***Коваленко Сергей***

***Группа ИВ-63***

***Кафедра ВТ***

***Факультет ИВТ***

***Операционные системы***

***Лабораторная работа № 1***

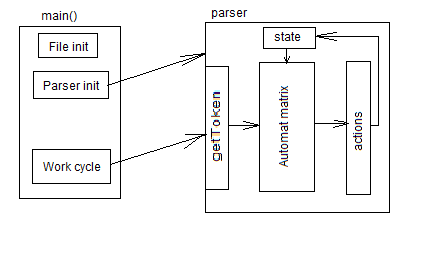
***Тема: простейший синтаксический анализатор (парсер).***

***Описание идеи разработанной инфраструктуры.***

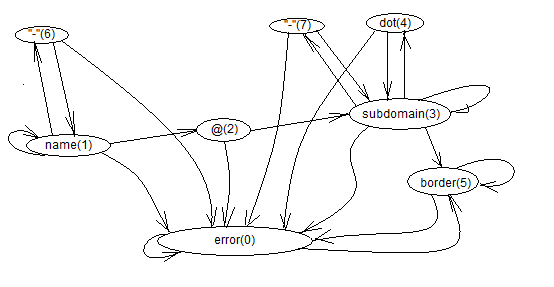
*Инфраструктура программных объектов являет собой усеченный вариант сервис-ориентированной инфраструктуры. Парсер выполнен в виде сервиса с простым интерфейсом, в который внесены методы для инициализации и получения структуры, содержащей структурированную информацию об адресе электронной почты. Из основной процедуры инициализируется дескриптор файла. Далее создается и инициализируется экземпляр сервиса. В него передается дескриптор файла. Последующая работа состоит в выполнении цикла получения очередной конструкции.*

*Сервис парсера содержит набор функций, реализующих действия автомата, дополнительные служебные процедуры и матрицу, описывающую зависимость выполняемых действий автомата от текущего состояния и класса очередного полученного символа. Таким образом, мы можем использовать данный сервис в многопоточных приложениях для параллельного разбора. Также в функции получения очередной структуры данных из парсера предусмотрена передача символьного буфера и первоочередная его обработка. Это сделано для более простого внедрения в дальнейшем дерева автоматов.*

***Графическое представление разработанной инфраструктуры***

******

***Графическое представление автомата.***

******

***Листинги программной реализации:***

#ifndef \_MAIL\_STRUCT

#define \_MAIL\_STRUCT

#include <stdio.h>

#include <STDLIB.H>

//#include <fstream>

#include <iostream>

using namespace std;

class subDomain

{

public:

subDomain\* next;

unsigned char size, first;

};

class MailStruct

{

friend class mailParser;

private:

subDomain\* subdmns;

unsigned char charIndex;

char dmns;

public:

char\* container;

unsigned char name;

MailStruct()

{

subdmns = NULL;

container = new char[255];

charIndex = 0;

dmns = 0;

}

void addChar(char c);

void output();

};

// mail parser is service

class mailParser

{

private:

FILE\* src;

unsigned char classes[256];

unsigned char automat[8][6];

unsigned char state;

MailStruct\* token;

bool going;

void (mailParser::\*actions[16])(char&);

//automat actions

void toErr(char &c);

void fromErr(char &c);

void borderCycle(char &c);

void toName(char &c);

void toNameDef(char &c);

void fromNameDef(char &c);

void nameCycle(char &c);

void endName(char &c);

void toSubdomain(char &c);

void toSubdomainDef(char &c);

void fromSubdomainDef(char &c);

void toDot(char &c);

void fromDot(char &c);

void subdomainCycle(char &c);

void endToken(char &c);

void errCycle(char &c);

public:

void getFile(FILE\* fsrc);

void parserInit();

void initClasses();

void initAutomat();

MailStruct\* getNextMail(int& currBufferLength, char\*& buffer);

};

#endif //\_MAIL\_STRUCT

#include "mailStruct.h"

using namespace std;

void MailStruct::addChar(char c)

{

container[charIndex] = c;

charIndex++;

}

void mailParser::getFile(FILE \*fsrc)

{

src = fsrc;

}

//automat actions

void mailParser::toErr(char &c)

{

state = 0;

if (token!=NULL)

delete token;

token = NULL;

}

void mailParser::fromErr(char &c)

{

state = 5;

going = false;

}

void mailParser::borderCycle(char &c)

{

}

void mailParser::toName(char &c)

{

state = 1;

token = new MailStruct();

token->addChar(c);

token->name = 1;

}

void mailParser::nameCycle(char &c)

{

token->name++;

token->addChar(c);

}

void mailParser::toNameDef(char &c)

{

token->name++;

token->addChar(c);

state = 6;

}

void mailParser::fromNameDef(char &c)

{

token->name++;

token->addChar(c);

state = 1;

}

void mailParser::endName(char &c)

{

state = 2;

token->addChar(c);

}

void mailParser::toSubdomain(char &c)

{

subDomain\* sd = new subDomain();

sd->next = token->subdmns;

token->subdmns = sd;

sd->first = token->charIndex;

sd->size = 1;

state = 3;

token->addChar(c);

token->dmns++;

}

void mailParser::toSubdomainDef(char &c)

{

state = 7;

token->addChar(c);

token->subdmns->size++;

//token->

}

void mailParser::fromSubdomainDef(char &c)

{

state = 3;

token->addChar(c);

token->subdmns->size++;

}

void mailParser::toDot(char &c)

{

state = 4;

token->addChar(c);

}

void mailParser::fromDot(char &c)

{

subDomain\* sd = new subDomain();

sd->next = token->subdmns;

token->subdmns = sd;

sd->first = token->charIndex;

sd->size = 1;

state = 3;

token->addChar(c);

token->dmns++;

}

void mailParser::subdomainCycle(char &c)

{

token->addChar(c);

token->subdmns->size++;

}

void mailParser::endToken(char &c)

{

going = false;

if (state != NULL)

token->addChar('\0');

state = 5;

}

void mailParser::errCycle(char &c)

{

//token->addChar(c);

}

//service functions

void mailParser::initClasses()

{

//prepare classes array

for (int i = 0; i < 256; i++)

classes[i] = 0;

// mark digits and letters as 1

for (int i = 48; i < 58; i++)

classes[i] = 1;

for (int i = 65; i < 91; i++)

classes[i] = 1;

for (int i = 97; i < 123; i++)

classes[i] = 1;

// mark "@" as 2

classes[64] = 2;

// mark "-" as 3

classes[45] = 3;

// mark "." as 4

classes[46] = 4;

// mark dividing symbols as 5

for (int i = 0; i < 33; i++)

classes[i] = 5;

}

void mailParser::initAutomat()

{

// init actions

actions[0] = &mailParser::toErr;

actions[1] = &mailParser::fromErr;

actions[2] = &mailParser::borderCycle;

actions[3] = &mailParser::toName;

actions[4] = &mailParser::nameCycle;

actions[5] = &mailParser::toNameDef;

actions[6] = &mailParser::fromNameDef;

actions[7] = &mailParser::endName;

actions[8] = &mailParser::toSubdomain;;

actions[9] = &mailParser::toSubdomainDef;

actions[10] = &mailParser::fromSubdomainDef;

actions[11] = &mailParser::toDot;

actions[12] = &mailParser::fromDot;

actions[13] = &mailParser::subdomainCycle;

actions[14] = &mailParser::endToken;

actions[15] = &mailParser::errCycle;

// init positions of automat

for (int i = 0; i < 8; i++)

for (int j = 0; j < 6; j++)

automat[i][j] = 0;

for (int i = 0; i < 5; i++)

automat[0][i] = 15;

automat[0][5] = 1;

automat[1][1] = 4; automat[1][2] = 7; automat[1][3] = 5;

automat[2][1] = 8;

automat[3][1] = 13; automat[3][3] = 9; automat[3][4] = 11; automat[3][5] = 14;

automat[4][1] = 12;

automat[5][1] = 3; automat[5][5] = 2;

automat[6][1] = 6;

automat[7][1] = 10;

}

void mailParser::parserInit()

{

initClasses();

state = 5;

going = false;

initAutomat();

token = NULL;

}

MailStruct\* mailParser::getNextMail(int& currBufferLength, char\*& buffer)

{

going = true;

char buf;

bool managed = false;

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

{

buf = buffer[i];

if (going && !managed)

{

(this->\*actions[automat [state] [classes [buf]]])(buf);

}

else if(!managed)

{

int diff = currBufferLength - i;

char\* newBuffer = new char[currBufferLength - i];

for (int j = 0; j < diff; j++)

newBuffer[j] = buffer[i + j];

delete[] buffer;

currBufferLength = diff;

buffer = newBuffer;

}

}

while (going && ((buf = (char)getc(src)) != EOF))

{

(this->\*actions[automat [state] [classes [buf]]])(buf);

}

return token;

}

void MailStruct::output()

{

// let it output itself

cout << "Mail token: \n";

cout << "name of box: ";

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

cout << container[i];

cout << "\n Subdomains: ";

subDomain\* tmp = NULL;

for (int i = dmns-1; i >= 0; i--)

{

tmp = subdmns;

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

tmp = tmp->next;

for (int k = tmp->first; k < (tmp->first + tmp->size); k++)

cout << container[k];

cout << " ";

}

cout << "\n";

}

***Листинг примера использования:***

#include <fstream>

#include <iostream>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include "mailStruct.h"

using namespace std;

int main()

{

cout << "Program for parsing list of emails \n";

//create file stream

char fileName[20];

cout << "Input name of target file\n";

cin >> fileName;

cout << "\n";

FILE \*fp;

if ((fp = fopen(fileName, "r")) == NULL)

{

cout << "Unopened file\n";

\_getch();

return 0;

}

mailParser parser = mailParser();

parser.getFile(fp);

parser.parserInit();

MailStruct\* token = NULL;

int length = 0;

char\* buf = NULL;

while (!(feof(fp)))

{

token = parser.getNextMail(length, buf);

if (token == NULL)

cout << "error in mailList!\n";

else

{

token->output();

delete token;

}

}

/\*int ch;

while ((ch = getc(fp)) != EOF)

{

cout << (char)ch;

}

cout << "End of file";//\*/

fclose(fp);

// get and output tokens in cycle

\_getch();

return 0;

}

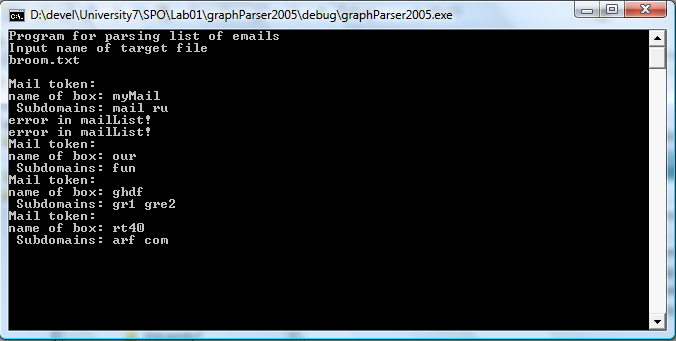
***Пример работы программы:***

myMail@mail.ru 12.thd &sdb our@fun

ghdf@gr1.gre2

rt40@arf.com

***Результат работы:***

******

***Парсинг программного синтаксиса.*** *Так же в данной работе требовалось построить дерево парсеров. Для этого дерева выделить 1 управляющий и дополнительные.*

*По структуре дополнительные парсеры являются компонентами основного. В обьекте основного парсера может быть развернут еще один обьект основного парсера. Это связано с тем, что требуется управлять вложенной секцией, возможно, с учетом контекста секции. Потому каждый экземпляр парсера секции хранит контекст закрепленной за ним сессии.*

***Программный код:***

#ifndef \_MAIN\_PARSER

#define \_MAIN\_PARSER

#include <stdio.h>

#include "MultiRowCP.h"

#include "NameParser.h"

#include "SingleRowCP.h"

#include "ValueParser.h"

class MainParser;

class MainParser

{

private:

static char letterClasses[256];

// parser actions

void error(char c);

void errorCycle(char c);

void fromError(char c);

void skipDividers(char c);

void singleRowComment(char c);

void slashFound(char c);

void multiRowComment(char c);

void nameFound(char c);

void equalFound(char c);

void endValue(char c);

void parametrFound(char c);

void subSection(char c);

void ruleLower(char c);

void endSection(char c);

// initialization

void letterClassesInit();

void functionsInit();

// state information

bool goMore;

static int rules[7][10];

char state;

char\* token;

unsigned int tokenIndex;

void (MainParser::\*actions[14])(char);

//context

char\*\* obuf;

int\* obufLen;

FILE\*\* osrc;

int\* opos;

public:

MainParser\* subParser;

MainParser\* curr;

MultiRowCommentParser mrcp;

NameParser np;

SingleRowCommentParser srcp;

ValueParser vp;

// payload

void initialize();

char\* getToken(char\* &buf, int &bufLen, FILE\* src, int &pos);

};

#endif //\_MAIN\_PARSER

#include "MainParser.h"

char MainParser::letterClasses[256];

int MainParser::rules[7][10];

//initialization

void MainParser::letterClassesInit()

{

for (int i = 0; i < 256; i++)

letterClasses[i] = 0;

for (int i = 48; i < 58; i++)

letterClasses[i] = 1;

for (int i = 65; i < 91; i++)

letterClasses[i] = 1;

for (int i = 97; i < 123; i++)

letterClasses[i] = 1;

letterClasses[' '] = 2;

letterClasses['\n'] = 2;

letterClasses['#'] = 3;

letterClasses['\*'] = 4;

letterClasses['/'] = 5;

letterClasses['='] = 6;

letterClasses[';'] = 7;

letterClasses['{'] = 8;

letterClasses['}'] = 9;

}

void MainParser::functionsInit()

{

actions[0] = &MainParser::error;

actions[1] = &MainParser::errorCycle;

actions[2] = &MainParser::fromError;

actions[3] = &MainParser::skipDividers;

actions[4] = &MainParser::singleRowComment;

actions[5] = &MainParser::slashFound;

actions[6] = &MainParser::multiRowComment;

actions[7] = &MainParser::nameFound;

actions[8] = &MainParser::equalFound;

actions[9] = &MainParser::endValue;

actions[10] = &MainParser::parametrFound;

actions[11] = &MainParser::subSection;

actions[12] = &MainParser::ruleLower;

actions[13] = &MainParser::endSection;

rules[0][0] = 1; rules[0][1] = 1; rules[0][2] = 2; rules[0][3] = 4; rules[0][4] = 1;

rules[0][5] = 5; rules[0][6] = 1; rules[0][7] = 1; rules[0][8] = 1; rules[0][9] = 13;

rules[1][0] = 0; rules[1][1] = 7; rules[1][2] = 3; rules[1][3] = 4; rules[1][4] = 0;

rules[1][5] = 5; rules[1][6] = 0; rules[1][7] = 0; rules[1][8] = 1; rules[1][9] = 13;

rules[2][0] = 0; rules[2][1] = 0; rules[2][2] = 0; rules[2][3] = 0; rules[2][4] = 6;

rules[2][5] = 0; rules[2][6] = 0; rules[2][7] = 0; rules[2][8] = 0; rules[2][9] = 0;

rules[3][0] = 0; rules[3][1] = 10; rules[3][2] = 3; rules[3][3] = 4; rules[3][4] = 0;

rules[3][5] = 5; rules[3][6] = 8; rules[3][7] = 0; rules[3][8] = 11; rules[3][9] = 0;

rules[4][0] = 0; rules[4][1] = 0; rules[4][2] = 0; rules[4][3] = 0; rules[4][4] = 0;

rules[4][5] = 0; rules[4][6] = 0; rules[4][7] = 9; rules[4][8] = 0; rules[4][9] = 13;

rules[5][0] = 0; rules[5][1] = 0; rules[5][2] = 3; rules[5][3] = 4; rules[5][4] = 0;

rules[5][5] = 5; rules[5][6] = 0; rules[5][7] = 0; rules[5][8] = 11; rules[5][9] = 0;

rules[6][0] = 12; rules[6][1] = 12; rules[6][2] = 12; rules[6][3] = 12; rules[6][4] = 12;

rules[6][5] = 12; rules[6][6] = 12; rules[6][7] = 12; rules[6][8] = 12; rules[6][9] = 13;

subParser = NULL;

mrcp = MultiRowCommentParser();

mrcp.initialize();

np = NameParser();

np.initialize();

srcp = SingleRowCommentParser();

srcp.initialize();

vp = ValueParser();

vp.initialize();

}

void MainParser::initialize()

{

letterClassesInit();

functionsInit();

}

// parser actions

void MainParser::error(char c)

{

delete[] token;

tokenIndex = 0;

state = 0;

}

void MainParser::errorCycle(char c)

{

}

void MainParser::fromError(char c)

{

state = 1;

goMore = false;

}

void MainParser::skipDividers(char c)

{

}

void MainParser::singleRowComment(char c)

{

delete[] token;

if (\*obuf != NULL)

\*opos = \*opos - 1;

else

{

obuf = &(new char[1]);

(\*obuf)[1] = '#';

\*obufLen = 1;

\*opos = 0;

}

token = srcp.getToken(\*obuf, \*obufLen, \*osrc, \*opos);

tokenIndex = strlen(token);

goMore = false;

}

void MainParser::slashFound(char c)

{

state = 2;

token[tokenIndex++] = c;

}

void MainParser::multiRowComment(char c)

{

delete[] token;

if (\*opos > 1)

\*opos = \*opos - 2;

else

{

obuf = &(new char[2]);

(\*obuf)[0] = '/';

(\*obuf)[1] = '\*';

\*obufLen = 2;

\*opos = 0;

}

token = mrcp.getToken(\*obuf, \*obufLen, \*osrc, \*opos);

tokenIndex = strlen(token);

goMore = false;

}

void MainParser::nameFound(char c)

{

delete[] token;

if (\*obuf != NULL)

\*opos = \*opos - 1;

else

{

obuf = &(new char[1]);

(\*obuf)[0] = c;

\*obufLen = 1;

\*opos = 0;

}

token = np.getToken(\*obuf, \*obufLen, \*osrc, \*opos);

tokenIndex = strlen(token);

goMore = false;

state = 3;

}

void MainParser::equalFound(char c)

{

delete[] token;

token = vp.getToken (\*obuf, \*obufLen, \*osrc, \*opos);

tokenIndex = strlen(token);

goMore = false;

state = 4;

}

void MainParser::endValue(char c)

{

state = 1;

}

void MainParser::parametrFound(char c)

{

if (\*obuf != NULL)

\*opos = \*opos - 1;

else

{

obuf = &(new char[1]);

(\*obuf)[0] = c;

\*obufLen = 1;

\*opos = 0;

}

delete[] token;

token = vp.getToken (\*obuf, \*obufLen, \*osrc, \*opos);

tokenIndex = strlen(token);

goMore = false;

state = 5;

}

void MainParser::subSection(char c)

{

if (subParser == NULL)

{

subParser = new MainParser();

subParser->functionsInit();

}

else

subParser->subSection(c);

state = 6;

}

void MainParser::ruleLower(char c)

{

if (\*obuf != NULL)

\*opos = \*opos - 1;

else

{

obuf = &(new char[1]);

(\*obuf)[0] = c;

\*obufLen = 1;

\*opos = 0;

}

delete[] token;

MainParser \*t = new MainParser();

while (t->subParser != NULL)

t = t->subParser;

token = t->getToken (\*obuf, \*obufLen, \*osrc, \*opos);

tokenIndex = strlen(token);

goMore = false;

}

void MainParser::endSection(char c)

{

if ((subParser != NULL) && (subParser->subParser == NULL))

{

delete subParser;

subParser = NULL;

}

else

if (subParser != NULL)

subParser->endSection(c);

}

char\* MainParser::getToken(char \*&buf, int &bufLen, FILE \*src, int &pos)

{

cout << "Global dj's parsing... \n";

// init parser

state = 1;

goMore = true;

token = new char[250];

tokenIndex = 0;

// copy context

\*obuf = buf;

\*obufLen = bufLen;

\*osrc = src;

\*opos = pos;

// working cycle

char c;

for (; ((buf != NULL) && (pos < bufLen) && goMore); pos++)

{

c = buf[pos];

(this->\*actions[rules[state][letterClasses[c]]])(c);

}

if (goMore)

{

pos = 0;

delete[] buf;

buf = NULL;

bufLen = 0;

// continue cycle

while (goMore && ((c = (char)getc(src)) != EOF))

{

(this->\*actions[rules[state][letterClasses[c]]])(c);

}

}

if (token != NULL)

token[tokenIndex++] = 0;

// return result

return token;

}

***Файл с основной процедурой:***

#include <iostream>

#include <conio.h>

#include "SingleRowCP.h"

#include "ValueParser.h"

#include "NameParser.h"

#include "MultiRowCP.h"

#include "MainParser.h"

using namespace std;

//#define dbg

int main()

{

cout << "Initializing parser...\n";

MainParser mp = MainParser();

mp.initialize();

int pos = 0;

cout << "Input name of file: \n";

char fileName[20];

cin >> fileName;

cout << "\n";

FILE \*fp;

if ((fp = fopen(fileName, "r")) == NULL)

{

cout << "Unopened file\n";

\_getch();

return 0;

}

char\* buf = NULL;

int length = 0;

while (!(feof(fp)))

{

char\* token = mp.getToken(buf, length, fp, pos);

if (token == NULL)

cout << "error happened!\n";

else

cout << token << "\n";

}

fclose(fp);

\_getch();

return 0;

}