Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Лабораторная работа №1

по дисциплине

«Компиляторные технологии»

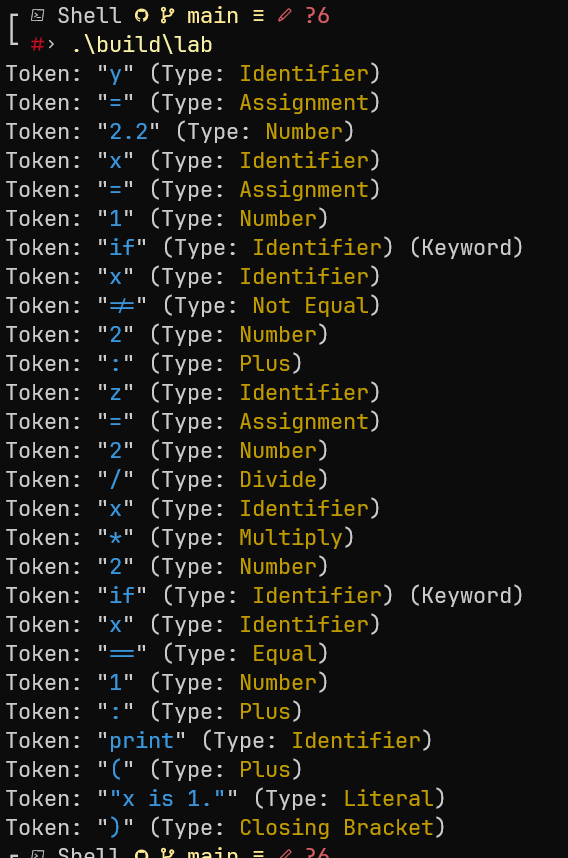
Вариант №5

|  |  |
| --- | --- |
|  | Выполнил:  студент группы 281073  Буйновский Михаил Валерьевич |
|  | Проверил:  Преподаватель Середа Ирина Александрона |

Минск 2023

Задание 1.

Разработай синтаксический анализатор, основанного на методе рекурсивного спуска для языка Python.



#pragma once

#include <string>

#include <unordered\_set>

const std::unordered\_set<std::string> keywords = {

"and", "as", "assert", "break", "class", "continue", "def", "del", "elif", "else", "except",

"false", "finally", "for", "from", "global", "if", "import", "in", "is", "lambda", "none",

"nonlocal", "not", "or", "pass", "raise", "return", "true", "try", "while", "with", "yield",

};

#pragma once

#include <cctype>

#include <string>

#include <unordered\_set>

enum class TokenType

{

None,

OperatorPlus,

OperatorMultiply,

OperatorDivide,

OperatorSubtract,

OperatorEqual,

OperatorNotEqual,

OperatorGreater,

OperatorLess,

OperatorAssignment,

Identifier,

Number,

Literal,

Colon,

BracketOpen,

BracketClose

};

struct Token

{

TokenType type;

std::string lexeme;

};

class Lexer

{

public:

Lexer(const std::string &code): code\_(code), position\_(0) {}

Token getNextToken()

{

Token token;

token.type = TokenType::None;

while (position\_ < code\_.length()) {

char currentChar = code\_[position\_];

if (position\_ + 1 < code\_.length()) {

std::string twoChars = std::string() + code\_[position\_] + code\_[position\_ + 1];

if (twoChars == "==") {

token.type = TokenType::OperatorEqual;

token.lexeme = "==";

position\_ += 2;

break;

}

if (twoChars == "!=") {

token.type = TokenType::OperatorNotEqual;

token.lexeme = "!=";

position\_ += 2;

break;

}

}

if (currentChar == ')') {

token.type = TokenType::BracketClose;

token.lexeme = ")";

position\_++;

break;

}

if (currentChar == '(') {

token.type = TokenType::OperatorPlus;

token.lexeme = "(";

position\_++;

break;

}

if (currentChar == '+') {

token.type = TokenType::OperatorPlus;

token.lexeme = "+";

position\_++;

break;

}

if (currentChar == '-') {

token.type = TokenType::OperatorSubtract;

token.lexeme = "-";

position\_++;

break;

}

if (currentChar == '\*') {

token.type = TokenType::OperatorMultiply;

token.lexeme = "\*";

position\_++;

break;

}

if (currentChar == '/') {

token.type = TokenType::OperatorDivide;

token.lexeme = "/";

position\_++;

break;

}

if (currentChar == ':') {

token.type = TokenType::OperatorPlus;

token.lexeme = ":";

position\_++;

break;

}

if (currentChar == '=') {

token.type = TokenType::OperatorAssignment;

token.lexeme = "=";

position\_++;

break;

}

if (currentChar == '>') {

token.type = TokenType::OperatorAssignment;

token.lexeme = "=";

position\_++;

break;

}

if (currentChar == '<') {

token.type = TokenType::OperatorAssignment;

token.lexeme = "=";

position\_++;

break;

}

if (std::isalpha(currentChar)) {

token.type = TokenType::Identifier;

token.lexeme += currentChar;

position\_++;

while (position\_ < code\_.length() && std::isalnum(code\_[position\_])) {

token.lexeme += code\_[position\_];

position\_++;

}

if (isKeyword(token.lexeme)) {

token.type = TokenType::Identifier;

token.lexeme = token.lexeme;

}

break;

}

if (std::isdigit(currentChar)) {

token.type = TokenType::Number;

token.lexeme += currentChar;

position\_++;

while (position\_ < code\_.length() && (std::isdigit(code\_[position\_]) || code\_[position\_] == '.')) {

token.lexeme += code\_[position\_];

position\_++;

}

break;

}

if (currentChar == '\'' || currentChar == '"') {

char \_char = currentChar;

token.type = TokenType::Literal;

token.lexeme += currentChar;

position\_++;

while (position\_ < code\_.length() && (code\_[position\_] != \_char)) {

token.lexeme += code\_[position\_];

position\_++;

}

token.lexeme += \_char;

position\_++;

break;

}

position\_++;

}

return token;

}

bool isKeyword(const std::string &lexeme)

{

return keywords\_.count(lexeme) > 0;

}

private:

std::string code\_;

size\_t position\_;

std::unordered\_set<std::string> keywords\_;

};

#include "keywords.hpp"

#include "lexer.hpp"

#include <iostream>

#include <string>

#include <unordered\_set>

int main()

{

std::string code =

"y = 2.2"

"x = 1"

"if x != 2:"

" z = 2 / x \* 2"

"if x == 1:"

" print(\"x is 1.\")";

Lexer lexer(code);

Token token = lexer.getNextToken();

while (token.type != TokenType::None) {

std::string tokenType;

switch (token.type) {

case TokenType::None:

tokenType = "Unknown";

break;

case TokenType::OperatorPlus:

tokenType = "Plus";

break;

case TokenType::OperatorMultiply:

tokenType = "Multiply";

break;

case TokenType::OperatorDivide:

tokenType = "Divide";

break;

case TokenType::OperatorSubtract:

tokenType = "Subtract";

break;

case TokenType::OperatorEqual:

tokenType = "Equal";

break;

case TokenType::OperatorNotEqual:

tokenType = "Not Equal";

break;

case TokenType::OperatorGreater:

tokenType = "Greater";

break;

case TokenType::OperatorLess:

tokenType = "Less";

break;

case TokenType::OperatorAssignment:

tokenType = "Assignment";

break;

case TokenType::Identifier:

tokenType = "Identifier";

break;

case TokenType::Number:

tokenType = "Number";

break;

case TokenType::Literal:

tokenType = "Literal";

break;

case TokenType::Colon:

tokenType = "Colon";

break;

case TokenType::BracketOpen:

tokenType = "Open Bracket";

break;

case TokenType::BracketClose:

tokenType = "Closing Bracket";

break;

}

std::string isKeyword =

(token.type == TokenType::Identifier && keywords.count(token.lexeme) > 0) ? " (Keyword)" : "";

std::cout << "Token: \""

<< "\033[36m" << token.lexeme << "\033[0m"

<< "\" (Type: "

<< "\033[33m" << tokenType << "\033[0m"

<< ")" << isKeyword << std::endl;

token = lexer.getNextToken();

}

return 0;

}