Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Теория автоматов и формальных языков

ЛАБОРАТОРНАЯ РАБОТА № 3-4

Тема: **Распознавание заданных слов языка программирования**

Выполнил:

студент гр. 1ПИб-02-2оп-23

Кринкин Олег Алексеевич

Проверил:

доцент, к.т.н. Ганичева Оксана Георгиевна

ЗАДАНИЕ

1. Построить синтаксические диаграммы  (конечный автомат) для распознавания заданных слов языка программирования для варианта:

* Константы с плавающей точкой и ключевые слова;

2. Написать программу для распознавания   слов языка С++  по своему варианту задания (ключевые слова, идентификаторы, знаки операций, знаки пунктуации, целые константы, вещественные константы, числа с фиксированной и плавающей точкой...).  **Правильные слова должны заноситься в таблицу. На словах с ошибкой - выдаваться сообщение об ошибке.**

**Распознавание лексем выполняется по конечному автомату!!!**

3. Составить блок-схему алгоритма решения задачи.

4. Оформить отчет (задание, блок-схема алгоритма , код программы с комментариями, результаты тестирования программы (не менее 10 тестовых заданий), скриншоты результатов)

ХОД РАБОТЫ

1. В рамках лабораторной работы были выбраны следующие ключевые слова и варианты чисел с плавающей точкой:

* auto, bool, break, case, char, class, delete, do, double, else, false, for, if, int, private, public, return, this, true, void
* 15.75, 1.575E1, 1575e-2, -2.5e-3, 25E-4, +3.4, +6.5E+2

1. Для распознавания ключевых слов языка программирования был построен следующий конечный автомат (рис. 1):

Изображение выглядит как текст, снимок экрана, документ, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 1. Граф автомата ключевых слов

1. Для определения констант построен следующий автомат (рис. 2):

Изображение выглядит как диаграмма, линия, круг

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 2. Граф автомата констант

1. В автоматах зарезервированы 6 первых состояний для определения класса лексемы:

* 1 – ключевое слово;
* 2 – константа;
* 3 – идентификатор;
* 4 – знак отношения;
* 5 – знак операции;
* 6 – разделитель.

1. Если автомат перешёл в одно из этих состояний, то лексема возвращается и записывается в таблицу для своего класса (рис. 3).
2. Определение границ лексем происходит ожиданием следующим символом разделителя: «если следующий символ – разделитель, то лексема закончилась».

РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ РАБОТЫ

В результате выполнения лабораторной работы получено представление одной из частей лексического анализатора. Для её реализации был составлен конечный автомат, который обрабатывает символы текстов программы, отделяет в них лексемы, классифицирует их и записывает в таблицу лексем. Функция анализатора реализована в виде графической программы (рис. 5), способной открывать и обрабатывать текстовые файлы.

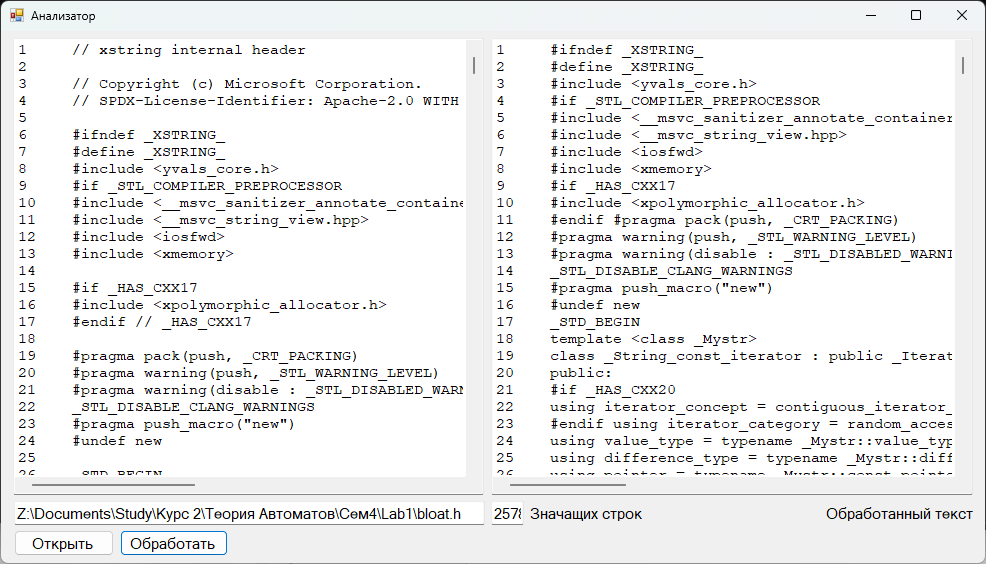


Рис. 3. Окно программы

Часть программного кода спроектированной программы:

private:

// Функция возвращающая некоторое количество пробелов

String^ spaces(int count) {

String^ spaces = gcnew String("");

for (int i = 0; i < count; i++)

spaces += " ";

return spaces;

}

// Функция-счётчик строк

int line = 0;

int getLine() {

line++;

return line;

}

// Функция добавляющая в ноер строки в исходный текст

String^ addLineNum(String^ target, int count) {

line = 0;

count = (count+1).ToString()->Length;

target = getLine() + spaces(count-1) + " " + target;

String^ out = gcnew String("");

for (int i = 0; i < target->Length; i++) {

if (target[i] == '\n') {

int line = getLine();

out += "\n" + line + spaces(count-(line.ToString()->Length)) + " ";

continue;

}

out += target[i];

}

target = out;

return target;

}

// Функция преобразующая System::String в std::string

std::string to\_string(String^ string) {

msclr::interop::marshal\_context context;

return context.marshal\_as<std::string>(string);

}

// Обработка нажатия на кнопку "Отрыть файл"

System::Void openFileButton\_Click(System::Object^ sender, System::EventArgs^ e) {

this->openFileDialog1->FileName = "";

this->openFileDialog1->ShowDialog();

this->fileNameBox->Text = openFileDialog1->FileName;

std::string& source = analyser->readSource(to\_string(openFileDialog1->FileName));

this->sourceBox->Text = gcnew String(source.c\_str());

this->sourceBox->Text = addLineNum(this->sourceBox->Text, analyser->getLineCount(source));

this->sourceBox->Text = this->sourceBox->Text->Replace("\n", "\r\n");

this->sourceBox->BackColor = System::Drawing::SystemColors::Window;

}

// Обработка нажатия на кнопку "Обработать"

System::Void processButton\_Click(System::Object^ sender, System::EventArgs^ e) {

this->outBox->BackColor = System::Drawing::SystemColors::Window;

this->outBox->Text = gcnew String(analyser->stripSource().c\_str());

this->outBox->Text = addLineNum(this->outBox->Text, analyser->getLineCount(analyser->stripped));

this->outBox->Text = this->outBox->Text->Replace("\n", "\r\n");

this->linesNum->Text = analyser->getLineCount(analyser->stripped).ToString();

}

};

class Analyser {

public:

std::string source;

std::string stripped;

int state;

Analyser() {

state = 0;

}

// Чтение файла

std::string& readSource(std::string fileName) {

std::ifstream file(fileName);

std::string line;

this->source = "";

while (std::getline(file, line)) {

this->source += line + '\n';

}

file.close();

return this->source;

}

// Функция, обрабатывающая текст и оправляющая символы анализатору

std::string& stripSource() {

this->stripped = "";

for (int i = 0; i < source.size(); i++) {

char out = filter(source[i]);

if (out != -1) this->stripped += out;

}

return this->stripped;

}

// Функция, возвращающая количество строк в тексте

int getLineCount(std::string text) {

int res = 0;

for (int i = 0; i < text.length() ;i++)

if (text[i] == '\n')

res++;

return res;

}

private:

// Функция анализатора

char filter(char c) {

if (c == ' ') return -1;

switch (state) {

case 0: {

switch (c) {

case '/': { state = 1; break; }

case ' ': { state = 4; break; }

case '\n': { state = 4; break; }

default: { state = 0; break; }

} break;

}

case 1: {

switch (c) {

case '/': { state = 1; break; }

case '\*': { state = 2; break; }

case '\n': { state = 5; break; }

} break;

}

case 2: {

switch (c) {

case '\*': { state = 3; break; }

default: { state = 2; break; }

} break;

}

case 3: {

switch (c) {

case '/': { state = 5; break; }

default: { state = 2; break; }

} break;

}

case 4: {

switch (c) {

case '/': { state = 1; break; }

case '\n': { state = 5; break; }

case ' ': { state = 5; break; }

default: { state = 0; break; }

} break;

}

case 5: {

switch (c) {

case '/': { state = 1; break; }

case ' ': { state = 5; break; }

case '\n': { state = 5; break; }

default: { state = 0; break; }

} break;

}

}

// Печать обрабатываемых символов в консоль (отладка)

#ifdef \_DEBUG

if (c == '\n') std::cout << "\\n" << " " << state;

else if (c == '\r') std::cout << "\\r" << " " << state;

else std::cout << c << " " << state;

if (state == 0 || state == 4) {

std::cout << " - Valid";

};

std::cout << std::endl;

#endif

if (state == 0 || state == 4)

return c;

return -1;

}

};

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было выполнено построение конечного автомата, на основе которого была спроектирована функция лексического анализатора, которая разделяет лексемы и классифицирует их, добавляя в таблицу лексем.

Функция лексического анализатора оформлена в виде графического приложения, и её работа протестирована на некотором тексте программы. Входной текст программы представлял собой описание класса сортированного массива, а также работу с ним.

Конечный автомат, на котором была построена функция лексического анализатора состоял из 63-х состояний, среди которых 6 конечных для определения класса лексем.