Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа № 2

По дисциплине ЯПИС

За шестой семестр

Тема: «Проектирование лексического анализатора»

Вариант 8

Выполнил:

Студент 3 курса

Группы ИИ-16(2)

Пилипук

Проверил:

Слинко

Брест, 2021

*Цель работы:* изучение основных понятий теории регулярных грамматик, ознакомление с назначением и принципами работы лексических анализаторов (сканеров), получение практических навыков построения сканера на примере заданного простейшего входного языка.

Вариант 8: Входной язык содержит последовательность вызовов процедур, разделенных символом ;(точка с запятой). Вызов процедуры должен состоять из имени процедуры и списка параметров. В качестве параметров могут выступать идентификаторы и римские цифры со знаком. (Римскими считать числа записанные большими буквами X, V и I)

#include <fstream>

#include <string>

#include <vector>

#include <map>

#include <iostream>

#include <algorithm>

class Analyst

{

public:

Analyst() {};

Analyst(const std::string& str) : text(str) {};

~Analyst() {};

// methods

void setText(const std::string& str)

{

variables.clear();

text = str;

}

void analyse()

{

std::vector<std::vector<std::string>> data = parseText(text);

for (size\_t i = 0; i < data.size(); i++)

{

for (size\_t j = 0; j < data[i].size(); j++)

{

if (data[i][j].find("#name") != std::string::npos)

{

data[i][j].erase(data[i][j].find("#name"));

variables.push\_back(std::make\_pair(data[i][j], type::PROC\_NAME));

continue;

}

if (checkRoman(data[i][j]))

{

variables.push\_back(std::make\_pair(data[i][j], type::ROMAN\_NUM));

continue;

}

variables.push\_back(std::make\_pair(data[i][j], type::ID));

}

}

}

void save()

{

unsigned int countId = 1;

std::map<std::string, unsigned int> data;

std::ofstream file("result.xls");

file << "Lexeme\t" << "Type\t" << "Value\r";

for (const auto& x : variables)

{

file << x.first << "\t";

switch (x.second)

{

case 1:

{

file << "PROCEDURE NAME\r";

break;

}

case 2:

{

file << "ROMAN NUMERAL\t" << x.first << '\r';

break;

}

case 3:

{

if (data.find(x.first) == data.end())

{

data.insert(std::make\_pair(x.first, countId++));

}

unsigned int localId = (\*data.find(x.first)).second;

file << "IDENTIFIER\t" << x.first << " : " << std::to\_string(localId) << '\r';

break;

}

default:

break;

}

}

file.close();

}

private:

std::vector<std::vector<std::string>> parseText(std::string str)

{

std::vector<std::vector<std::string>> data;

int lastPos = 0;

int it = 0;

bool last = false;

if (std::count(str.begin(), str.end(), ';') == 0)

{

last = true;

}

while (!str.empty())

{

std::string proc = str.substr(0, str.find(';'));

if (proc.empty())

{

proc = str.substr(lastPos, str.size());

}

lastPos = str.find(lastPos, ';');

if (proc.empty())

{

return data;

}

std::string name = proc.substr(0, proc.find('('));

proc.erase(proc.begin(), proc.begin() + proc.find('(') + 1);

proc.pop\_back();

std::vector<std::string> params = parseParameters(proc);

data.push\_back(std::vector<std::string>());

data[it].push\_back(name + "#name");

for (const auto& p : params)

{

data[it].push\_back(p);

}

it++;

str.erase(0, str.find(';') + 1);

if (str.find(';') == std::string::npos)

{

if (last)

{

break;

}

last = true;

}

}

return data;

}

std::vector<std::string> parseParameters(std::string str)

{

std::vector<std::string> data;

while (!str.empty())

{

std::string temp = str.substr(0, str.find(','));

str.erase(0, str.find(',') + 1);

data.push\_back(temp);

if (str.find(',') == std::string::npos)

{

data.push\_back(str);

break;

}

}

return data;

}

bool checkRoman(const std::string& str)

{

for (size\_t i = 0; i < str.size(); i++)

{

if (str[i] != 'I' && str[i] != 'V' && str[i] != 'X' && str[i] != '+' && str[i] != '-')

{

return false;

}

}

return true;

}

enum type

{

PROC\_NAME = 1,

ROMAN\_NUM = 2,

ID = 3

};

std::string text;

std::vector<std::pair<std::string, type>> variables;

};

int main()

{

std::string text;

std::cout << "Enter text: ";

std::getline(std::cin, text);

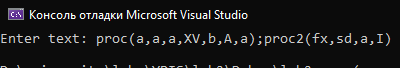
Analyst obj(text);

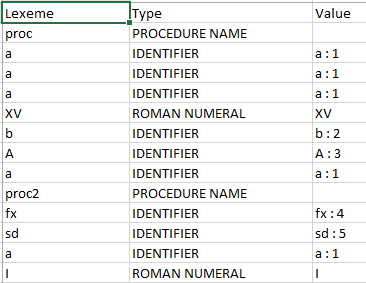
obj.analyse();

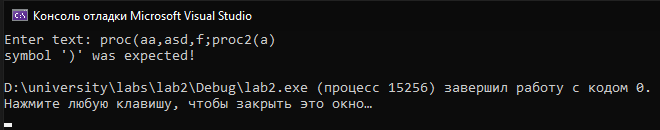
obj.save();

return 0;

}









Вывод: изучил основные понятия теории регулярных грамматик, реализовал сканер по заданию.