Министерство образования Республики Беларусь Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ

И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

По теме “Методы трансляции”

Выполнила: студентка гр. 053501 Шурко Т.А.

Проверил: ассистент кафедры информатики Гриценко Н. Ю.

Минск 2023

Содержание

[1. Цель работы 3](#_Toc7030)

[2. Теория 3](#_Toc7031)

[3. Программа и комментарии 4](#_Toc7035)

4. Демонстрация работы 6

5. Демонстрация нахождения лексических ошибок 7

Вывод 10

Приложение 1. Таблица 11

Приложение 2. Текст программы 13

Приложение 3. Текст программ 19

# Цель работы

Разработка лексического анализатора подмножества языка программирования, определённого в лабораторной работе 1. Программа анализа определяет лексические правила и выполняет перевод потока символов программ лабораторной работы 1 в поток лексем (токенов).

# Теория

Лексический анализатор (сканер) читает поток символов, составляющих исходную программу, и группирует эти символы в значащие последовательности, называющиеся лексемами.

Лексема – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своём составе других структурных единиц языка. Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и т.п.

На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору. Для каждой лексемы сканер строит выходной токен (англ. token – знак, символ) вида

‹имя\_токена, значение\_атрибута›

Первый компонент токена, имя\_токена, представляет собой абстрактный символ, использующийся во время синтаксического анализа, а второй компонент, значение атрибута, указывает на запись в таблице идентификаторов, соответствующую данному токену.

Предположим, например, что исходная программа содержит инструкцию присваивания a = b + c \* d.

Символы в этом присваивании могут быть сгруппированы в следующие лексемы и отображены в следующие токены:

1) a представляет собой лексему, которая может отображаться в токен ‹id, 1›, где id – абстрактный символ, обозначающий идентификатор, а 1 указывает запись в таблице идентификаторов для a, в которой хранится такая информация как имя и тип идентификатора;

2) cимвол присваивания = представляет собой лексему, которая отображается в токен ‹=›. Поскольку этот токен не требует значения атрибута, второй компонент данного токена 2 опущен. В качестве имени токена может быть использован любой абстрактный символ, например, такой, как «assign», но для удобства записи в качестве имени абстрактного символа можно использовать саму лексему;

3) b представляет собой лексему, которая отображается в токен ‹id, 2›, где 2 указывает на запись в таблице идентификаторов для b;

4) + является лексемой, отображаемой в токен ‹+›;

5) c – лексема, отображаемая в токен ‹id, 3›, где 3 указывает на запись в таблице идентификаторов для c;

6) \* – лексема, отображаемая в токен ‹\*›;

7) d – лексема, отображаемая в токен ‹id, 4›, где 4 указывает на запись в таблице идентификаторов для d.

Пробелы, разделяющие лексемы, лексическим анализатором пропускаются.

Представление инструкции присваивания после лексического анализа в виде последовательности токенов примет следующий вид: ‹id, 1›‹=›‹id, 2›‹+›‹\*›‹id, 3›‹id, 4›.

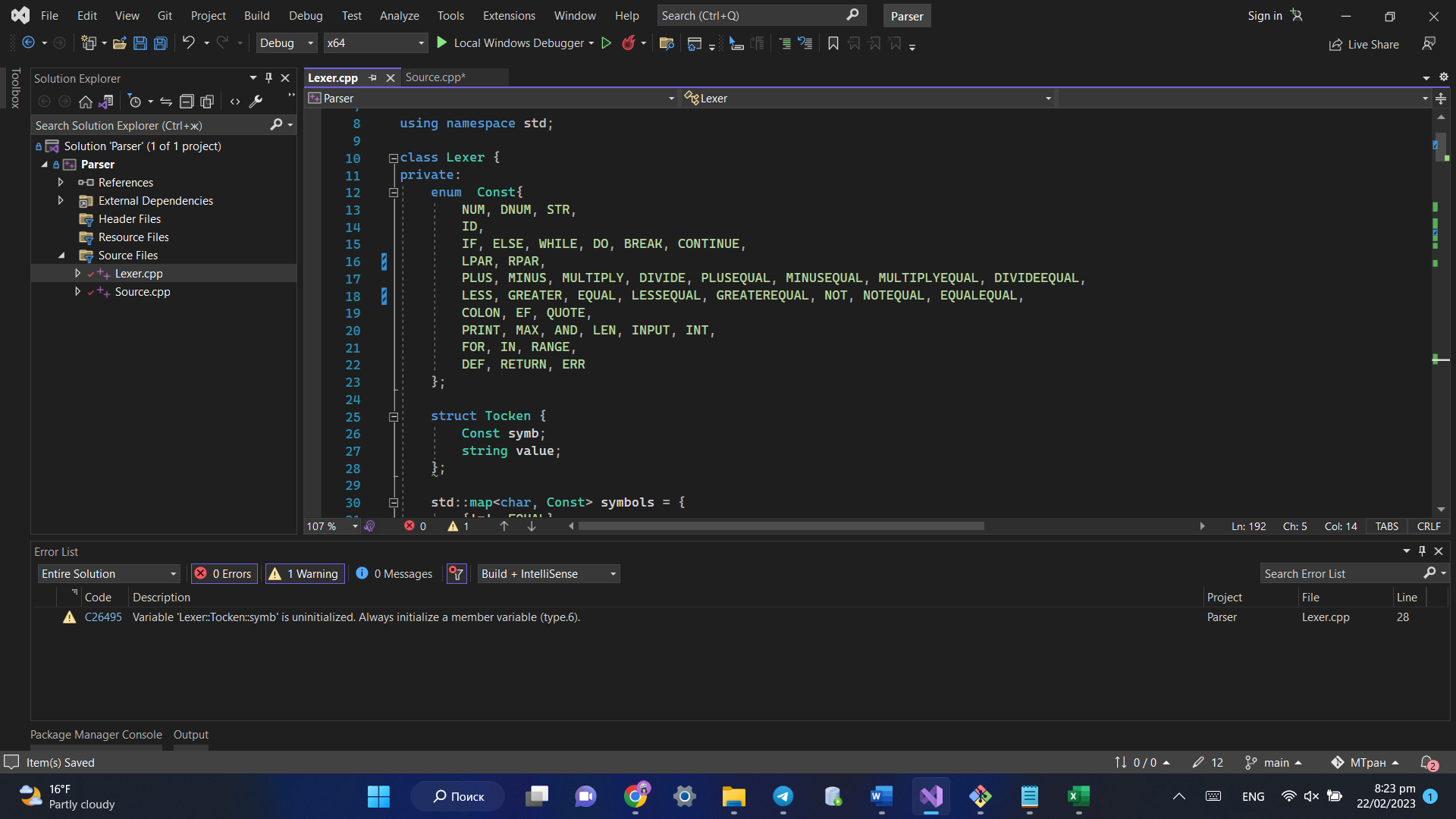
1. Программа и комментарии

Рис. 3.1 Перечисление Const

Перечисление Const (Рис. 3.1) представляет собой набор основных констант выбранного языка программирования (Python).

* NUM, DNUM, STR ­– представляют основные типы данных: int, float, str.
* ID – для обозначения идентификатора (переменной).
* IF, ELSE, WHILE, DO, BREAK, CONTINUE, PRINT, AND, MAX, LEN, INPUT, INT, FOR, IN, RANGE, RETURN, DEF – ключевые слова языка.
* LPAR, RPAR – скобки.
* PLUS(EQUAL), MINUS(EQUAL), MULTIPLY(EQUAL), DIVIDE(EQUAL) – операторы + (+=), – (– =), \* (\*=), / (/=).
* LESS, GREATER, EQUAL, LESSEQUAL, GREATEREQUAL, NOT, NOTEQUAL, EQUALEQUAL – операторы сравнения <, >, =, <=, >=, !, !=, ==.
* ERR – для обозначения ошибки.
* COLON – для обозначения :.
* QUOTE – кавычки и одинарные, и двойные.
* EF – конец файла с исходным кодом.

Ниже приведена таблица соответствия основных констант и их int-значения, так как перечисление представляет набор констант, каждой из которых соответствует единственное число (Приложение 1).

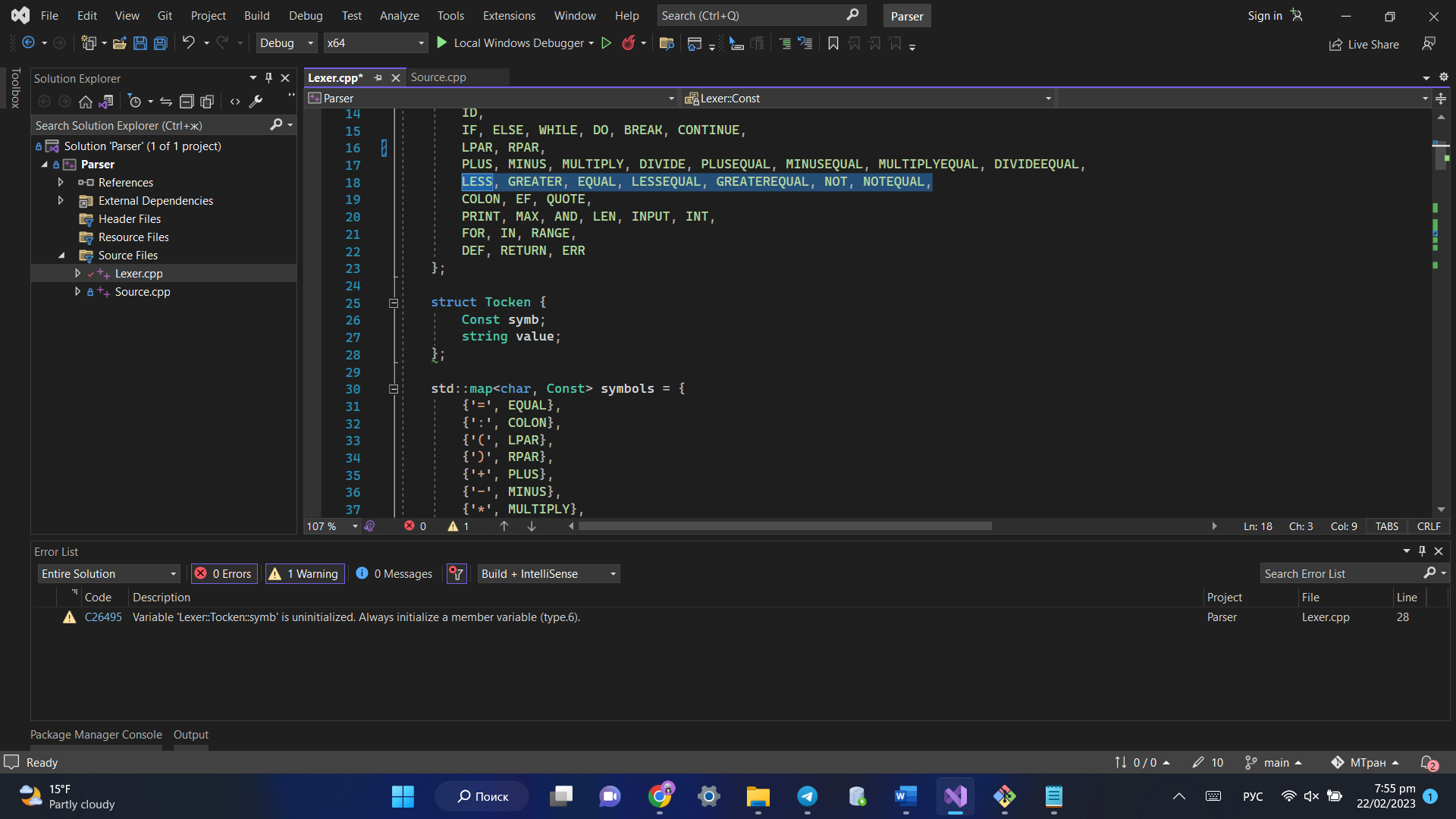
Сам токен представляет из себя структуру с 2 полями:

Рис. 3.2 Структура Tocken

Symb – это элемент перечисления, а value – значение токена, если оно необходимо.

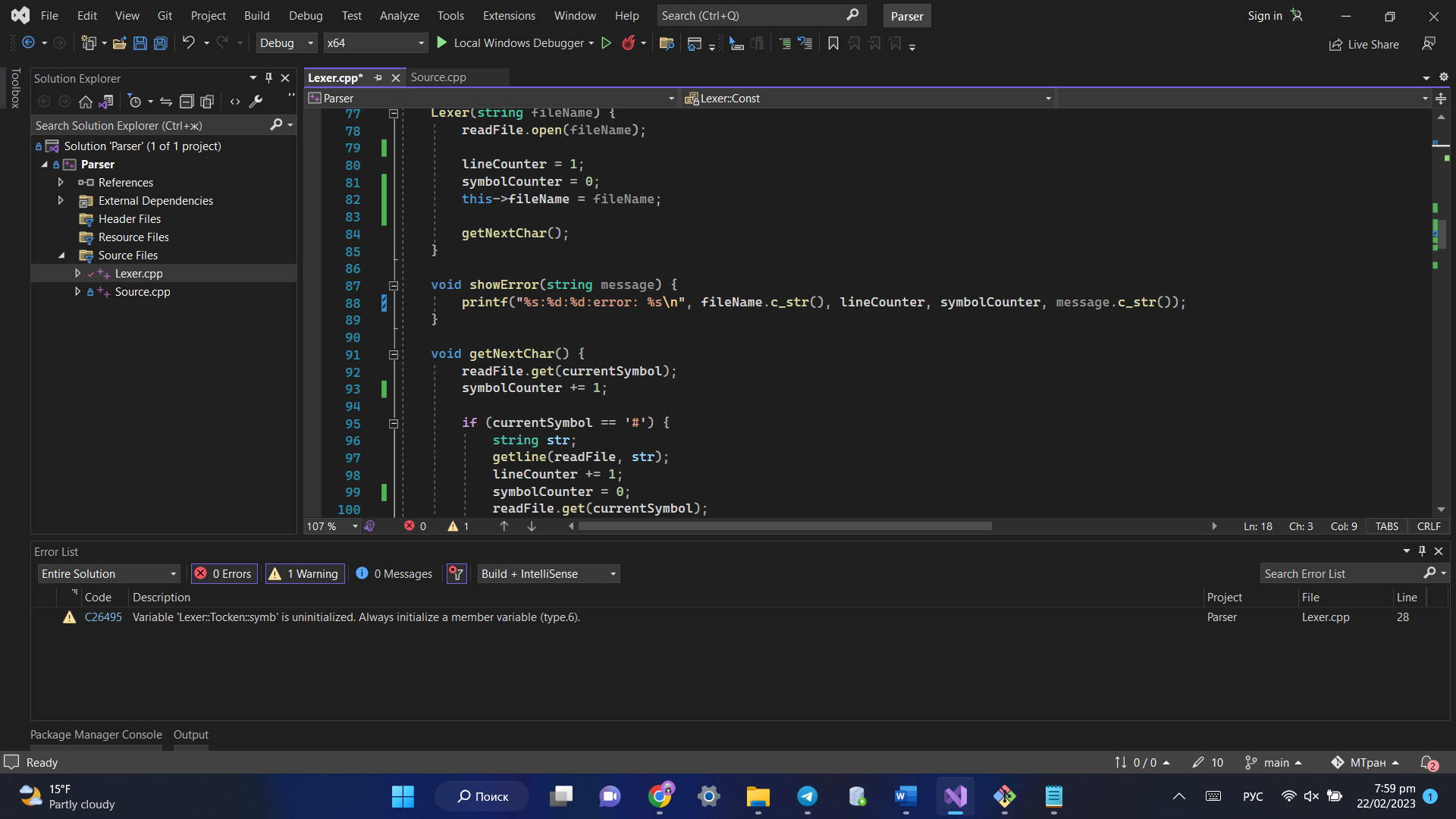
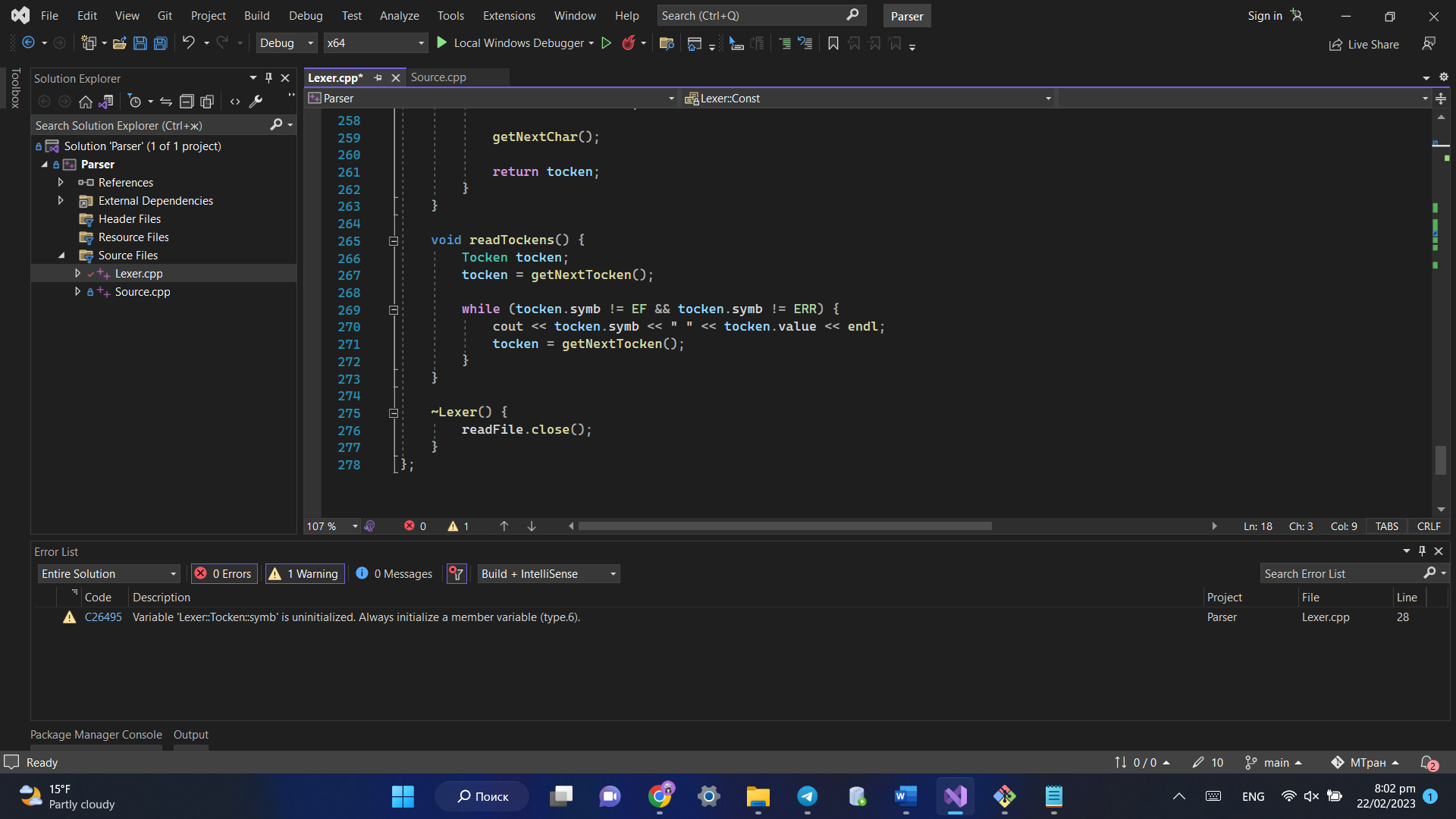


Рис. 3.3 Метод вывода ошибки showError

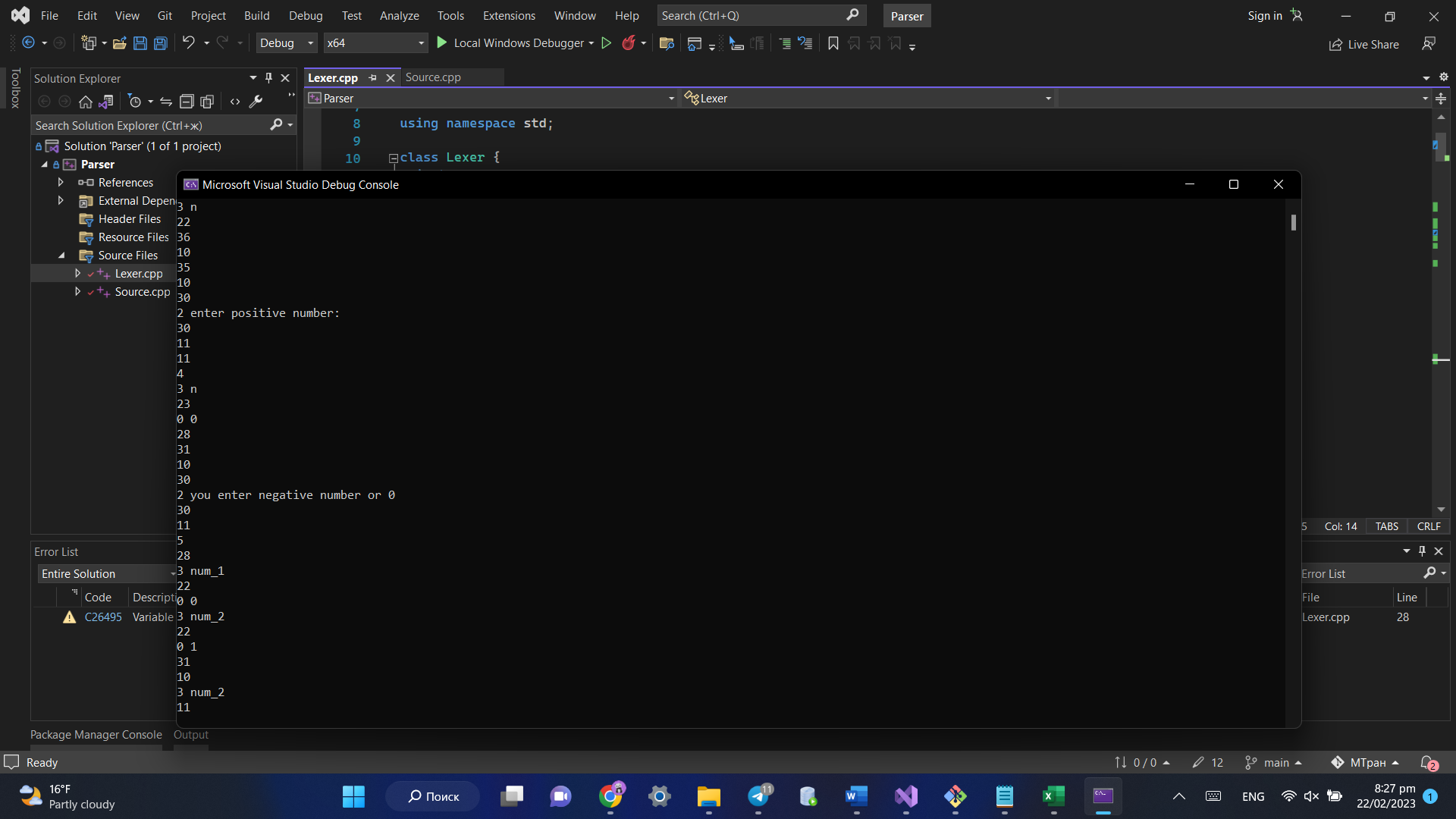
Метод класса showError (Рис. 3.3) выводит переданное пользователем сообщение об ошибке, а также имя файла, номер строки и номер символа, который вызвал ошибку.

Рис. 3.4 Вспомогательный метод для вывода токенов

Данная информация хранится в приватных полях класса fileName, lineCounter и symbolCounter. Для демонстрации работы был написан метод readTockens(), который просто выводит все токены. Программа прекращает обработку по достижению конца файла или при встрече первой ошибки, найденной лексическим анализатором.

1. Демонстрация работы

Рассмотрим в качестве входного файла задачу о нахождение n чисел Фибоначчи, введенного пользователем (Приложение 3).

В результате разбиения на токены получаем следующий результат:

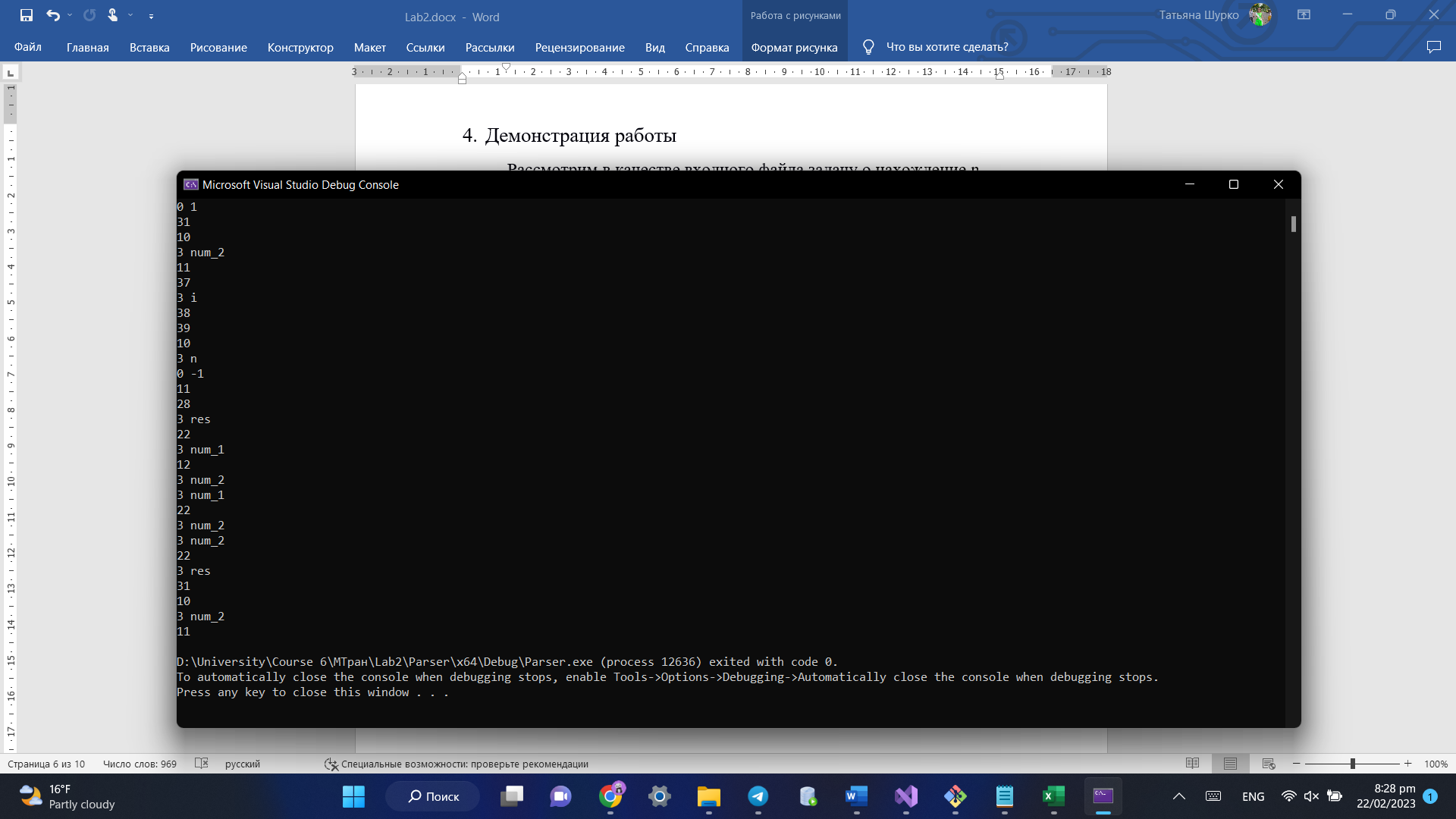
Рис. 4.1. Результат разбиение на токены задачи 1

Рис. 4.2. Результат разбиение на токены задачи 1

Программа отработала, разбиение на токены произошло, никаких ошибок в результате анализа найдено не было (см. рис. 4.1, 4.2).

Аналогично были протестированы остальные 2 задачи (Приложение 3).

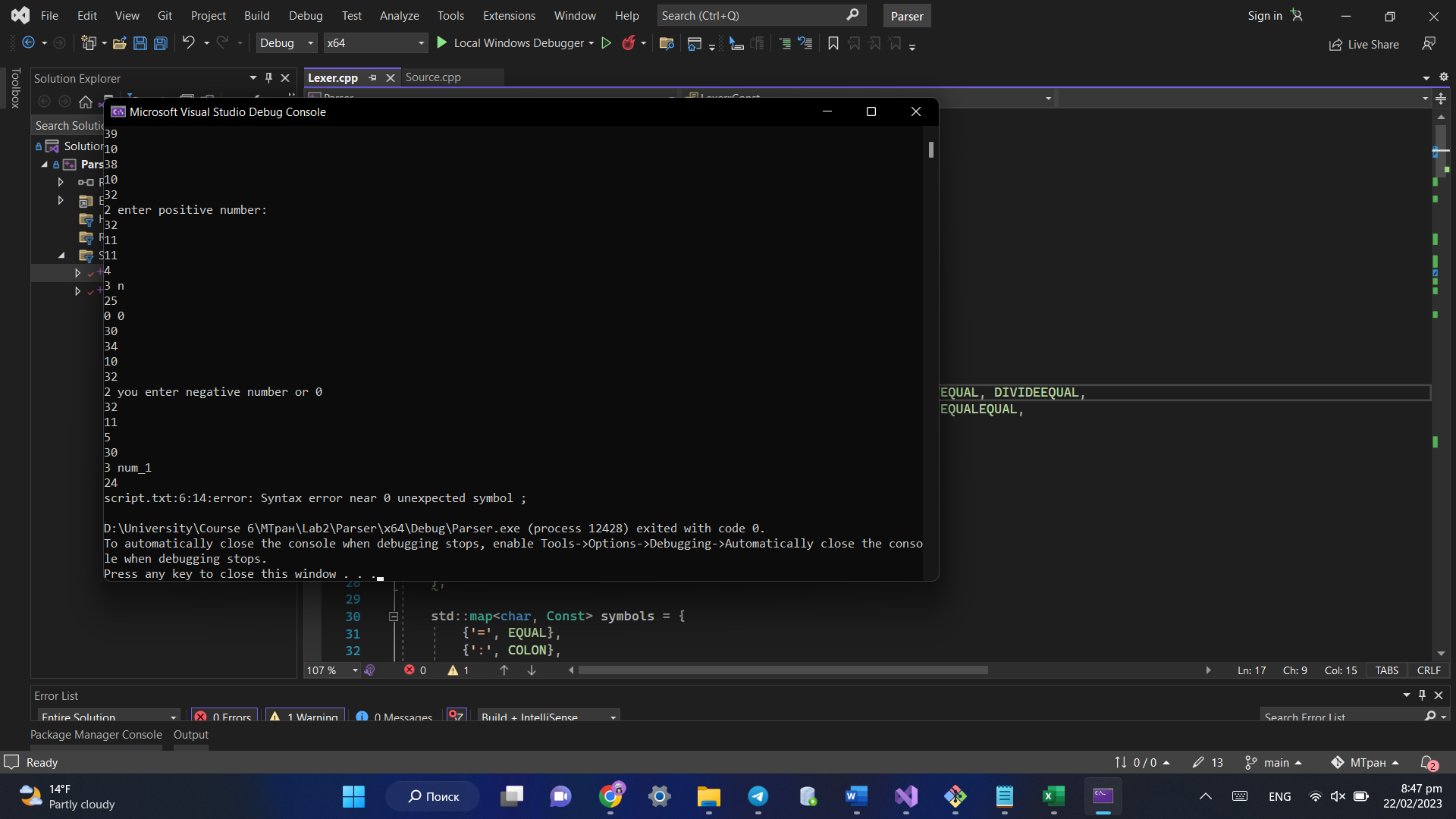
1. Демонстрация нахождения лексических ошибок

Рис. 5.1 Лексическая ошибка “;”.

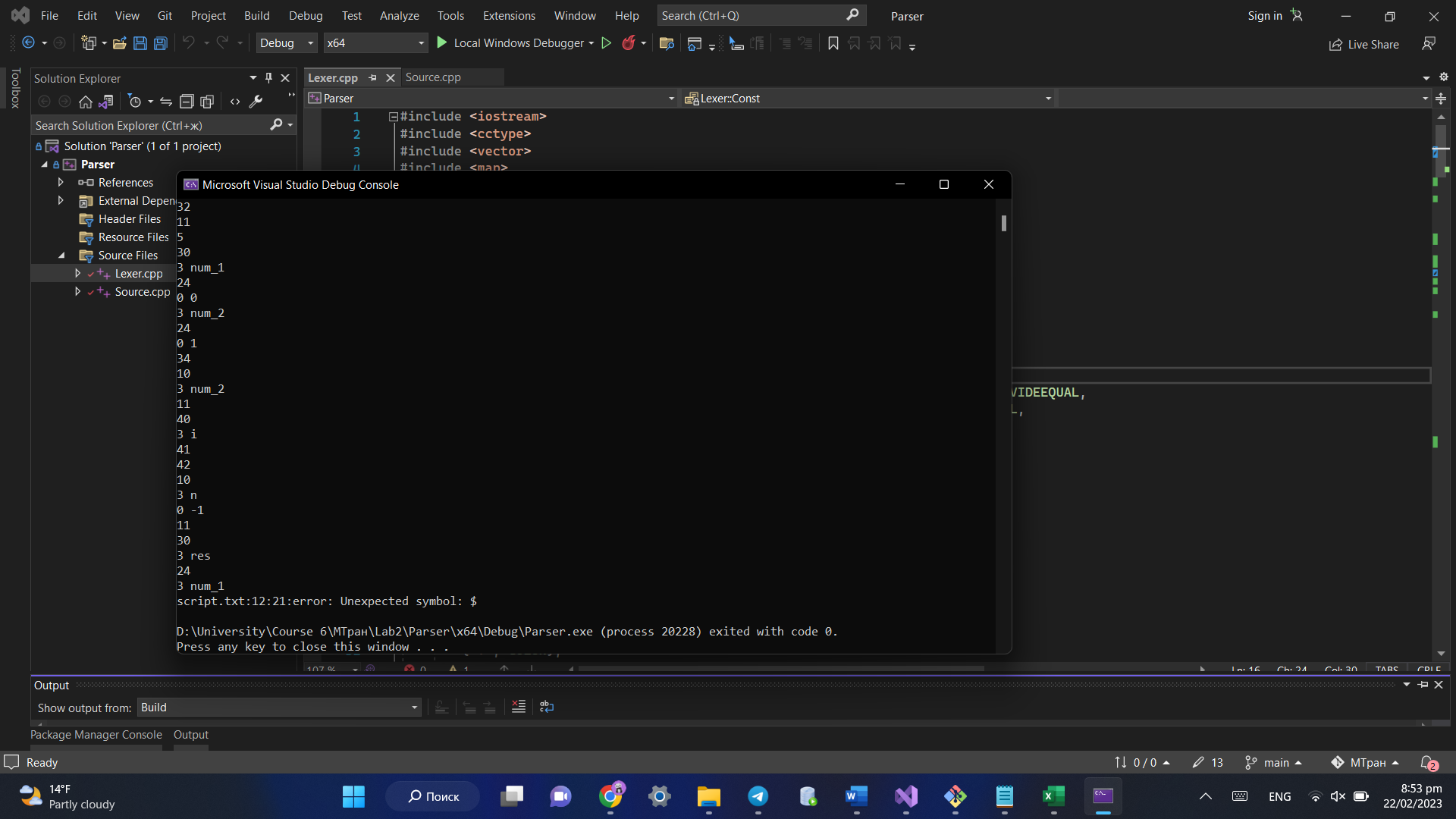
На Рис. 5.1 продемонстрирована лексическая ошибка, наличие “;”. В языке программирования Python “;” ставить не нужно.

Рис. 5.2 Лексическая ошибка. Наличие постороннего символа в выражении.

На Рис. 5.2 показана реакция программного продукта на наличие постороннего символа в выражении res = num\_1 $+ num\_2.

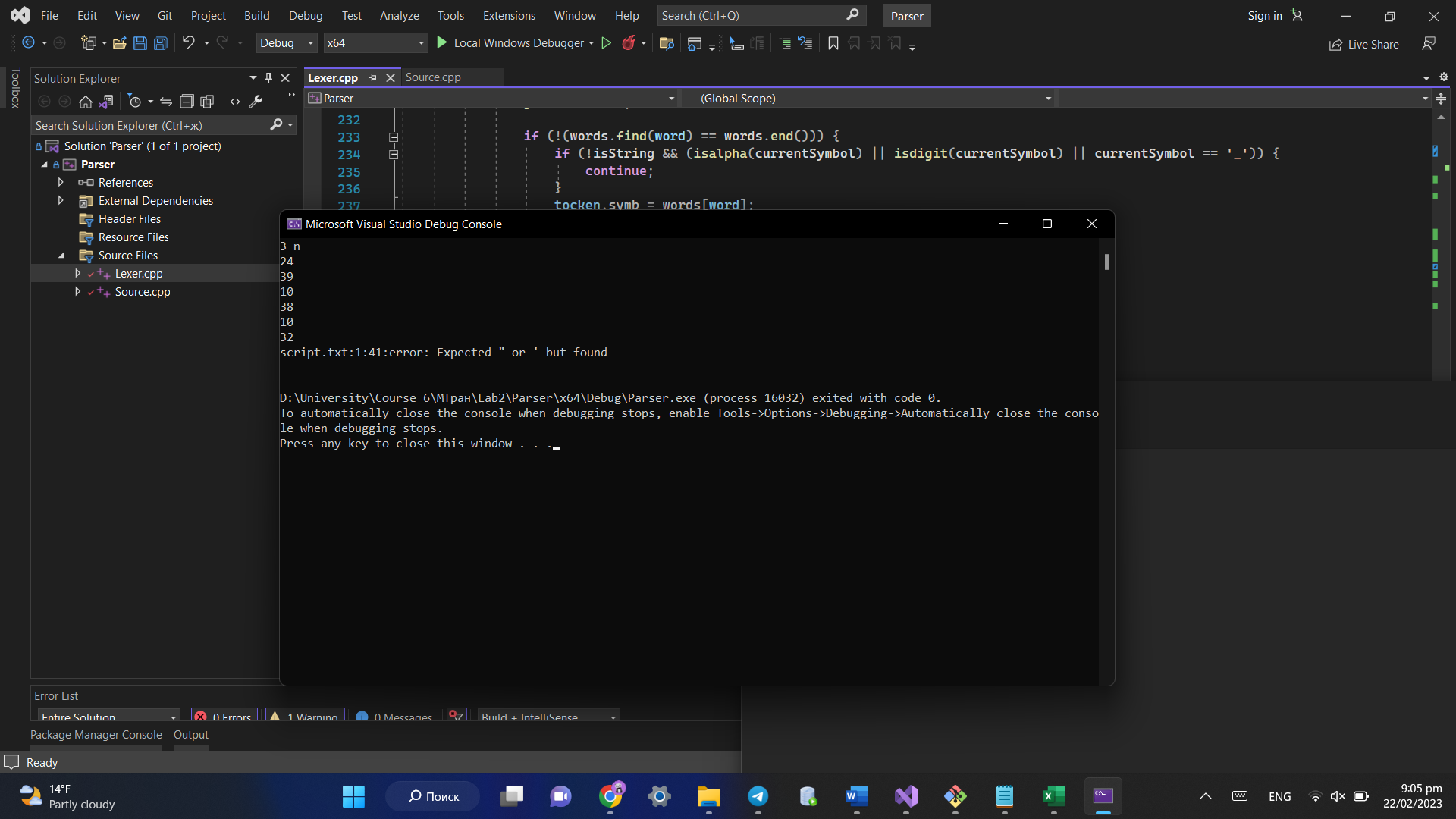


Рис. 5.3 Лексическая ошибка. Отсутствие закрывающейся кавычки.

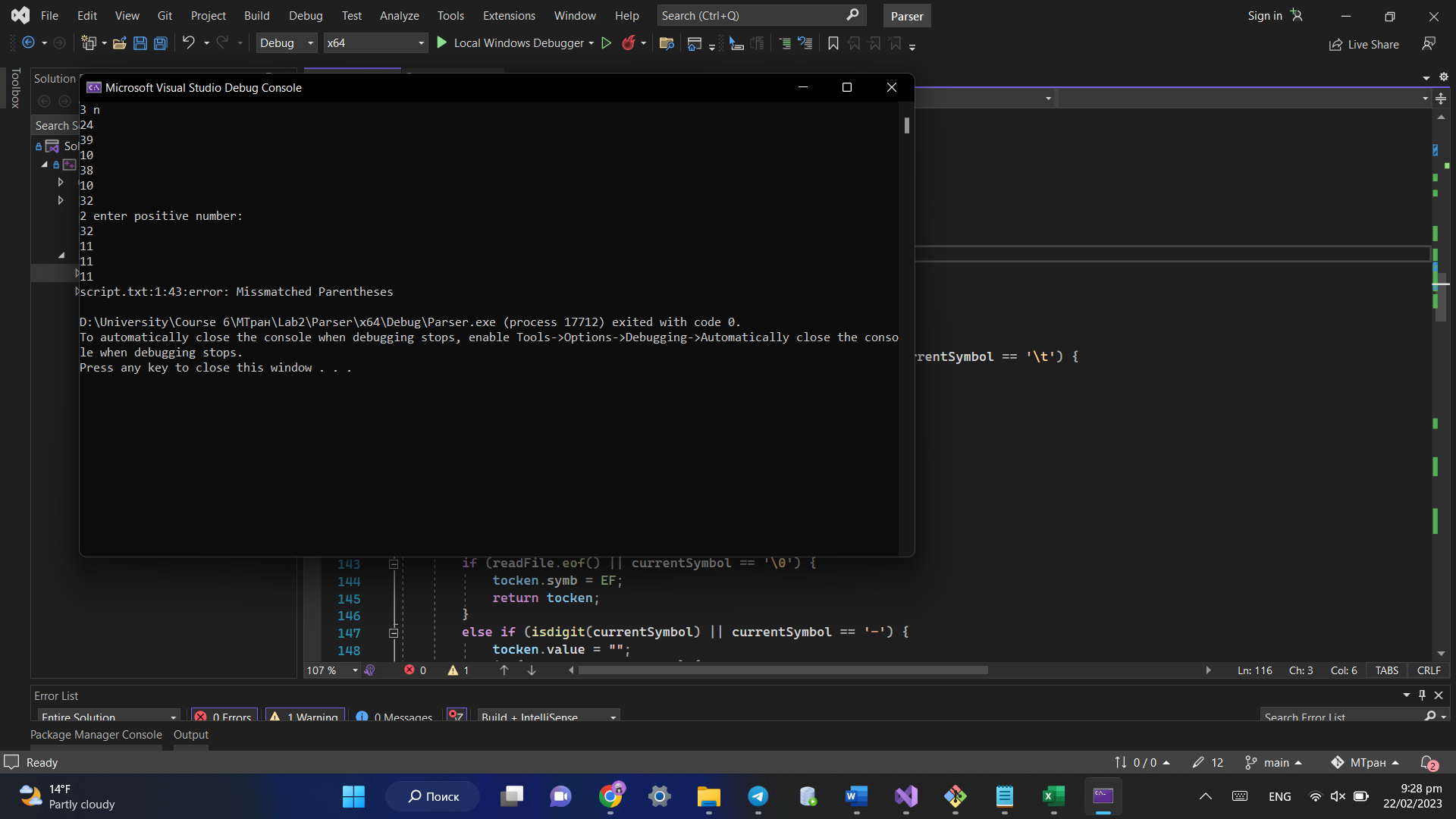
На Рис. 5.3 показана обработка лексической ошибки при работе со строковыми переменными: отсутствие закрывающихся кавычек n = int(input("Enter positive number: )).

Рис. 5.4 Лексическая ошибка. Несоответствие скобок

На Рис. 5.4 реакция программного продукта на несоответствие количества скобок n = int(input("Enter positive number: "))). В данном примере количество закрывающихся скобок больше, чем открывающихся.

## Вывод

В результате выполнения лабораторной работы была изучена теория о лексических анализаторах и лексемах. Рассмотрена стадия лексического анализа.

Разработан лексический анализатор подмножества языка программирования Python, определенного в лабораторной работе 1. Программа анализатора определяет лексические правила и выполняет перевод потока символов программ лабораторной работы 1 в поток лексем (токенов).

Приложение 1. Таблица

|  |  |
| --- | --- |
| Int-value | name |
| 0 | NUM |
| 1 | DNUM |
| 2 | STR |
| 3 | ID |
| 4 | IF |
| 5 | ELSE |
| 6 | WHILE |
| 7 | DO |
| 8 | BREAK |
| 9 | CONTINUE |
| 10 | LPAR |
| 11 | RPAR |
| 12 | LBR |
| 13 | RBR |
| 14 | PLUS |
| 15 | MINUS |
| 16 | MULTIPLY |
| 17 | DIVIDE |
| 18 | PLUSEQUAL |
| 19 | MINUSEQUAL |
| 20 | MULTIPLYEQUAL |
| 21 | DIVIDEEQUAL |
| 22 | LESS |
| 23 | GREATER |
| 24 | EQUAL |
| 25 | LESSEQUAL |
| 26 | GREATEREQUAL |
| 27 | NOT |
| 28 | NOTEQUAL |
| 29 | EQUALEQUAL |
| 30 | COLON |
| 31 | EF |
| 32 | QUOTE |
| 33 | COMMA |
| 34 | PRINT |
| 35 | MAX |
| 36 | AND |
| 37 | LEN |
| 38 | INPUT |
| 39 | INT |
| 40 | FOR |
| 41 | IN |
| 42 | RANGE |
| 43 | DEF |
| 44 | RETURN |
| 45 | ERR |

Приложение 2. Текст программы

#include <iostream>

#include <cctype>

#include <vector>

#include <map>

#include <string>

#include <fstream>

using namespace std;

class Lexer {

private:

enum Const{

NUM, DNUM, STR,

ID,

IF, ELSE, WHILE, DO, BREAK, CONTINUE,

LPAR, RPAR, LBR, RBR,

PLUS, MINUS, MULTIPLY, DIVIDE, PLUSEQUAL, MINUSEQUAL, MULTIPLYEQUAL, DIVIDEEQUAL,

LESS, GREATER, EQUAL, LESSEQUAL, GREATEREQUAL, NOT, NOTEQUAL, EQUALEQUAL,

COLON, EF, QUOTE, COMMA,

PRINT, MAX, AND, LEN, INPUT, INT,

FOR, IN, RANGE,

DEF, RETURN, ERR

};

struct Tocken {

Const symb;

string value;

};

std::map<char, Const> symbols = {

{'=', EQUAL},

{':', COLON},

{'(', LPAR},

{')', RPAR},

{'[', LBR},

{']', RBR},

{'+', PLUS},

{'-', MINUS},

{'\*', MULTIPLY},

{'/', DIVIDE},

{'<', LESS},

{'>', GREATER},

{'!', NOT},

{'"', QUOTE},

{'\'', QUOTE},

{',', COMMA},

};

std::map<string, Const> words = {

{"if", IF},

{"else", ELSE},

{"do", DO},

{"while", WHILE},

{"print", PRINT},

{"max", MAX},

{"for", FOR},

{"in", IN},

{"range", RANGE},

{"break", BREAK},

{"continue", CONTINUE},

{"and", AND},

{"len", LEN},

{"int", INT},

{"input", INPUT},

{"def", DEF}

};

char currentSymbol;

string fileName;

int lineCounter;

int symbolCounter;

int parenthesesCounter = 0;

bool isString = false;

bool isU = false;

ifstream readFile;

public:

Lexer(string fileName) {

readFile.open(fileName);

lineCounter = 1;

symbolCounter = 0;

this->fileName = fileName;

getNextChar();

}

void showError(string message) {

printf("%s:%d:%d:error: %s\n", fileName.c\_str(), lineCounter, symbolCounter, message.c\_str());

}

void removeComment() {

if (currentSymbol == '#') {

string str;

getline(readFile, str);

lineCounter += 1;

symbolCounter = 0;

getNextChar();

}

}

string checkParentless() {

if (parenthesesCounter != 0) {

string err = "Missmatched Parentheses";

this->showError(err);

return err;

}

parenthesesCounter = 0;

return "";

}

void getNextChar() {

readFile.get(currentSymbol);

symbolCounter += 1;

this->removeComment();

if (readFile.eof()) {

currentSymbol = '\0';

}

}

Tocken getNextTocken() {

Tocken tocken;

while (currentSymbol == ' ' || currentSymbol == '\n' || currentSymbol == '\t') {

if (currentSymbol == '\n') {

string check = checkParentless();

if (check != "") {

return Tocken(ERR, check);

}

lineCounter += 1;

symbolCounter = 0;

}

getNextChar();

}

if (readFile.eof() || currentSymbol == '\0') {

tocken.symb = EF;

return tocken;

}

else if (isdigit(currentSymbol) || currentSymbol == '-') {

tocken.value = "";

if (currentSymbol == '-') {

getNextChar();

if (!isdigit(currentSymbol)) {

tocken.symb = symbols[currentSymbol];

getNextChar();

return tocken;

}

tocken.value += '-';

}

while (isdigit(currentSymbol) && currentSymbol != '\0') {

tocken.value += currentSymbol;

getNextChar();

}

if (currentSymbol == '.') {

tocken.value += currentSymbol;

tocken.symb = DNUM;

getNextChar();

}

else if (currentSymbol == ' ' || currentSymbol == '\0' || currentSymbol == '\n' || (symbols.find(currentSymbol) != symbols.end())) {

tocken.symb = NUM;

return tocken;

}

else {

string err = "Syntax error near ";

err += tocken.value;

err += " unexpected symbol ";

err += currentSymbol;

this->showError(err);

tocken.symb = ERR;

tocken.value = err;

getNextChar();

return tocken;

}

while (isdigit(currentSymbol)) {

tocken.value += currentSymbol;

getNextChar();

}

return tocken;

}

else if (!(symbols.find(currentSymbol) == symbols.end())) {

tocken.symb = symbols[currentSymbol];

getNextChar();

if (tocken.symb == GREATER && this->currentSymbol == '=') {

tocken.symb = GREATEREQUAL;

getNextChar();

}

else if (tocken.symb == LESS && this->currentSymbol == '=') {

tocken.symb = LESSEQUAL;

getNextChar();

}

else if (tocken.symb == NOT && this->currentSymbol == '=') {

tocken.symb = NOTEQUAL;

getNextChar();

}

else if (tocken.symb == EQUAL && this->currentSymbol == '=') {

tocken.symb = EQUALEQUAL;

getNextChar();

}

else if (tocken.symb == PLUS && this->currentSymbol == '=') {

tocken.symb = PLUSEQUAL;

getNextChar();

}

else if (tocken.symb == MINUS && this->currentSymbol == '=') {

tocken.symb = MINUSEQUAL;

getNextChar();

}

else if (tocken.symb == MULTIPLY && this->currentSymbol == '=') {

tocken.symb = MULTIPLYEQUAL;

getNextChar();

}

else if (tocken.symb == DIVIDE && this->currentSymbol == '=') {

tocken.symb = DIVIDEEQUAL;

getNextChar();

}

if (tocken.symb == QUOTE) {

isString = !isString;

if (isString) {

isU = false;

}

}

if (tocken.symb == LPAR) {

parenthesesCounter += 1;

}

else if (tocken.symb == RPAR) {

parenthesesCounter -= 1;

}

return tocken;

}

else if (isalpha(currentSymbol)) {

string word = "";

bool isSpecialWord = false;

while ((!isString && (isalpha(currentSymbol) || isdigit(currentSymbol) || currentSymbol == '\_')) || (isString && (currentSymbol != '"' && currentSymbol != '\''))) {

word += tolower(currentSymbol);

if (currentSymbol == 'u') {

isU = true;

}

getNextChar();

if (!(words.find(word) == words.end())) {

if (!isString && (isalpha(currentSymbol) || isdigit(currentSymbol) || currentSymbol == '\_')) {

continue;

}

tocken.symb = words[word];

isSpecialWord = true;

return tocken;

}

if (isString && currentSymbol == '\n') {

string err = "Expected \" or ' but found ";

err += this->currentSymbol;

this->showError(err);

tocken.symb = ERR;

tocken.value = err;

getNextChar();

return tocken;

}

}

if (isString) {

tocken.symb = STR;

tocken.value = word;

return tocken;

}

else if (!isString && !isSpecialWord) {

tocken.symb = ID;

tocken.value = word;

return tocken;

}

}

else {

string err = "Unexpected symbol: ";

err += this->currentSymbol;

this->showError(err);

tocken.symb = ERR;

tocken.value = err;

getNextChar();

return tocken;

}

}

void readTockens() {

Tocken tocken;

tocken = getNextTocken();

while (tocken.symb != EF && tocken.symb != ERR) {

cout << tocken.symb << " " << tocken.value << endl;

tocken = getNextTocken();

}

}

~Lexer() {

readFile.close();

}

};

Приложение 3. Текст программ

1. Нахождения n чисел Фибоначчи, введенного пользователем

n = int(input("Enter positive number: "))  
  
if n <= 0:  
 print("You enter negative number or 0")  
else:  
 num\_1 = 0  
 num\_2 = 1  
   
 print(num\_2)  
   
 for i in range(n-1):  
 res = num\_1 + num\_2  
 num\_1 = num\_2  
 num\_2 = res  
   
 print(num\_2)

1. Дана последовательность натуральных чисел, завершающаяся числом 0. Определите, какое наибольшее число подряд идущих элементов этой последовательности равны друг другу.

prev = -1  
curr\_rep\_len = 0  
max\_rep\_len = 0  
  
element = int(input())  
  
while element != 0:  
 if prev == element:  
 curr\_rep\_len += 1  
 else:  
 prev = element  
 max\_rep\_len = max(max\_rep\_len, curr\_rep\_len)  
 curr\_rep\_len = 1  
   
 element = int(input())  
   
max\_rep\_len = max(max\_rep\_len, curr\_rep\_len)  
print(max\_rep\_len)

1. Вычислить факториал введенного пользователем числа.

number = int(input("Enter a number: "))  
factorial = 1  
  
if number < 0:  
 print("Sorry, factorial does not exist for negative numbers")  
elif number == 0:  
 print("The factorial of 0 is 1")  
else:  
 for i in range(1, number + 1):  
 factorial = factorial\*i  
 print(factorial)