Министерство образования Республики Беларусь Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ

И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №3

По теме “Синтаксический анализатор”

Выполнила: студентка гр. 053501 Шурко Т.А.

Проверил: ассистент кафедры информатики Гриценко Н. Ю.

Минск 2023

Содержание

[1. Цель работы 2](#_Toc7030)

[2. Теория 4](#_Toc7031)

[3. Программа и комментарии 6](#_Toc7035)

4. Демонстрация работы 8

5. Демонстрация нахождения лексических ошибок 9

Вывод 11

Приложение 1. Таблица 12

Приложение 2. Текст программы 14

Приложение 3. Текст программ 20

# Цель работы

Освоение работы с существующими синтаксическими анализаторами.

Разработать свой собственный синтаксический анализатор, выбранного подмножества языка программирования. Построить синтаксическое дерево. Определить минимум 4 возможных синтаксических ошибки и показать их корректное выявление. Основной целью работы является написание сценариев, которые задают синтаксические правила для выбранного подмножества языка.

В качестве анализируемого подмножества языка программирования будет использован язык программирования Python. Для написания анализатора использован язык программирования C++.

# Теория

Синтаксический анализ — это процесс сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой. Результатом обычно является дерево разбора (синтаксическое дерево). Обычно применяется совместно с лексическим анализом. Синтаксический анализатор — это программа или часть программы, выполняющая синтаксический анализ. Задача синтаксического анализатора – проверить правильность записи выражения и разбить его на лексемы. Лексемой называется неделимая часть выражения, состоящая, в общем случае, из нескольких символов. Результатом синтаксического анализа является синтаксическое строение предложения, представленное либо в виде дерева зависимостей, либо в виде дерева составляющих, либо в виде некоторого сочетания первого и второго способов представления.

Типы алгоритмов:

● Нисходящий парсер — продукции грамматики раскрываются, начиная со стартового символа, до получения требуемой последовательности токенов, им соответствуют LL-грамматики;

● Восходящий парсер — продукции восстанавливаются из правых частей, начиная с токенов и кончая стартовым символом, им соответствуют LR-грамматики.

1. Программа и комментарии
2. Демонстрация работы

## Вывод

В результате выполнения лабораторной работы была изучена теория о синтаксических анализаторах, разработан собственный синтаксический анализатор, выбранного подмножества языка программирования, построить синтаксическое дерево. Определены минимум 4 возможных синтаксических ошибки и показано их корректное выявление.

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)