ((tokType == Types.DELIMITER) && token == "+" || token == "-")

{

op = token;

GetToken();

}

EvalExp6(out result);

if (op == "-") result = -result;

}

// Обчислення виразів в дужках

void EvalExp6(out double result)

{

if ((token == "("))

{

GetToken();

EvalExp2(out result);

if (token != ")")

SyntaxErr(Errors.UNBALPARENS);

GetToken();

}

else Atom(out result);

}

// Одержання значення числа, або змінної

void Atom(out double result)

{

switch (tokType)

{

case Types.NUMBER:

try

{

result = Double.Parse(token);

}

catch (FormatException)

{

result = 0.0;

SyntaxErr(Errors.SYNTAX);

}

GetToken();

return;

case Types.VARIABLE:

result = FindVar(token);

GetToken();

return;

default:

result = 0.0;

SyntaxErr(Errors.SYNTAX);

break;

}

}

// Повертаємо значення змінної

double FindVar(string vname)

{

if (!Char.IsLetter(vname[0]))

{

SyntaxErr(Errors.SYNTAX);

return 0.0;

}

return vars[Char.ToUpper(vname[0]) - 'A'];

}

//повертаємо лексему у вхідний потік.

void PutBack()

{

for (int i = 0; i < token.Length; i++) expIdx--;

}

// Обробляємо синтаксичну помилку

void SyntaxErr(Errors error)

{

string[] err ={

"Синтаксическая оошибка",

"Дисбаланс скобок",

"Выражение отсутствет",

"Деление на нуль"};

throw new ParserException(err[(int)error]);

}

// отримуємо наступну лексему

void GetToken()

{

tokType = Types.NONE;

token = "";

if (expIdx == exp.Length) return; //кінець виразу

// пропускаємо пробіл

while (expIdx < exp.Length &&

Char.IsWhiteSpace(exp[expIdx])) ++expIdx;

// Хвостовий пробіл

if (expIdx == exp.Length) return;

if (IsDelim(exp[expIdx]))

{

token += exp[expIdx];

expIdx++;

tokType = Types.DELIMITER;

}

else if (Char.IsLetter(exp[expIdx]))

{

// Це змінна?

while (!IsDelim(exp[expIdx]))

{

token += exp[expIdx];

expIdx++;

if (expIdx >= exp.Length) break;

}

tokType = Types.VARIABLE;

}

else if (Char.IsDigit(exp[expIdx]))

{

// Це число?

while (!IsDelim(exp[expIdx]))

{

token += exp[expIdx];

expIdx++;

if (expIdx >= exp.Length) break;

}

tokType = Types.NUMBER;

}

}

// Метод повертає значення true,

// якщо с - роздільник

bool IsDelim(char c)

{

if (("+-/\*%^=()".IndexOf(c) != -1))

return true;

return false;

}

}

}