Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа № 3

По дисциплине ЯПИС

За шестой семестр

Тема: «**Проектирование лексического анализатора  
 с использованием регулярных выражений**

»

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>

#include <regex>

class Analyst

{

public:

Analyst() : checkRule(true) {};

Analyst(const std::string& str) : text(str), checkRule(true), regular((

"(\\w\*)"

"([\(])"

"([\\a-zIVX\,]+)"

"([\)])"

)) {};

~Analyst() {};

// methods

void setText(const std::string& str)

{

variables.clear();

text = str;

}

void analyse()

{

if (!checkRulesAndParse())

{

checkRule = false;

return;

}

}

void save()

{

if (!checkRule)

{

std::cout << "[INFO] Error is in expression!\n";

return;

}

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();

std::cout << "[INFO] File \"result.xls\" was successfully generated!\n";

}

private:

bool checkRulesAndParse()

{

std::vector<std::string> procs;

while (std::count(text.begin(), text.end(), ';') != 0)

{

size\_t pos = text.find(';');

std::string temp = text.substr(0, pos);

text.erase(0, pos + 1);

procs.push\_back(temp);

}

procs.push\_back(text);

std::cmatch res;

std::cout << "\n\n";

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

{

if (std::regex\_match(procs[i].c\_str(), res, regular))

{

variables.push\_back(std::make\_pair(res[1], type::PROC\_NAME));

std::vector<std::string> params;

std::string strParams = res[3];

size\_t pos = 0;

while ((pos = strParams.find(',')) != std::string::npos) {

std::string token = strParams.substr(0, pos);

params.push\_back(token);

strParams.erase(0, pos + 1);

}

params.push\_back(strParams);

for (size\_t j = 0; j < params.size(); j++)

{

if (islower(params[j][0]))

{

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

}

else

{

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

}

}

}

else

{

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;

std::regex regular;

bool checkRule;

};

int main()

{

std::string text = "";

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

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

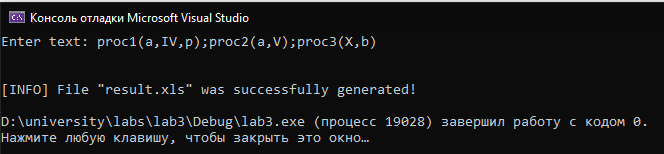
Analyst obj(text);

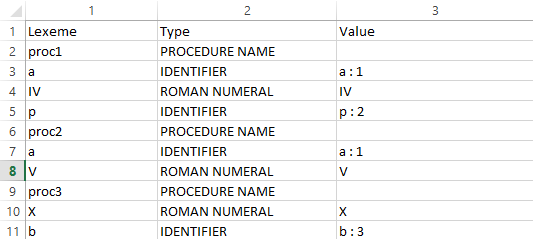
obj.analyse();

obj.save();

return 0;

}





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