Я разрабатываю свой компилятор для своего языка программирования. Сейчас я на этапе создания лексического анализатора, но программа почему-то завершается без какого-либо вывода в консоль. Это происходит при вызове функции FillingInTables в LPA-2024.cpp в main. Я не знаю почему это происходит помоги решить эту проблему. Резульатом выполнения программы должна быть запись в файл таблицы идетнификаторов и таблицы лексем.

Вот мой код состоящий из множества файлов:

Error.cpp:

#include "stdafx.h"

#include "Error.h"

#include <iostream>

#include <cstring>

namespace Error

{

// 0-99 Cистемные ошибки

// 100-109 Ошибки параметров

// 110-119 Ошибки открытия и чтения файлов

// 120-129 Ошибки заполнения таблиц Лексем и Идентификаторов

// 130-139 Ошибки заполнения таблицы токенов

// Таблица ошибок

ERROR errors[ERROR\_MAX\_ENTRY] =

{

// Код ошибки вне диапазона 0 - ERROR\_MAX\_ENTRY

ERROR\_ENTRY(0,"Системная ошибка: Недопустимый код ошибки"),

ERROR\_ENTRY(1,"Системная ошибка: Системный сбой"),

ERROR\_ENTRY\_NODEF(2), ERROR\_ENTRY\_NODEF(3),

ERROR\_ENTRY\_NODEF(4), ERROR\_ENTRY\_NODEF(5),

ERROR\_ENTRY\_NODEF(6), ERROR\_ENTRY\_NODEF(7),ERROR\_ENTRY\_NODEF(8),ERROR\_ENTRY\_NODEF(9),

ERROR\_ENTRY(10, "Ошибка параметра: Параметр -in должен быть задан"),

ERROR\_ENTRY(11, "Ошибка параметра: Превышена длина входного параметра"),

ERROR\_ENTRY(12, "Ошибка параметра: Параметр -out должен быть задан"),

ERROR\_ENTRY\_NODEF(13),

ERROR\_ENTRY\_NODEF(14), ERROR\_ENTRY\_NODEF(15), ERROR\_ENTRY\_NODEF(16), ERROR\_ENTRY\_NODEF(17),

ERROR\_ENTRY\_NODEF(18), ERROR\_ENTRY\_NODEF(19),

ERROR\_ENTRY(20, "Ошибка файла: Ошибка при открытии файла с исходным кодом (-in:)"),

ERROR\_ENTRY(21, "Ошибка файла: Ошибка при создании файла протокола (-log:)"),

ERROR\_ENTRY(22, "Ошибка файла: Ошибка при открытии файла с таблицей лексем (-lex:)"),

ERROR\_ENTRY(23, "Ошибка файла: Ошибка при открытии файла с таблицей идентификаторов (-id:)"),

ERROR\_ENTRY\_NODEF(24), ERROR\_ENTRY\_NODEF(25), ERROR\_ENTRY\_NODEF(26),

ERROR\_ENTRY\_NODEF(27), ERROR\_ENTRY\_NODEF(28), ERROR\_ENTRY\_NODEF(29),

ERROR\_ENTRY\_NODEF10(30), ERROR\_ENTRY\_NODEF10(40), ERROR\_ENTRY\_NODEF10(50),

ERROR\_ENTRY\_NODEF10(60), ERROR\_ENTRY\_NODEF10(70), ERROR\_ENTRY\_NODEF10(80), ERROR\_ENTRY\_NODEF10(90),

ERROR\_ENTRY\_NODEF10(100),

ERROR\_ENTRY(110, "Ошибка лексического анализатора: Недопустимый символ в исходном файле (-in:)"),

ERROR\_ENTRY(111, "Ошибка лексического анализатора: Превышена емкость таблицы лексем"),

ERROR\_ENTRY(112, "Ошибка лексического анализатора: Превышено количество строк в таблице лексем"),

ERROR\_ENTRY(113, "Ошибка лексического анализатора: В таблице лексем отсутствует строка с заданным номером"),

ERROR\_ENTRY(114, "Ошибка лексического анализатора: Превышена емкость таблицы идентификаторов"),

ERROR\_ENTRY(115, "Ошибка лексического анализатора: Превышено количество строк в таблице идентификаторов"),

ERROR\_ENTRY(116, "Ошибка лексического анализатора: В таблице идентификаторов отсутствует строка с заданным номером"),

ERROR\_ENTRY(117, "Ошибка лексического анализатора: Не удалось определить тип лексемы"),

ERROR\_ENTRY(118, "Ошибка лексического анализатора: Превышена емкость таблицы токенов"),

ERROR\_ENTRY(119, "Ошибка лексического анализатора: Превышено количество токенов в таблице токенов"),

ERROR\_ENTRY(120, "Ошибка лексического анализатора: Ошибка с разбиением исходного текста на токены"),

ERROR\_ENTRY(121, "Ошибка лексического анализатора: Ошибка с разбором строкового литерала"),

ERROR\_ENTRY\_NODEF(122), ERROR\_ENTRY\_NODEF(123), ERROR\_ENTRY\_NODEF(124), ERROR\_ENTRY\_NODEF(125),

ERROR\_ENTRY\_NODEF(126), ERROR\_ENTRY\_NODEF(127), ERROR\_ENTRY\_NODEF(128), ERROR\_ENTRY\_NODEF(129),

ERROR\_ENTRY(130, "Ошибка семантического анализа: Превышена длина строки в 255 символов"),

ERROR\_ENTRY(131, "Ошибка семантического анализа: Функция main уже имеет реализацию"),

ERROR\_ENTRY(132, "Ошибка семантического анализа: Превышена длина лексемы"),

ERROR\_ENTRY(133, "Ошибка семантического анализа: Превышено значение целочисленного литерала (2 byte)"),

ERROR\_ENTRY(134, "Ошибка семантического анализа: Не найдена точка входа в программу (main)"),

ERROR\_ENTRY(135, "Ошибка семантического анализа: Идентификатор с таким именем не найден"),

ERROR\_ENTRY(136, "Ошибка семантического анализа: Повторное объявление идентификатора"),

ERROR\_ENTRY(137, "Ошибка семантического анализа: Несоответствие типов в выражении"),

ERROR\_ENTRY(138, "Ошибка семантического анализа: Слишком много параметров в функции"),

ERROR\_ENTRY(139, "Ошибка семантического анализа: Превышено количество функций"),

ERROR\_ENTRY(140, "Ошибка семантического анализа: Несоответствие параметров объявленной и вызываемой функций"),

ERROR\_ENTRY(141, "Ошибка семантического анализа: Несоответствие параметров встроенной функции"),

ERROR\_ENTRY(142, "Ошибка семантического анализа: Левостороннее выражение не является идентификатором и не должно являться функцией"),

ERROR\_ENTRY(143, "Ошибка семантического анализа: Данная функция уже имеет реализацию"),

ERROR\_ENTRY(144, "Ошибка семантического анализа: В вызове функции отсутствуют ()"),

ERROR\_ENTRY(145, "Ошибка семантического анализа: Тип возвращаемого значения не соответствует типу функции"),

ERROR\_ENTRY(146, "Ошибка семантического анализа: Оператор не предназначен для работы со строками"),

ERROR\_ENTRY(147, "Ошибка семантического анализа: В функции отсутствует возвращаемое значение"),

ERROR\_ENTRY(148, "Ошибка семантического анализа: Деление на 0 недопустимо"),

ERROR\_ENTRY(149, "Ошибка семантического анализа: Недопустимый строковый литерал"),

ERROR\_ENTRY\_NODEF10(150),

ERROR\_ENTRY\_NODEF10(160),ERROR\_ENTRY\_NODEF10(170),ERROR\_ENTRY\_NODEF10(180),ERROR\_ENTRY\_NODEF10(190),

ERROR\_ENTRY\_NODEF100(200),ERROR\_ENTRY\_NODEF100(300),ERROR\_ENTRY\_NODEF100(400),ERROR\_ENTRY\_NODEF100(500),

ERROR\_ENTRY(600, "Ошибка синтаксического анализа: Неверная структура программы"),

ERROR\_ENTRY(601, "Ошибка синтаксического анализа: Ошибка в конструкции блока кода"),

ERROR\_ENTRY(602, "Ошибка синтаксического анализа: Ошибка в выражении"),

ERROR\_ENTRY(603, "Ошибка синтаксического анализа: Ошибка в параметрах функции"),

ERROR\_ENTRY(604, "Ошибка синтаксического анализа: Ошибка в параметрах вызываемой функции"),

ERROR\_ENTRY(605, "Ошибка синтаксического анализа: Ошибочный оператор"),

ERROR\_ENTRY(606, "Ошибка синтаксического анализа: Ошибка в условной конструкции"),

ERROR\_ENTRY(607, "Ошибка синтаксического анализа: Ошибка типа"),

ERROR\_ENTRY\_NODEF(608),ERROR\_ENTRY\_NODEF(609),

ERROR\_ENTRY\_NODEF10(610), ERROR\_ENTRY\_NODEF10(620), ERROR\_ENTRY\_NODEF10(630), ERROR\_ENTRY\_NODEF10(640),

ERROR\_ENTRY\_NODEF10(660), ERROR\_ENTRY\_NODEF10(670), ERROR\_ENTRY\_NODEF10(680), ERROR\_ENTRY\_NODEF10(690),

ERROR\_ENTRY\_NODEF100(700),ERROR\_ENTRY\_NODEF100(800),ERROR\_ENTRY\_NODEF100(900),

};

ERROR geterror(int id)

{

ERROR error\_info;

if (id > 0 && id < ERROR\_MAX\_ENTRY)

{

error\_info.id = errors[id].id;

strcpy(error\_info.message, errors[id].message);

error\_info.inext.col = -1;

error\_info.inext.line = -1;

}

else

{

error\_info.id = errors[0].id;

strcpy(error\_info.message, errors[0].message);

}

return error\_info;

}

ERROR geterrorin(int id, int line, int col)

{

ERROR error\_info;

if (id > 0 && id < ERROR\_MAX\_ENTRY)

{

error\_info.id = errors[id].id;

error\_info.inext.col = col + 1;

error\_info.inext.line = line + 1;

strcpy(error\_info.message, errors[id].message);

return error\_info;

}

else

{

error\_info.id = 0;

error\_info.inext.col = -1;

error\_info.inext.line = -1;

strcpy(error\_info.message, errors[0].message);

return error\_info;

}

};

};

Error.h:

#pragma once

#define ERROR\_THROW(id) Error::geterror(id);

#define ERROR\_THROW\_IN(id,l,c) Error::geterrorin(id,l,c);

#define ERROR\_ENTRY(id, m) {id,m,{-1,-1}}

#define ERROR\_MAXSIZE\_MESSAGE 200

#define ERROR\_ENTRY\_NODEF(id) ERROR\_ENTRY(-id,"Неопределенная ошибка")

#define ERROR\_ENTRY\_NODEF10(id) ERROR\_ENTRY\_NODEF(id+0),ERROR\_ENTRY\_NODEF(id+1),ERROR\_ENTRY\_NODEF(id+2),ERROR\_ENTRY\_NODEF(id+3),\

ERROR\_ENTRY\_NODEF(id+4),ERROR\_ENTRY\_NODEF(id+5),ERROR\_ENTRY\_NODEF(id+6),ERROR\_ENTRY\_NODEF(id+7),\

ERROR\_ENTRY\_NODEF(id+8),ERROR\_ENTRY\_NODEF(id+9)

#define ERROR\_ENTRY\_NODEF100(id) ERROR\_ENTRY\_NODEF10(id+0),ERROR\_ENTRY\_NODEF10(id+10),ERROR\_ENTRY\_NODEF10(id+20),ERROR\_ENTRY\_NODEF10(id+30),\

ERROR\_ENTRY\_NODEF10(id+40),ERROR\_ENTRY\_NODEF10(id+50),ERROR\_ENTRY\_NODEF10(id+60),ERROR\_ENTRY\_NODEF10(id+70),\

ERROR\_ENTRY\_NODEF10(id+80),ERROR\_ENTRY\_NODEF10(id+90)

#define ERROR\_MAX\_ENTRY 1000

namespace Error

{

struct ERROR

{

int id;

char message[ERROR\_MAXSIZE\_MESSAGE];

struct IN

{

short line;

short col;

}inext;

};

ERROR geterror(int id);

ERROR geterrorin(int id, int line, int col);

};

FST.cpp:

#include "stdafx.h"

namespace FST

{

RELATION::RELATION(char c, short nn)

{

symbol = c;

nnode = nn;

};

NODE::NODE()

{

n\_relation = 0;

relations = nullptr;

};

NODE::NODE(short n, RELATION rel, ...)

{

n\_relation = n;

RELATION\* p = &rel;

relations = new RELATION[n];

for (short i = 0; i < n; i++) relations[i] = p[i];

};

FST::FST(char\* s, short ns, NODE n, ...)

{

string = s;

nstates = ns;

nodes = new NODE[ns];

NODE\* p = &n;

for (int k = 0; k < ns; k++)

nodes[k] = p[k];

rstates = new short[nstates];

memset(rstates, 0xff, sizeof(short) \* nstates);

rstates[0] = 0;

position = -1;

};

FST::FST(short ns, NODE n, ...)

{

nstates = ns;

nodes = new NODE[ns];

NODE\* p = &n;

for (int k = 0; k < ns; k++)

nodes[k] = p[k];

rstates = new short[nstates];

memset(rstates, 0xff, sizeof(short) \* nstates);

rstates[0] = 0;

position = -1;

}

FST::FST(char\* s, FST& fst)

{

this->nodes = new NODE[fst.nstates];

NODE\* temp = fst.nodes;

short i = 0;

this->string = s;

this->nstates = fst.nstates;

this->rstates = new short[this->nstates];

while (i < nstates)

{

this->nodes[i] = \*(temp + i);

i++;

};

rstates[0] = 0;

position = -1;

}

// Выполнение шага из одного массива в другой

bool step(FST& fst, short\*& rstates)

{

bool rc = false;

std::swap(rstates, fst.rstates);

for (short i = 0; i < fst.nstates; i++)

{

if (rstates[i] == fst.position)

for (short j = 0; j < fst.nodes[i].n\_relation; j++)

{

if (fst.nodes[i].relations[j].symbol == fst.string[fst.position])

{

fst.rstates[fst.nodes[i].relations[j].nnode] = fst.position + 1;

rc = true;

}

}

}

return rc;

};

// Выполнение распознавания цепочки с помощью алгоритма двух массивов

bool execute(FST& fst)

{

short\* rstates = new short[fst.nstates];

memset(rstates, 0xff, sizeof(short) \* fst.nstates);

short lstring = strlen(fst.string);

bool rc = true;

for (short i = 0; i < lstring && rc; i++)

{

fst.position++;

rc = step(fst, rstates);

}

delete[] rstates;

return (rc ? (fst.rstates[fst.nstates - 1] == lstring) : rc);

};

}

FST.h:

#pragma once

#include "stdafx.h"

// Конечный недетерминированный автомат предназначен для разбора цепочек символов в Graphs.h

namespace FST

{

struct RELATION // ребро:символ -> вершина графа переходов КА

{

char symbol; // символ перехода

short nnode; // номер смежной вершины

RELATION(

char c = 0x00, // символ перехода

short ns = NULL // новое состояние

);

};

struct NODE // вершина графа перехода

{

short n\_relation; // количество инцидентных ребер

RELATION\* relations; // инцидентные ребра

NODE();

NODE(

short n, // количество инцидентных ребер

RELATION rel, ... // список ребер

);

};

struct FST // недетерминированный конечный автомат

{

char\* string; // цепочка (строка, завершается 0x00)

short position; // текущая позиция в цепочке

short nstates; // количество состояний автомата

NODE\* nodes; // граф переходов: [0] -начальное состояние, [nstate - 1] -конечное

short\* rstates; // возможные состояния автомата на данной позиции

FST(

char\* s, // цепочка (строка, завершается 0x00)

short ns, // количество состояний автомата

NODE n, ... // список состояний (граф переходов)

);

FST(

short ns,

NODE n, ...

);

FST(

char\* s,

FST& fst

);

};

bool execute( // выполнить распознование цепочки

FST& fst // недетерминированный конечный автомат

);

}

In.cpp:

#include "stdafx.h"

#include "In.h"

#include "Error.h"

#include <fstream>

#include <string>

#include <iostream>

namespace In

{

unsigned char c = ':';

IN GetIn(wchar\_t infile[], wchar\_t outfile[])

{

int currentLine = 0, currentCol = 0;

IN Info{ 0, 0, 0, nullptr, IN\_CODE\_TABLE };

std::ifstream in(infile);

if (in.fail())

throw ERROR\_THROW(20);

std::string fulltext;

std::string temp;

while (!in.eof())

{

Info.lines++;

std::getline(in, temp);

temp += '\n';

fulltext += temp;

}

Info.text = new unsigned char[fulltext.size() + 1];

unsigned char\* Ptemp = Info.text;

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

{

if (Info.code[(unsigned char)fulltext[i]] == IN::F)

{

throw ERROR\_THROW\_IN(110, currentLine, currentCol)

}

else if (Info.code[(unsigned char)fulltext[i]] == IN::I)

++Info.ignor;

else if (Info.code[(unsigned char)fulltext[i]] == IN::T || Info.code[(unsigned char)fulltext[i]] == IN::S || Info.code[(unsigned char)fulltext[i]] == IN::P || Info.code[(unsigned char)fulltext[i]] == IN::N)

{

\*Ptemp = fulltext[i];

++Ptemp;

}

else

{

\*Ptemp = Info.code[(unsigned char)fulltext[i]];

++Ptemp;

}

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

{

++currentLine;

currentCol = 0;

}

else

++currentCol;

}

\*Ptemp = IN\_CODE\_ENDL;

Info.size = strlen((char\*)Info.text) - Info.lines;

in.close();

return Info;

}

}

In.h:

#pragma once

#define IN\_MAX\_LEN\_TEXT 1024\*1024

#define IN\_CODE\_ENDL '\0'

/\*

T - Разрешённые символы

F - Запрещённые символы

I - Игнорируемые символы

S - Символы-сепараторы

P - Пробел и табуляция

N - Переход на новую строку

\*/

#define IN\_CODE\_TABLE {\

/\*0\*/ /\*1\*/ /\*2\*/ /\*3\*/ /\*4\*/ /\*5\*/ /\*6\*/ /\*7\*/ /\*8\*/ /\*9\*/ /\*A\*/ /\*B\*/ /\*C\*/ /\*D\*/ /\*E\*/ /\*F\*/\

/\*0\*/ IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::P, IN::N, IN::F, IN::F, IN::I, IN::F, IN::F,\

/\*1\*/ IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

/\*2\*/ IN::P, IN::S, IN::T, IN::F, IN::F, IN::S, IN::F, IN::T, IN::S, IN::S, IN::S, IN::S, IN::S, IN::S, IN::T, IN::S,\

/\*3\*/ IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::S, IN::S, IN::S, IN::F,\

/\*4\*/ IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

/\*5\*/ IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F,\

/\*6\*/ IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

/\*7\*/ IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::F, IN::S, IN::F, IN::F,\

/\*0\*/ /\*1\*/ /\*2\*/ /\*3\*/ /\*4\*/ /\*5\*/ /\*6\*/ /\*7\*/ /\*8\*/ /\*9\*/ /\*A\*/ /\*B\*/ /\*C\*/ /\*D\*/ /\*E\*/ /\*F\*/\

/\*8\*/ IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

/\*9\*/ IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

/\*A\*/ IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

/\*B\*/ IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

/\*C\*/ IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

/\*D\*/ IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

/\*E\*/ IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

/\*F\*/ IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

}

namespace In

{

extern unsigned char c;

struct IN

{

enum { T = 1024, F = 2048, I = 4096, S = 8192, P = 16384, N = 32768 };

int size;

int lines;

int ignor;

unsigned char\* text;

int code[256];

};

// Функция вывода и проверки входного потока

IN GetIn(wchar\_t infile[], wchar\_t outfile[]);

}

IT.cpp:

#include "stdafx.h"

namespace IT

{

IdTable Create(int size)

{

if (size > TI\_MAXSIZE)

throw ERROR\_THROW(114);

IdTable idtable;

idtable.maxsize = size;

idtable.size = 0;

idtable.table = new Entry[idtable.maxsize];

return idtable;

}

void Add(IdTable& idtable, Entry entry)

{

if (idtable.size > idtable.maxsize)

throw ERROR\_THROW(115);

idtable.table[idtable.size] = entry;

idtable.size++;

}

Entry GetEntry(IdTable& idtable, int n)

{

if (n > (idtable.size - 1))

throw ERROR\_THROW(116);

return idtable.table[n];

}

int IsId(IdTable& idtable, std::string id)

{

if (idtable.size > 0)

{

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

{

if (idtable.table[i].id == id)

{

return i;

}

}

}

return TI\_NULLIDX;

}

void Delete(IdTable& idtable)

{

delete[] idtable.table;

}

Entry CreateEntry(int lineLT, std::string id, IDDATATYPE idDataType, IDTYPE idType)

{

Entry result;

result.idxfirstLE = lineLT;

result.id = id;

result.idDataType = idDataType;

result.idType = idType;

return result;

}

Entry CreateEntry(int lineLT, std::string id, IDDATATYPE idDataType, IDTYPE idType, int vint)

{

Entry result;

result.idxfirstLE = lineLT;

result.id = id;

result.idDataType = idDataType;

result.idType = idType;

result.value.vshort = vint;

return result;

}

Entry CreateEntry(int lineLT, std::string id, IDDATATYPE idDataType, IDTYPE idType, std::string vstr)

{

Entry result;

result.idxfirstLE = lineLT;

result.id = id;

result.idDataType = idDataType;

result.idType = idType;

result.value.vstr.str = vstr;

result.value.vstr.len = vstr.size();

return result;

}

Entry CreateEntry(int lineLT, std::string& id, IDDATATYPE idDataType, IDTYPE idType, char vchar)

{

Entry result;

result.idxfirstLE = lineLT;

result.id = id;

result.idDataType = idDataType;

result.idType = idType;

result.value.vchar = vchar;

return result;

}

void SaveToFile(IdTable& idtable, wchar\_t outfile[])

{

std::ofstream file(outfile, std::ios\_base::app);

if (!file.is\_open())

throw ERROR\_THROW(23);

int iddatatype;

int idtype;

int counter = 0;

file << std::setfill('=') << std::setw(43) << "ID TABLE" << std::setw(48) << "\n\n";

file << '+' << std::setfill('-') << std::setw(6) << '+' << std::setw(13) << '+' << std::setw(16) << '+' << std::setw(16) << '+'

<< std::setw(16) << '+' << std::setw(21) << '+' << std::endl;

file << '|' << std::setfill(' ') << std::setw(5) << std::left << "№" << '|' << std::setw(12) << std::left << "Name" << '|'

<< std::setw(15) << std::left << "Type" << '|' << std::setw(15) << std::left << "Data type" << '|' <<

std::setw(15) << std::left << "First in LT" << '|' << std::setw(20) << std::left << "Value" << '|' << std::endl;

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

{

iddatatype = idtable.table[i].idDataType;

idtype = idtable.table[i].idType;

file << '|' << std::setfill(' ') << std::setw(5) << std::left << i;

file << '|' << std::setfill(' ') << std::setw(12) << std::left << idtable.table[i].id;

switch (idtype)

{

case IDTYPE::F:

file << '|' << std::setw(15) << std::left << "Function";

break;

case IDTYPE::V:

file << '|' << std::setw(15) << std::left << "Variable";

break;

case IDTYPE::P:

file << '|' << std::setw(15) << std::left << "Parameter";

break;

case IDTYPE::L:

file << '|' << std::setw(15) << std::left << "Literal";

break;

case IDTYPE::N:

file << '|' << std::setw(15) << std::left << "Undefined";

break;

}

switch (iddatatype)

{

case IDDATATYPE::STR:

file << '|' << std::setw(15) << std::left << "Str";

break;

case IDDATATYPE::SHORT:

file << '|' << std::setw(15) << std::left << "Int";

break;

case IDDATATYPE::BOOL:

file << '|' << std::setw(15) << std::left << "Bool";

break;

case IDDATATYPE::CHAR:

file << '|' << std::setw(15) << std::left << "Char";

break;

case IDDATATYPE::UNDF:

file << '|' << std::setw(15) << std::left << "Undefined";

break;

}

file << '|' << std::setw(15) << std::left << idtable.table[i].idxfirstLE;

if (idtype == IDTYPE::L)

{

switch (iddatatype)

{

case IDDATATYPE::BOOL:

if (idtable.table[i].value.vshort)

file << '|' << std::setw(20) << std::left << "true";

else

file << '|' << std::setw(20) << std::left << "false";

break;

case IDDATATYPE::SHORT:

file << '|' << std::setw(20) << std::left << idtable.table[i].value.vshort;

break;

case IDDATATYPE::STR:

file << '|' << std::setw(20) << std::left << idtable.table[i].value.vstr.str;

break;

case IDDATATYPE::CHAR:

file << L'|' << std::setw(20) << std::left << idtable.table[i].value.vchar;

break;

}

}

else

{

file << '|' << std::setw(20) << std::left << '-';

}

file << '|';

file << std::endl;

}

file.close();

}

}

IT.h:

#pragma once

#define TI\_MAXSIZE 4096

#define TI\_INT\_DEFAULT 0x00000000

#define TI\_STR\_DEFAULT 0x00

#define TI\_NULLIDX 0xffffffff

#define TI\_STR\_MAXSIZE 255

namespace IT

{

enum IDDATATYPE { UNDF = 0, SHORT = 1, STR = 2, BOOL = 3, CHAR = 4 };

enum IDTYPE { N = 0, F = 1, V = 2, P = 3, L = 4 }; // типы идентификаторов: переменная, функция, параметр, литерал, не определено

struct Entry

{

int idxfirstLE; // идекс первого вхождения в таблицу лексем

std::string id; // идентификатор

IDDATATYPE idDataType; // тип данных

IDTYPE idType; // тип идентиикатора

struct

{

int vshort; // целочисленное значение

struct

{

int len; // длин астроки

std::string str; // строка

} vstr;

char vchar; // значение типа char

} value;

};

struct IdTable

{

int maxsize;

int size;

Entry\* table;

};

IdTable Create(int size);

void Add(IdTable& idtable, Entry entry);

void Delete(IdTable& idtable);

void SaveToFile(IdTable& idtable, wchar\_t outfile[]);

Entry GetEntry(IdTable& idtable, int n);

int IsId(IdTable& idtable, std::string id);

Entry CreateEntry(int lineLT, std::string id, IDDATATYPE idDataType, IDTYPE idType);

Entry CreateEntry(int lineLT, std::string id, IDDATATYPE idDataType, IDTYPE idType, int vint);

Entry CreateEntry(int lineLT, std::string id, IDDATATYPE idDataType, IDTYPE idType, std::string vstr);

}

LexicalAnalyzer.cpp:

#include "stdafx.h"

using namespace std;

namespace LA

{

// Конструктор таблиц

LEX::LEX(int lexTableSize, int idTableSize)

{

this->lextable = LT::Create(lexTableSize);

this->idtable = IT::Create(idTableSize);

};

LEX::LEX() { };

GRAPH graph[] =

{

// Каждой лексеме соответсвует своя цепочка разбора

// Типы данных

{ LEX\_SHORT, FST::FST(GRAPH\_SHORT) },

{ LEX\_CHAR, FST::FST(GRAPH\_CHAR) },

{ LEX\_STR, FST::FST(GRAPH\_STR) },

{ LEX\_BOOL, FST::FST(GRAPH\_BOOL) },

// Идентификаторы

{ LEX\_IDENTIFIER, FST::FST(GRAPH\_IDENTIFIERS) },

// Литералы

{ LEX\_LITERAL, FST::FST(GRAPH\_BOOL\_TRUE) },

{ LEX\_LITERAL, FST::FST(GRAPH\_BOOL\_FALSE) },

{ LEX\_LITERAL, FST::FST(GRAPH\_SHORT\_LITERAL\_8) },

{ LEX\_LITERAL, FST::FST(GRAPH\_SHORT\_LITERAL\_10) },

{ LEX\_LITERAL, FST::FST(GRAPH\_STRING\_LITERAL) },

{ LEX\_LITERAL, FST::FST(GRAPH\_CHAR\_LITERAL) },

// Ключевые слова

{ LEX\_FUNCTION, FST::FST(GRAPH\_FUNCTION) },

{ LEX\_NEW, FST::FST(GRAPH\_NEW) },

{ LEX\_RETURN, FST::FST(GRAPH\_RETURN) },

// Функции

{ LEX\_MAIN, FST::FST(GRAPH\_MAIN) },

{ LEX\_STRDUPLICATE, FST::FST(GRAPH\_STRDUPLICATE) },

{ LEX\_STRLENGTH, FST::FST(GRAPH\_STRLENGTH) },

{ LEX\_STRTRANSINT, FST::FST(GRAPH\_STRTRANSINT) },

// Операторы вывода и цикла

{ LEX\_WRITELINE, FST::FST(GRAPH\_WRITELINE) },

{ LEX\_WRITE, FST::FST(GRAPH\_WRITE) },

{ LEX\_CYCLE, FST::FST(GRAPH\_CYCLE) },

// Скобки

{ LEX\_LEFTBRACE\_OPEN, FST::FST(GRAPH\_LEFTBRACE\_OPEN) },

{ LEX\_RIGHTBRACE\_CLOSE, FST::FST(GRAPH\_RIGHTBRACE\_CLOSE) },

{ LEX\_LEFTHESIS\_OPEN, FST::FST(GRAPH\_LEFTHESIS\_OPEN) },

{ LEX\_RIGHTHESIS\_CLOSE, FST::FST(GRAPH\_RIGHTHESIS\_CLOSE) },

// Арифметические операторы

{ LEX\_ADDITION, FST::FST(GRAPH\_ADDITION) },

{ LEX\_SUBSTRACTION, FST::FST(GRAPH\_SUBSTRACTION) },

{ LEX\_MULTIPLICATION, FST::FST(GRAPH\_MULTIPLICATION) },

{ LEX\_DIVISION, FST::FST(GRAPH\_DIVISION) },

{ LEX\_REMAINDERDIVISION, FST::FST(GRAPH\_REMAINDERDIVISION) },

{ LEX\_EQUALS, FST::FST(GRAPH\_EQUALS) },

{ LEX\_MORE, FST::FST(GRAPH\_MORE) },

{ LEX\_LESS, FST::FST(GRAPH\_LESS) },

{ LEX\_MOREEQUAL, FST::FST(GRAPH\_MOREEQUAL) },

{ LEX\_LESSEQUAL, FST::FST(GRAPH\_LESSEQUAL) },

{ LEX\_INEQUALITY, FST::FST(GRAPH\_INEQUALITY) },

{ LEX\_NOTEQUALS, FST::FST(GRAPH\_NOTEQUALS) },

// Разное

{ LEX\_COMMA, FST::FST(GRAPH\_COMMA) },

{ LEX\_SEMICOLON, FST::FST(GRAPH\_SEMICOLON) },

};

// Определяем тип лексемы

char LexType(Tokens::Token token)

{

for (int i = 0; i < LENGTHOF(graph); i++)

{

FST::FST fstlex = FST::FST(token.token, graph[i].graph);

if (FST::execute(fstlex))

{

return graph[i].lex;

}

}

throw ERROR\_THROW\_IN(117, token.line, 0);

}

LEX FillingInTables(Tokens::TokenTable tokenTable)

{

LEX lex = LEX(LT\_MAXSIZE, TI\_MAXSIZE);

char lexema;

string id;

int lineIT = TI\_NULLIDX;

stack<int> areaOfVisibility;

areaOfVisibility.push(0);

int globalAreaOfVisibility = 0;

char\* currentVisibility = new char[0];

IT::IDTYPE idType = IT::IDTYPE::P;

IT::IDDATATYPE idDataType = IT::IDDATATYPE::UNDF;

int isNotGlobal = 0;

bool isMain = false;

int numOfLit = 0;

bool isDeclare = false;

bool GlobalFunIsFound = false;

for (int index = 0; index < tokenTable.size; index++)

{

switch (lexema = LexType(tokenTable.table[index]))

{

case LEX\_IDENTIFIER:

lineIT = SearchID(areaOfVisibility, string(tokenTable.table[index].token), lex.idtable);

// Если id переменной не найден в IT то ищем id функции

if (lineIT == TI\_NULLIDX)

{

lineIT = SearchGlobalFunctionID(globalAreaOfVisibility, string(tokenTable.table[index].token), lex.idtable);

if (lineIT != TI\_NULLIDX)

GlobalFunIsFound = true;

}

// Если id не найден в IT то это объявление

if (lineIT == TI\_NULLIDX && isDeclare || lineIT == TI\_NULLIDX && !isNotGlobal)

{

lineIT = lex.idtable.size;

id = string(tokenTable.table[index].token) + to\_string(areaOfVisibility.top());

IT::Add(lex.idtable, IT::CreateEntry(lex.lextable.size, id, idDataType, idType));

}

else if (lineIT == TI\_NULLIDX)

{

throw ERROR\_THROW\_IN(135, tokenTable.table[index].line, tokenTable.table[index].linePosition);

}

// Если id есть в IT

else if (lineIT != TI\_NULLIDX && isDeclare)

{

if ((IT::GetEntry(lex.idtable, lineIT)).id == string(tokenTable.table[index].token) + to\_string(areaOfVisibility.top()) || GlobalFunIsFound

|| (IT::GetEntry(lex.idtable, lineIT)).id == string(tokenTable.table[index].token) + to\_string(globalAreaOfVisibility) && ((IT::GetEntry(lex.idtable, lineIT)).idType == IT::IDTYPE::F)

|| (IT::GetEntry(lex.idtable, lineIT)).id == string(tokenTable.table[index].token) + to\_string(globalAreaOfVisibility) && ((IT::GetEntry(lex.idtable, lineIT)).idType == IT::IDTYPE::P) && (isNotGlobal == 1))

{

throw ERROR\_THROW\_IN(136, tokenTable.table[index].line, tokenTable.table[index].linePosition);

}

else

{

lineIT = lex.idtable.size;

id = string(tokenTable.table[index].token) + to\_string(areaOfVisibility.top());

IT::Add(lex.idtable, IT::CreateEntry(lex.lextable.size, id, idDataType, idType));

}

}

// Такая функция уже объявлена

else if (lineIT != TI\_NULLIDX && idType == IT::IDTYPE::F)

throw ERROR\_THROW\_IN(143, tokenTable.table[index].line, tokenTable.table[index].linePosition);

LT::Add(lex.lextable, LT::CreateEntry(lexema, tokenTable.table[index].line, lineIT));

idType = IT::IDTYPE::P;

idDataType = IT::IDDATATYPE::UNDF;

GlobalFunIsFound = false;

isDeclare = false;

break;

case LEX\_LITERAL:

// Строковый литерал

if (tokenTable.table[index].token[0] == '"')

{

if (tokenTable.table[index].length == 2)

{

throw ERROR\_THROW\_IN(149, tokenTable.table[index].line, tokenTable.table[index].linePosition);

}

id = "str" + to\_string(numOfLit++);

IT::Add(lex.idtable, IT::CreateEntry(lex.lextable.size, id, IT::IDDATATYPE::STR, IT::IDTYPE::L, string(tokenTable.table[index].token)));

}

// Литерал bool

else if (tokenTable.table[index].token[0] == 't' || tokenTable.table[index].token[0] == 'f')

{

bool vBool;

if (tokenTable.table[index].token[0] == 't')

vBool = true;

else vBool = false;

id = "bool" + to\_string(numOfLit++);

IT::Add(lex.idtable, IT::CreateEntry(lex.lextable.size, id, IT::IDDATATYPE::BOOL, IT::IDTYPE::L, vBool));

}

// Целочисленный литерал

else

{

int vShort;

switch (tokenTable.table[index].token[strlen(tokenTable.table[index].token) - 1])

{

case 'B':

vShort = strtol(tokenTable.table[index].token, NULL, 2);

break;

case 'O':

vShort = strtol(tokenTable.table[index].token, NULL, 8);

break;

case 'H':

vShort = strtol(tokenTable.table[index].token, NULL, 16);

break;

default:

vShort = strtol(tokenTable.table[index].token, NULL, 10);

break;

}

if (vShort < -32768 || vShort > 32767)

throw ERROR\_THROW\_IN(133, tokenTable.table[index].line, tokenTable.table[index].linePosition);

id = "short" + to\_string(numOfLit++);

IT::Add(lex.idtable, IT::CreateEntry(lex.lextable.size, id, IT::IDDATATYPE::SHORT, IT::IDTYPE::L, vShort));

}

LT::Add(lex.lextable, LT::CreateEntry(lexema, tokenTable.table[index].line, lex.idtable.size - 1));

break;

default:

LT::Add(lex.lextable, LT::CreateEntry(lexema, tokenTable.table[index].line));

switch (lexema)

{

case LEX\_LEFTBRACE\_OPEN:

areaOfVisibility.push(lex.lextable.size - 1);

isNotGlobal++;

break;

case LEX\_RIGHTBRACE\_CLOSE:

areaOfVisibility.pop();

isNotGlobal--;

break;

case LEX\_SHORT:

idDataType = IT::IDDATATYPE::SHORT;

break;

case LEX\_STR:

idDataType = IT::IDDATATYPE::STR;

break;

case LEX\_BOOL:

idDataType = IT::IDDATATYPE::BOOL;

break;

case LEX\_NEW:

idType = IT::IDTYPE::V;

isDeclare = true;

break;

case LEX\_FUNCTION:

idType = IT::IDTYPE::F;

if (!isNotGlobal)

{

globalAreaOfVisibility++;

if (areaOfVisibility.top() != 0)

areaOfVisibility.pop();

areaOfVisibility.push(globalAreaOfVisibility);

}

break;

case LEX\_MAIN:

if (isMain)

throw ERROR\_THROW\_IN(131, tokenTable.table[index].line, tokenTable.table[index].linePosition);

isMain = true;

if (!isNotGlobal)

{

globalAreaOfVisibility++;

if (areaOfVisibility.top() != 0)

areaOfVisibility.pop();

areaOfVisibility.push(globalAreaOfVisibility);

}

break;

default:

break;

}

break;

}

}

if (!isMain)

throw ERROR\_THROW(134);

return lex;

}

// Поиск id в IT

int SearchID(stack<int> areaOfVisibility, string id, IT::IdTable& idtable)

{

char\* currentVisibility = new char[0];

string searchingID;

int result = TI\_NULLIDX;

while (areaOfVisibility.top() != 0)

{

itoa(areaOfVisibility.top(), currentVisibility, 10);

areaOfVisibility.pop();

searchingID = id + string(currentVisibility);

result = IT::IsId(idtable, searchingID);

if (result != TI\_NULLIDX)

return result;

}

return result;

}

// Поиск id функции в IT

int SearchGlobalFunctionID(int globalAreaOfVisibility, string id, IT::IdTable& idtable)

{

int result = TI\_NULLIDX;

string searchingID;

for (int i = 1; i < globalAreaOfVisibility; i++)

{

searchingID = id + to\_string(i);

result = IT::IsId(idtable, searchingID);

if (result != TI\_NULLIDX && idtable.table[result].idType == IT::IDTYPE::F)

return result;

}

return TI\_NULLIDX;

}

}

LexicalAnalyzer.h:

#pragma once

#include "stdafx.h"

#include "IT.h"

#include "LT.h"

#include "Token.h"

using namespace std;

namespace LA

{

struct LEX

{

IT::IdTable idtable;

LT::LexTable lextable;

LEX();

LEX(int lexTableSize, int idTableSize);

};

// Определяем тип лексемы

char LexType(Tokens::Token token);

// Заполнение таблиц

LEX FillingInTables(Tokens::TokenTable tokenTable);

// Поиск id в таблице

int SearchID(stack<int> areaOfVisibility, string id, IT::IdTable& idTable);

// Поиск id функции в таблице

int SearchGlobalFunctionID(int globalAreaOfVisibility, string id, IT::IdTable& idTable);

}

Log.cpp:

#include "stdafx.h"

#include "Log.h"

#include <fstream>

#include <string>

#include <iostream>

#include <ctime>

namespace Log

{

LOG getlog(wchar\_t logfile[])

{

LOG temp;

temp.stream = new std::ofstream(logfile);

if (!temp.stream->is\_open())

throw ERROR\_THROW(21);

wcscpy\_s(temp.logfile, logfile);

return temp;

}

void WriteLine(LOG log, const char\* c, ...)

{

const char\*\* ptr = &c;

if (log.stream == nullptr || !log.stream->is\_open())

{

while (strlen(\*ptr))

{

std::cout << \*ptr;

++ptr;

}

}

else

{

while (strlen(\*ptr))

{

\*log.stream << \*ptr;

++ptr;

}

}

}

void WriteLine(LOG log, const wchar\_t\* c, ...)

{

const wchar\_t\*\* ptr = &c;

while (wcslen(\*ptr))

{

char out[PARM\_MAX\_SIZE];

size\_t charsConverted = 0;

wcstombs\_s(&charsConverted, out, \*ptr, PARM\_MAX\_SIZE);

\*log.stream << out;

++ptr;

}

\*log.stream << std::endl;

}

void WriteLog(LOG log)

{

time\_t t = time(nullptr);

tm now;

localtime\_s(&now, &t);

char date[PARM\_MAX\_SIZE];

strftime(date, PARM\_MAX\_SIZE, "%d.%m.%Y %H:%M:%S", &now);

\*log.stream << "-------------- Протокол ----------------\nДата: " << date << std::endl;

}

void WriteParm(LOG log, Parm::PARM parm)

{

\*log.stream << "------------- Параметры ----------------" << std::endl;

char out[PARM\_MAX\_SIZE];

size\_t charsConverted(0);

wcstombs\_s(&charsConverted, out, parm.in, PARM\_MAX\_SIZE);

\*log.stream << "-in: " << out << std::endl;

wcstombs\_s(&charsConverted, out, parm.log, PARM\_MAX\_SIZE);

\*log.stream << "-log: " << out << std::endl;

wcstombs\_s(&charsConverted, out, parm.out, PARM\_MAX\_SIZE);

\*log.stream << "-out: " << out << std::endl;

wcstombs\_s(&charsConverted, out, parm.an, PARM\_MAX\_SIZE);

\*log.stream << "-an: " << out << std::endl;

}

void WriteIn(LOG log, In::IN in)

{

\*log.stream << "---------- Исходные данные -------------" << std::endl;

\*log.stream << "Количество символов: " << in.size << std::endl;

\*log.stream << "Проигнорировано: " << in.ignor << std::endl;

\*log.stream << "Количество строк: " << in.lines << std::endl;

\*log.stream << "----------------------------------------" << std::endl;

}

void WriteError(LOG log, Error::ERROR error)

{

std::cout << "Ошибка " << error.id << ": " << error.message << ", строка " << error.inext.line << ", позиция " << error.inext.col << std::endl;

if (!(log.stream == nullptr || !log.stream->is\_open()))

{

\*log.stream << "Ошибка " << error.id << ": " << error.message << ", строка " << error.inext.line << ", позиция " << error.inext.col << std::endl;

}

}

void Close(LOG log)

{

if (log.stream != nullptr)

{

log.stream->close();

delete log.stream;

}

}

}

Log.h:

#pragma once

#include <fstream>

#include "In.h"

#include "Parm.h"

#include "Error.h"

namespace Log

{

struct LOG

{

wchar\_t logfile[PARM\_MAX\_SIZE];

std::ofstream\* stream;

};

static const LOG INITLOG = { L"",NULL };

LOG getlog(wchar\_t logfile[]);

void WriteLine(LOG log, const char\* c, ...);

void WriteLine(LOG, const wchar\_t\* c, ...);

void WriteLog(LOG log);

void WriteParm(LOG log, Parm::PARM parm);

void WriteIn(LOG log, In::IN in);

void WriteError(LOG log, Error::ERROR error);

void Close(LOG log);

}

LPA-2024.cpp:

#include "stdafx.h"

#include <locale>

#include <cwchar>

int \_tmain(int argc, \_TCHAR\*\* argv)

{

//argc = 3;

//argv[1] = (\_TCHAR\*)L"-in:D:\\KPO\_3sem\\SIV-2023\\SIV-2023\\ControlExample.txt";

//argv[1] = (\_TCHAR\*)L"-in:D:\\KPO\_3sem\\SIV-2023\\SIV-2023\\SemErrors.txt";

//argv[2] = (\_TCHAR\*)L"-out:D:\\KPO\_3sem\\SIV-2023\\SIV-2023\\SIV\_result.asm";

//argv[3] = (\_TCHAR\*)L"-log:D:\\KPO\_3sem\\SIV-2023\\SIV-2023\\Logger.txt";

//argv[4] = (\_TCHAR\*)L"-an:D:\\KPO\_3sem\\SIV-2023\\SIV-2023\\Analize.txt";

setlocale(LC\_ALL, "rus");

Log::LOG log = Log::INITLOG;

try

{

Parm::PARM parm = Parm::getparm(argc, argv);

std::ofstream clearFile(parm.an, std::ios\_base::trunc);

log = Log::getlog(parm.log);

Log::WriteLog(log);

Log::WriteParm(log, parm);

In::IN in = In::GetIn(parm.in, parm.out);

Log::WriteIn(log, in);

Log::WriteLine(log, "No forbidden characters\n", "");

Log::WriteLine(log, "Tokenization : ", "");

Tokens::TokenTable tokentable = Tokens::Tokenize(in);

Log::WriteLine(log, " Completed successfully\n", "");

Log::WriteLine(log, "Lexical analysis : ", "");

LA::LEX lex = LA::FillingInTables(tokentable);

Log::WriteLine(log, " Completed successfully\n", "");

LT::SaveToFile(lex.lextable, parm.an);

IT::SaveToFile(lex.idtable, parm.an);

Log::Close(log);

std::cout << "Code translation completed successfully\n";

}

catch (Error::ERROR e)

{

std::cout << "Произошла ошибка\n";

Log::WriteLine(log, "- ERROR\n", "");

Log::WriteError(log, e);

}

return 0;

}

LT.cpp:

#include "stdafx.h"

namespace LT

{

LexTable Create(int size) // Ф-ция создания таблицы лексем

{

if (size > LT\_MAXSIZE) // Ошибка, если size > LT\_MAXSIZE

throw ERROR\_THROW(111);

LexTable lextable; // Создание таблицы лексем

lextable.maxsize = size; // Размер таблицы лексем

lextable.size = 0; // Текущий размер таблица лексем

lextable.table = new Entry[lextable.maxsize]; // Массив строк таблиц лексем

return lextable; // Возвращает заполненную структуру

}

void Add(LexTable& lextable, Entry entry) // Добавление строки в таблицу лексем

{

if (lextable.size > lextable.maxsize) // Проверка: есть ли свободное место в таблице лексем

throw ERROR\_THROW(112); // Если нет, то ошибка

lextable.table[lextable.size] = entry; // Добавление строки в таблицу под индексом = текущему размеру таблицы

lextable.size++; // Увеличение текущего размера таблицы, т.к. была добавлена новая строка

}

Entry GetEntry(LexTable& lextable, int n) // Получение строки таблицы лексем

{

if (n > (lextable.size - 1)) // Проверка: есть ли строка с таким номером входит в таблицу лексем

throw ERROR\_THROW(113); // Если нет, то ошибка

return lextable.table[n]; // Возвращает строку таблицы идентификаторов

}

void Delete(LexTable& lextable) // Удаление таблицы лексем (Освобождение памяти)

{

delete[] lextable.table; // Освобождение ранее выделенной памяти

}

void SaveLexTableInFile(LexTable lextable, wchar\_t outfile[])

{

std::ofstream out(outfile);

if (!out.is\_open())

throw ERROR\_THROW(22);

int line = 0;

out << std::right << std::setw(3) << line << " ";

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

{

if (lextable.table[i].sn == line)

{

out << lextable.table[i].lexema;

}

else

{

out << std::endl;

line += (lextable.table[i].sn - line);

out << std::right << std::setw(3) << line << " " << lextable.table[i].lexema;

}

}

out.close();

}

void SaveToFile(LexTable& lextable, wchar\_t outfile[])

{

std::ofstream file(outfile, std::ios\_base::app);

if (!file.is\_open())

throw ERROR\_THROW(22);

file << std::setfill('=') << std::setw(23) << "LEX TABLE" << std::setw(17) << "\n\n";

file << '+' << std::setfill('-') << std::setw(6) << '+' << std::setw(10) <<

'+' << std::setw(10) << '+' << std::setw(11) << '+' << std::endl;

file << '|' << std::setfill(' ') << std::setw(5) << std::left << "№" << '|' << std::setw(9) << std::left << "Line" << '|'

<< std::setw(9) << std::left << "Lexema" << '|' << std::setw(10) << std::left << "ID from IT" << '|' << std::endl;

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

{

file << '|' << std::setfill(' ') << std::setw(5) << std::left << i << '|' << std::setw(9) << std::left << lextable.table[i].sn << '|'

<< std::setw(9) << std::left << lextable.table[i].lexema << '|';

if (lextable.table[i].idxTI == TI\_NULLIDX)

file << std::setw(10) << std::left << '-' << '|' << std::endl;

else

file << std::setw(10) << std::left << lextable.table[i].idxTI << '|' << std::endl;

}

file.close();

}

Entry CreateEntry(char lexema, int line)

{

Entry entry;

entry.lexema = lexema;

entry.sn = line;

entry.idxTI = LT\_TI\_NULLTDX;

return entry;

}

Entry CreateEntry(char lexema, int line, int idTI)

{

Entry result;

result.lexema = lexema;

result.sn = line;

result.idxTI = idTI;

return result;

}

}

LT.h:

#pragma once

#include "stdafx.h"

#define LEXEMA\_FIXSIZE 1

#define LT\_MAXSIZE 4096

#define LT\_TI\_NULLTDX 0xffffffff

// Типы данных

#define LEX\_SHORT 'd'

#define LEX\_CHAR 'c'

#define LEX\_STR 's'

#define LEX\_BOOL 'b'

// Идентификаторы и литералы

#define LEX\_IDENTIFIER 'i'

#define LEX\_LITERAL 'l'

// Функции

#define LEX\_STRDUPLICATE 'g'

#define LEX\_STRLENGTH 'h'

#define LEX\_STRTRANSINT 't'

#define LEX\_MAIN 'm'

// Ключевые слова

#define LEX\_NEW 'n'

#define LEX\_FUNCTION 'f'

#define LEX\_RETURN 'r'

// Операторы вывода в консоль и цикла

#define LEX\_WRITELINE 'w'

#define LEX\_WRITE 'v'

#define LEX\_CYCLE 'p'

// Скобки

#define LEX\_LEFTBRACE\_OPEN '{'

#define LEX\_RIGHTBRACE\_CLOSE '}'

#define LEX\_LEFTHESIS\_OPEN '('

#define LEX\_RIGHTHESIS\_CLOSE ')'

// Арифметические операторы

#define LEX\_ADDITION '+'

#define LEX\_SUBSTRACTION '-'

#define LEX\_MULTIPLICATION '\*'

#define LEX\_DIVISION '/'

#define LEX\_REMAINDERDIVISION '%'

#define LEX\_EQUALS '='

// Логические операторы

#define LEX\_MORE '>'

#define LEX\_LESS '<'

#define LEX\_MOREEQUAL ']'

#define LEX\_LESSEQUAL '['

#define LEX\_INEQUALITY '#'

#define LEX\_NOTEQUALS '!'

// Разное

#define LEX\_COMMA ','

#define LEX\_SEMICOLON ';'

namespace LT

{

struct Entry

{

char lexema; // символ лексемы

int sn; // номер строки в коде

int idxTI; // номер в IT

};

struct LexTable

{

int maxsize;

int size;

Entry\* table;

};

LexTable Create(int size);

void Add(LexTable& lextable, Entry entry);

void Delete(LexTable& lextable);

void SaveToFile(LexTable& lextable, wchar\_t outfile[]);

Entry GetEntry(LexTable& lextable, int n);

Entry CreateEntry(char lexema, int line);

Entry CreateEntry(char lexema, int line, int idTI);

void SaveLexTableInFile(LexTable lextable, wchar\_t outfile[]);

};

Parm.cpp:

#include "stdafx.h"

namespace Parm

{

PARM getparm(int argc, \_TCHAR\* argv[])

{

PARM parm = {};

for (int i = 1; i < argc; i++)

{

if (wcslen(argv[i]) > PARM\_MAX\_SIZE)

{

throw ERROR\_THROW(11)

}

if (wcsstr(argv[i], PARM\_IN))

wcscpy\_s(parm.in, argv[i] + wcslen(PARM\_IN));

else if (wcsstr(argv[i], PARM\_OUT))

wcscpy\_s(parm.out, argv[i] + wcslen(PARM\_OUT));

else if (wcsstr(argv[i], PARM\_LOG))

wcscpy\_s(parm.log, argv[i] + wcslen(PARM\_LOG));

else if (wcsstr(argv[i], PARM\_AN))

wcscpy\_s(parm.an, argv[i] + wcslen(PARM\_AN));

}

if (wcslen(parm.in) == 0)

throw ERROR\_THROW(10);

if (wcslen(parm.out) == 0)

{

throw ERROR\_THROW(12);

/\*wcscpy\_s(parm.out, parm.in);

wcsncat\_s(parm.out, PARM\_OUT\_DEFAULT\_EXT, wcslen(PARM\_OUT\_DEFAULT\_EXT));\*/

}

if (wcslen(parm.log) == 0)

{

wcscpy\_s(parm.log, parm.in);

wcsncat\_s(parm.log, PARM\_LOG\_DEFAULT\_EXT, wcslen(PARM\_LOG\_DEFAULT\_EXT));

}

if (wcslen(parm.an) == 0)

{

wcscpy\_s(parm.an, parm.in);

wcsncat\_s(parm.an, PARM\_AN\_DEFAULT\_EXT, wcslen(PARM\_AN\_DEFAULT\_EXT));

}

return parm;

}

}

Parm.h:

#pragma once

#define PARM\_IN L"-in:"

#define PARM\_OUT L"-out:"

#define PARM\_LOG L"-log:"

#define PARM\_AN L"-an:"

#define PARM\_MAX\_SIZE 300

#define PARM\_OUT\_DEFAULT\_EXT L".asm"

#define PARM\_LOG\_DEFAULT\_EXT L".log"

#define PARM\_AN\_DEFAULT\_EXT L".an.txt"

namespace Parm

{

struct PARM

{

wchar\_t in[PARM\_MAX\_SIZE];

wchar\_t out[PARM\_MAX\_SIZE];

wchar\_t log[PARM\_MAX\_SIZE];

wchar\_t an[PARM\_MAX\_SIZE];

};

PARM getparm(int argc, \_TCHAR\* argv[]);

}

stdafx.cpp:

// stdafx.cpp: файл исходного кода, соответствующий предварительно скомпилированному заголовочному файлу

#include "stdafx.h"

// При использовании предварительно скомпилированных заголовочных файлов необходим следующий файл исходного кода для выполнения сборки.

stdafx.h:

// stdafx.h: это предварительно скомпилированный заголовочный файл.

// Перечисленные ниже файлы компилируются только один раз, что ускоряет последующие сборки.

// Это также влияет на работу IntelliSense, включая многие функции просмотра и завершения кода.

// Однако изменение любого из приведенных здесь файлов между операциями сборки приведет к повторной компиляции всех(!) этих файлов.

// Не добавляйте сюда файлы, которые планируете часто изменять, так как в этом случае выигрыша в производительности не будет.

#ifndef STDAFX\_H

#define STDAFX\_H

#define LENGTHOF(x)(sizeof(x)/sizeof(\*x))

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_NONSTDC\_NO\_DEPRECATE

#include <SDKDDKVer.h>

#include <stdio.h>

#include <tchar.h>

#include <iostream>

#include <fstream>

#include <ctime>

#include <mbstring.h>

#include <iomanip>

#include <string>

#include <stack>

#include "Error.h"

#include "FST.h"

#include "Graphs.h"

#include "IT.h"

#include "LexicalAnalyzer.h"

#include "Log.h"

#include "LT.h"

#include "Parm.h"

#include "Token.h"

#endif //STDAFX\_H

Graphs.h:

#pragma once

#include "FST.h"

struct GRAPH

{

char lex;

FST::FST graph; // Недетерминированный конечный автомат

};

// Типы данных

#define GRAPH\_SHORT \

4, \

FST::NODE(1, FST::RELATION('i', 1)), \

FST::NODE(1, FST::RELATION('n', 2)), \

FST::NODE(1, FST::RELATION('t', 3)), \

FST::NODE()

#define GRAPH\_CHAR \

5, \

FST::NODE(1, FST::RELATION('c', 1)), \

FST::NODE(1, FST::RELATION('h', 2)), \

FST::NODE(1, FST::RELATION('a', 3)), \

FST::NODE(1, FST::RELATION('r', 4)), \

FST::NODE()

#define GRAPH\_STR \

4, \

FST::NODE(1, FST::RELATION('s', 1)), \

FST::NODE(1, FST::RELATION('t', 2)), \

FST::NODE(1, FST::RELATION('r', 3)), \

FST::NODE()

#define GRAPH\_BOOL \

5, \

FST::NODE(1, FST::RELATION('b', 1)), \

FST::NODE(1, FST::RELATION('o', 2)), \

FST::NODE(1, FST::RELATION('o', 3)), \

FST::NODE(1, FST::RELATION('l', 4)), \

FST::NODE()

// Идентификаторы

#define GRAPH\_IDENTIFIERS \

3, \

FST::NODE(52, \

FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), \

FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1), \

FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), \

FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1), \

FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), \

FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1), \

FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), \

FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1), \

FST::RELATION('y', 1), FST::RELATION('z', 1), \

FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), \

FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2), \

FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), \

FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2), \

FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), \

FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2), \

FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), \

FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2), \

FST::RELATION('y', 2), FST::RELATION('z', 2)), \

FST::NODE(72, \

FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), \

FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1), \

FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), \

FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1), \

FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), \

FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1), \

FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), \

FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1), \

FST::RELATION('y', 1), FST::RELATION('z', 1), FST::RELATION('0', 1), \

FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), \

FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1), \

FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), \

\

FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), \

FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2), \

FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), \

FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2), \

FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), \

FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2), \

FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), \

FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2), \

FST::RELATION('y', 2), FST::RELATION('z', 2), FST::RELATION('0', 2), \

FST::RELATION('1', 2), FST::RELATION('2', 2), FST::RELATION('3', 2), \

FST::RELATION('4', 2), FST::RELATION('5', 2), FST::RELATION('6', 2), \

FST::RELATION('7', 2), FST::RELATION('8', 2), FST::RELATION('9', 2)), \

FST::NODE()

// Литералы

#define GRAPH\_BOOL\_TRUE \

5, \

FST::NODE(1, FST::RELATION('t', 1)), \

FST::NODE(1, FST::RELATION('r', 2)), \

FST::NODE(1, FST::RELATION('u', 3)), \

FST::NODE(1, FST::RELATION('e', 4)), \

FST::NODE()

#define GRAPH\_BOOL\_FALSE \

6, \

FST::NODE(1, FST::RELATION('f', 1)), \

FST::NODE(1, FST::RELATION('a', 2)), \

FST::NODE(1, FST::RELATION('l', 3)), \

FST::NODE(1, FST::RELATION('s', 4)), \

FST::NODE(1, FST::RELATION('e', 5)), \

FST::NODE()

#define GRAPH\_SHORT\_LITERAL\_8 \

5, \

FST::NODE(3, \

FST::RELATION('-', 1), FST::RELATION('0', 2), FST::RELATION('0', 4)), \

FST::NODE(1, FST::RELATION('0', 2)), \

FST::NODE(14, \

FST::RELATION('1', 3), FST::RELATION('2', 3), FST::RELATION('3', 3), \

FST::RELATION('4', 3), FST::RELATION('5', 3), FST::RELATION('6', 3), \

FST::RELATION('7', 3), \

FST::RELATION('1', 4), FST::RELATION('2', 4), FST::RELATION('3', 4), \

FST::RELATION('4', 4), FST::RELATION('5', 4), FST::RELATION('6', 4), \

FST::RELATION('7', 4)), \

FST::NODE(16, \

FST::RELATION('0', 3), FST::RELATION('1', 3), FST::RELATION('2', 3), \

FST::RELATION('3', 3), FST::RELATION('4', 3), FST::RELATION('5', 3), \

FST::RELATION('6', 3), FST::RELATION('7', 3), \

FST::RELATION('0', 4), FST::RELATION('1', 4), FST::RELATION('2', 4), \

FST::RELATION('3', 4), FST::RELATION('4', 4), FST::RELATION('5', 4), \

FST::RELATION('6', 4), FST::RELATION('7', 4)), \

FST::NODE()

#define GRAPH\_SHORT\_LITERAL\_10 \

4, \

FST::NODE(11, \

FST::RELATION('-', 1), FST::RELATION('0', 3), \

FST::RELATION('1', 2), FST::RELATION('2', 2), FST::RELATION('3', 2), \

FST::RELATION('4', 2), FST::RELATION('5', 2), FST::RELATION('6', 2), \

FST::RELATION('7', 2), FST::RELATION('8', 2), FST::RELATION('9', 2)), \

FST::NODE(18, \

FST::RELATION('1', 2), FST::RELATION('2', 2), FST::RELATION('3', 2), \

FST::RELATION('4', 2), FST::RELATION('5', 2), FST::RELATION('6', 2), \

FST::RELATION('7', 2), FST::RELATION('8', 2), FST::RELATION('9', 2), \

FST::RELATION('1', 3), FST::RELATION('2', 3), FST::RELATION('3', 3), \

FST::RELATION('4', 3), FST::RELATION('5', 3), FST::RELATION('6', 3), \

FST::RELATION('7', 3), FST::RELATION('8', 3), FST::RELATION('9', 3)), \

FST::NODE(20, \

FST::RELATION('0', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), \

FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2), \

FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), \

FST::RELATION('9', 2), \

FST::RELATION('0', 3), FST::RELATION('1', 3), FST::RELATION('2', 3), \

FST::RELATION('3', 3), FST::RELATION('4', 3), FST::RELATION('5', 3), \

FST::RELATION('6', 3), FST::RELATION('7', 3), FST::RELATION('8', 3), \

FST::RELATION('9', 3)), \

FST::NODE()

#define GRAPH\_STRING\_LITERAL \

4, \

FST::NODE(2, FST::RELATION('\"', 1), FST::RELATION('\"', 2)), \

FST::NODE(290, \

FST::RELATION('A', 1), FST::RELATION('B', 1), FST::RELATION('C', 1), \

FST::RELATION('D', 1), FST::RELATION('E', 1), FST::RELATION('F', 1), \

FST::RELATION('G', 1), FST::RELATION('H', 1), FST::RELATION('I', 1), \

FST::RELATION('J', 1), FST::RELATION('K', 1), FST::RELATION('L', 1), \

FST::RELATION('M', 1), FST::RELATION('N', 1), FST::RELATION('O', 1), \

FST::RELATION('P', 1), FST::RELATION('Q', 1), FST::RELATION('R', 1), \

FST::RELATION('S', 1), FST::RELATION('T', 1), FST::RELATION('U', 1), \

FST::RELATION('V', 1), FST::RELATION('W', 1), FST::RELATION('X', 1), \

FST::RELATION('Y', 1), FST::RELATION('Z', 1), \

FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), \

FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1), \

FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), \

FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1), \

FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), \

FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1), \

FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), \

FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1), \

FST::RELATION('y', 1), FST::RELATION('z', 1), \

\

FST::RELATION('0', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), \

FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), \

FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1), \

FST::RELATION('9', 1), \

\

FST::RELATION('А', 1), FST::RELATION('Б', 1), FST::RELATION('В', 1), \

FST::RELATION('Г', 1), FST::RELATION('Д', 1), FST::RELATION('Е', 1), \

FST::RELATION('Ё', 1), FST::RELATION('Ж', 1), FST::RELATION('З', 1), \

FST::RELATION('И', 1), FST::RELATION('Й', 1), FST::RELATION('К', 1), \

FST::RELATION('Л', 1), FST::RELATION('М', 1), FST::RELATION('Н', 1), \

FST::RELATION('О', 1), FST::RELATION('П', 1), FST::RELATION('Р', 1), \

FST::RELATION('С', 1), FST::RELATION('Т', 1), FST::RELATION('У', 1), \

FST::RELATION('Ф', 1), FST::RELATION('Х', 1), FST::RELATION('Ц', 1), \

FST::RELATION('Ч', 1), FST::RELATION('Ш', 1), FST::RELATION('Щ', 1), \

FST::RELATION('Ъ', 1), FST::RELATION('Ы', 1), FST::RELATION('Ь', 1), \

FST::RELATION('Э', 1), FST::RELATION('Ю', 1), FST::RELATION('Я', 1), \

FST::RELATION('а', 1), FST::RELATION('б', 1), FST::RELATION('в', 1), \

FST::RELATION('г', 1), FST::RELATION('д', 1), FST::RELATION('е', 1), \

FST::RELATION('ё', 1), FST::RELATION('ж', 1), FST::RELATION('з', 1), \

FST::RELATION('и', 1), FST::RELATION('й', 1), FST::RELATION('к', 1), \

FST::RELATION('л', 1), FST::RELATION('м', 1), FST::RELATION('н', 1), \

FST::RELATION('о', 1), FST::RELATION('п', 1), FST::RELATION('р', 1), \

FST::RELATION('с', 1), FST::RELATION('т', 1), FST::RELATION('у', 1), \

FST::RELATION('ф', 1), FST::RELATION('х', 1), FST::RELATION('ц', 1), \

FST::RELATION('ч', 1), FST::RELATION('ш', 1), FST::RELATION('щ', 1), \

FST::RELATION('ъ', 1), FST::RELATION('ы', 1), FST::RELATION('ь', 1), \

FST::RELATION('э', 1), FST::RELATION('ю', 1), FST::RELATION('я', 1), \

\

FST::RELATION(' ', 1), FST::RELATION('.', 1), FST::RELATION(',', 1), \

FST::RELATION('?', 1), FST::RELATION('!', 1), FST::RELATION(';', 1), \

FST::RELATION(':', 1), FST::RELATION('-', 1), FST::RELATION(')', 1), \

FST::RELATION('(', 1), FST::RELATION('<', 1), FST::RELATION('>', 1), \

FST::RELATION('/', 1), FST::RELATION('%', 1), FST::RELATION('=', 1), \

FST::RELATION('+', 1), FST::RELATION('\*', 1), \

\

FST::RELATION('A', 2), FST::RELATION('B', 2), FST::RELATION('C', 2), \

FST::RELATION('D', 2), FST::RELATION('E', 2), FST::RELATION('F', 2), \

FST::RELATION('G', 2), FST::RELATION('H', 2), FST::RELATION('I', 2), \

FST::RELATION('J', 2), FST::RELATION('K', 2), FST::RELATION('L', 2), \

FST::RELATION('M', 2), FST::RELATION('N', 2), FST::RELATION('O', 2), \

FST::RELATION('P', 2), FST::RELATION('Q', 2), FST::RELATION('R', 2), \

FST::RELATION('S', 2), FST::RELATION('T', 2), FST::RELATION('U', 2), \

FST::RELATION('V', 2), FST::RELATION('W', 2), FST::RELATION('X', 2), \

FST::RELATION('Y', 2), FST::RELATION('Z', 2), \

FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), \

FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2), \

FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), \

FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2), \

FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), \

FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2), \

FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), \

FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2), \

FST::RELATION('y', 2), FST::RELATION('z', 2), \

\

FST::RELATION('0', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), \

FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2), \

FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), \

FST::RELATION('9', 2), \

\

FST::RELATION('А', 2), FST::RELATION('Б', 2), FST::RELATION('В', 2), \

FST::RELATION('Г', 2), FST::RELATION('Д', 2), FST::RELATION('Е', 2), \

FST::RELATION('Ё', 2), FST::RELATION('Ж', 2), FST::RELATION('З', 2), \

FST::RELATION('И', 2), FST::RELATION('Й', 2), FST::RELATION('К', 2), \

FST::RELATION('Л', 2), FST::RELATION('М', 2), FST::RELATION('Н', 2), \

FST::RELATION('О', 2), FST::RELATION('П', 2), FST::RELATION('Р', 2), \

FST::RELATION('С', 2), FST::RELATION('Т', 2), FST::RELATION('У', 2), \

FST::RELATION('Ф', 2), FST::RELATION('Х', 2), FST::RELATION('Ц', 2), \

FST::RELATION('Ч', 2), FST::RELATION('Ш', 2), FST::RELATION('Щ', 2), \

FST::RELATION('Ъ', 2), FST::RELATION('Ы', 2), FST::RELATION('Ь', 2), \

FST::RELATION('Э', 2), FST::RELATION('Ю', 2), FST::RELATION('Я', 2), \

FST::RELATION('а', 2), FST::RELATION('б', 2), FST::RELATION('в', 2), \

FST::RELATION('г', 2), FST::RELATION('д', 2), FST::RELATION('е', 2), \

FST::RELATION('ё', 2), FST::RELATION('ж', 2), FST::RELATION('з', 2), \

FST::RELATION('и', 2), FST::RELATION('й', 2), FST::RELATION('к', 2), \

FST::RELATION('л', 2), FST::RELATION('м', 2), FST::RELATION('н', 2), \

FST::RELATION('о', 2), FST::RELATION('п', 2), FST::RELATION('р', 2), \

FST::RELATION('с', 2), FST::RELATION('т', 2), FST::RELATION('у', 2), \

FST::RELATION('ф', 2), FST::RELATION('х', 2), FST::RELATION('ц', 2), \

FST::RELATION('ч', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), \

FST::RELATION('ъ', 2), FST::RELATION('ы', 2), FST::RELATION('ь', 2), \

FST::RELATION('э', 2), FST::RELATION('ю', 2), FST::RELATION('я', 2), \

\

FST::RELATION(' ', 2), FST::RELATION('.', 2), FST::RELATION(',', 2), \

FST::RELATION('?', 2), FST::RELATION('!', 2), FST::RELATION(';', 2), \

FST::RELATION(':', 2), FST::RELATION('-', 2), FST::RELATION(')', 2), \

FST::RELATION('(', 2), FST::RELATION('<', 2), FST::RELATION('>', 2), \

FST::RELATION('/', 2), FST::RELATION('%', 2), FST::RELATION('=', 2), \

FST::RELATION('+', 2), FST::RELATION('\*', 2)), \

FST::NODE(1, FST::RELATION('\"', 3)), \

FST::NODE()

#define GRAPH\_CHAR\_LITERAL \

4, \

FST::NODE(2, FST::RELATION('\'', 1), FST::RELATION('\'', 2)), \

FST::NODE(145, \

FST::RELATION('A', 2), FST::RELATION('B', 2), FST::RELATION('C', 2), \

FST::RELATION('D', 2), FST::RELATION('E', 2), FST::RELATION('F', 2), \

FST::RELATION('G', 2), FST::RELATION('H', 2), FST::RELATION('I', 2), \

FST::RELATION('J', 2), FST::RELATION('K', 2), FST::RELATION('L', 2), \

FST::RELATION('M', 2), FST::RELATION('N', 2), FST::RELATION('O', 2), \

FST::RELATION('P', 2), FST::RELATION('Q', 2), FST::RELATION('R', 2), \

FST::RELATION('S', 2), FST::RELATION('T', 2), FST::RELATION('U', 2), \

FST::RELATION('V', 2), FST::RELATION('W', 2), FST::RELATION('X', 2), \

FST::RELATION('Y', 2), FST::RELATION('Z', 2), \

FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), \

FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2), \

FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), \

FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2), \

FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), \

FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2), \

FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), \

FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2), \

FST::RELATION('y', 2), FST::RELATION('z', 2), \

\

FST::RELATION('0', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), \

FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2), \

FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), \

FST::RELATION('9', 2), \

\

FST::RELATION('А', 2), FST::RELATION('Б', 2), FST::RELATION('В', 2), \

FST::RELATION('Г', 2), FST::RELATION('Д', 2), FST::RELATION('Е', 2), \

FST::RELATION('Ё', 2), FST::RELATION('Ж', 2), FST::RELATION('З', 2), \

FST::RELATION('И', 2), FST::RELATION('Й', 2), FST::RELATION('К', 2), \

FST::RELATION('Л', 2), FST::RELATION('М', 2), FST::RELATION('Н', 2), \

FST::RELATION('О', 2), FST::RELATION('П', 2), FST::RELATION('Р', 2), \

FST::RELATION('С', 2), FST::RELATION('Т', 2), FST::RELATION('У', 2), \

FST::RELATION('Ф', 2), FST::RELATION('Х', 2), FST::RELATION('Ц', 2), \

FST::RELATION('Ч', 2), FST::RELATION('Ш', 2), FST::RELATION('Щ', 2), \

FST::RELATION('Ъ', 2), FST::RELATION('Ы', 2), FST::RELATION('Ь', 2), \

FST::RELATION('Э', 2), FST::RELATION('Ю', 2), FST::RELATION('Я', 2), \

FST::RELATION('а', 2), FST::RELATION('б', 2), FST::RELATION('в', 2), \

FST::RELATION('г', 2), FST::RELATION('д', 2), FST::RELATION('е', 2), \

FST::RELATION('ё', 2), FST::RELATION('ж', 2), FST::RELATION('з', 2), \

FST::RELATION('и', 2), FST::RELATION('й', 2), FST::RELATION('к', 2), \

FST::RELATION('л', 2), FST::RELATION('м', 2), FST::RELATION('н', 2), \

FST::RELATION('о', 2), FST::RELATION('п', 2), FST::RELATION('р', 2), \

FST::RELATION('с', 2), FST::RELATION('т', 2), FST::RELATION('у', 2), \

FST::RELATION('ф', 2), FST::RELATION('х', 2), FST::RELATION('ц', 2), \

FST::RELATION('ч', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), \

FST::RELATION('ъ', 2), FST::RELATION('ы', 2), FST::RELATION('ь', 2), \

FST::RELATION('э', 2), FST::RELATION('ю', 2), FST::RELATION('я', 2), \

\

FST::RELATION(' ', 2), FST::RELATION('.', 2), FST::RELATION(',', 2), \

FST::RELATION('?', 2), FST::RELATION('!', 2), FST::RELATION(';', 2), \

FST::RELATION(':', 2), FST::RELATION('-', 2), FST::RELATION(')', 2), \

FST::RELATION('(', 2), FST::RELATION('<', 2), FST::RELATION('>', 2), \

FST::RELATION('/', 2), FST::RELATION('%', 2), FST::RELATION('=', 2), \

FST::RELATION('+', 2), FST::RELATION('\*', 2)), \

FST::NODE(1, FST::RELATION('\'', 3)), \

FST::NODE()

// Функции

#define GRAPH\_MAIN \

5, \

FST::NODE(1,FST::RELATION('m', 1)), \

FST::NODE(1,FST::RELATION('a', 2)), \

FST::NODE(1,FST::RELATION('i', 3)), \

FST::NODE(1,FST::RELATION('n', 4)), \

FST::NODE()

#define GRAPH\_STRDUPLICATE \

13, \

FST::NODE(1,FST::RELATION('s', 1)), \

FST::NODE(1,FST::RELATION('t', 2)), \

FST::NODE(1,FST::RELATION('r', 3)), \

FST::NODE(1,FST::RELATION('d', 4)), \

FST::NODE(1,FST::RELATION('u', 5)), \

FST::NODE(1,FST::RELATION('p', 6)), \

FST::NODE(1,FST::RELATION('l', 7)), \

FST::NODE(1,FST::RELATION('i', 8)), \

FST::NODE(1,FST::RELATION('c', 9)), \

FST::NODE(1,FST::RELATION('a', 10)), \

FST::NODE(1,FST::RELATION('t', 11)), \

FST::NODE(1,FST::RELATION('e', 12)), \

FST::NODE()

#define GRAPH\_STRLENGTH \

10, \

FST::NODE(1,FST::RELATION('s', 1)), \

FST::NODE(1,FST::RELATION('t', 2)), \

FST::NODE(1,FST::RELATION('r', 3)), \

FST::NODE(1,FST::RELATION('l', 4)), \

FST::NODE(1,FST::RELATION('e', 5)), \

FST::NODE(1,FST::RELATION('n', 6)), \

FST::NODE(1,FST::RELATION('g', 7)), \

FST::NODE(1,FST::RELATION('t', 8)), \

FST::NODE(1,FST::RELATION('h', 9)), \

FST::NODE()

#define GRAPH\_STRTRANSINT \

12, \

FST::NODE(1,FST::RELATION('s', 1)), \

FST::NODE(1,FST::RELATION('t', 2)), \

FST::NODE(1,FST::RELATION('r', 3)), \

FST::NODE(1,FST::RELATION('t', 4)), \

FST::NODE(1,FST::RELATION('r', 5)), \

FST::NODE(1,FST::RELATION('a', 6)), \

FST::NODE(1,FST::RELATION('n', 7)), \

FST::NODE(1,FST::RELATION('s', 8)), \

FST::NODE(1,FST::RELATION('i', 9)), \

FST::NODE(1,FST::RELATION('n', 10)), \

FST::NODE(1,FST::RELATION('t', 11)), \

FST::NODE()

// Ключевые слова

#define GRAPH\_RETURN \

7, \

FST::NODE(1,FST::RELATION('r', 1)), \

FST::NODE(1,FST::RELATION('e', 2)), \

FST::NODE(1,FST::RELATION('t', 3)), \

FST::NODE(1,FST::RELATION('u', 4)), \

FST::NODE(1,FST::RELATION('r', 5)), \

FST::NODE(1,FST::RELATION('n', 6)), \

FST::NODE()

#define GRAPH\_NEW \

4, \

FST::NODE(1,FST::RELATION('n', 1)), \

FST::NODE(1,FST::RELATION('e', 2)), \

FST::NODE(1,FST::RELATION('w', 3)), \

FST::NODE()

#define GRAPH\_FUNCTION \

9, \

FST::NODE(1,FST::RELATION('f', 1)), \

FST::NODE(1,FST::RELATION('u', 2)), \

FST::NODE(1,FST::RELATION('n', 3)), \

FST::NODE(1,FST::RELATION('c', 4)), \

FST::NODE(1,FST::RELATION('t', 5)), \

FST::NODE(1,FST::RELATION('i', 6)), \

FST::NODE(1,FST::RELATION('o', 7)), \

FST::NODE(1,FST::RELATION('n', 8)), \

FST::NODE()

// Операторы вывода и цикла

#define GRAPH\_WRITELINE \

10, \

FST::NODE(1,FST::RELATION('w', 1)), \

FST::NODE(1,FST::RELATION('r', 2)), \

FST::NODE(1,FST::RELATION('i', 3)), \

FST::NODE(1,FST::RELATION('t', 4)), \

FST::NODE(1,FST::RELATION('e', 5)), \

FST::NODE(1,FST::RELATION('l', 6)), \

FST::NODE(1,FST::RELATION('i', 7)), \

FST::NODE(1,FST::RELATION('n', 8)), \

FST::NODE(1,FST::RELATION('e', 9)), \

FST::NODE()

#define GRAPH\_WRITE \

6, \

FST::NODE(1,FST::RELATION('w', 1)), \

FST::NODE(1,FST::RELATION('r', 2)), \

FST::NODE(1,FST::RELATION('i', 3)), \

FST::NODE(1,FST::RELATION('t', 4)), \

FST::NODE(1,FST::RELATION('e', 5)), \

FST::NODE()

#define GRAPH\_CYCLE \

6, \

FST::NODE(1,FST::RELATION('c', 1)), \

FST::NODE(1,FST::RELATION('y', 2)), \

FST::NODE(1,FST::RELATION('c', 3)), \

FST::NODE(1,FST::RELATION('l', 4)), \

FST::NODE(1,FST::RELATION('e', 5)), \

FST::NODE()

// Скобки

#define GRAPH\_LEFTBRACE\_OPEN \

2, \

FST::NODE(1,FST::RELATION('{', 1)), \

FST::NODE()

#define GRAPH\_RIGHTBRACE\_CLOSE \

2, \

FST::NODE(1,FST::RELATION('}', 1)), \

FST::NODE()

#define GRAPH\_LEFTHESIS\_OPEN \

2, \

FST::NODE(1,FST::RELATION('(', 1)), \

FST::NODE()

#define GRAPH\_RIGHTHESIS\_CLOSE \

2, \

FST::NODE(1,FST::RELATION(')', 1)), \

FST::NODE()

// Арифметические операторы

#define GRAPH\_ADDITION \

2, \

FST::NODE(1,FST::RELATION('+', 1)), \

FST::NODE()

#define GRAPH\_SUBSTRACTION \

2, \

FST::NODE(1,FST::RELATION('-', 1)), \

FST::NODE()

#define GRAPH\_MULTIPLICATION \

2, \

FST::NODE(1,FST::RELATION('\*', 1)), \

FST::NODE()

#define GRAPH\_DIVISION \

2, \

FST::NODE(1,FST::RELATION('/', 1)), \

FST::NODE()

#define GRAPH\_REMAINDERDIVISION \

2, \

FST::NODE(1,FST::RELATION('%', 1)), \

FST::NODE()

#define GRAPH\_EQUALS \

2, \

FST::NODE(1,FST::RELATION('=', 1)), \

FST::NODE()

// Операторы сравнения

#define GRAPH\_MORE \

2, \

FST::NODE(1,FST::RELATION('>', 1)), \

FST::NODE()

#define GRAPH\_LESS \

2, \

FST::NODE(1,FST::RELATION('<', 1)), \

FST::NODE()

#define GRAPH\_MOREEQUAL \

3, \

FST::NODE(1,FST::RELATION('>', 1)), \

FST::NODE(1,FST::RELATION('=', 2)), \

FST::NODE()

#define GRAPH\_LESSEQUAL \

3, \

FST::NODE(1,FST::RELATION('<', 1)), \

FST::NODE(1,FST::RELATION('=', 2)), \

FST::NODE()

#define GRAPH\_INEQUALITY \

3, \

FST::NODE(1,FST::RELATION('=', 1)), \

FST::NODE(1,FST::RELATION('=', 2)), \

FST::NODE()

#define GRAPH\_NOTEQUALS \

3, \

FST::NODE(1,FST::RELATION('!', 1)), \

FST::NODE(1,FST::RELATION('=', 2)), \

FST::NODE()

// Разное

#define GRAPH\_COMMA \

2, \

FST::NODE(1,FST::RELATION(',', 1)), \

FST::NODE()

#define GRAPH\_SEMICOLON \

2, \

FST::NODE(1,FST::RELATION(';', 1)), \

FST::NODE()

Я думаю проблема заключается в создании токенов. Проверь очень внимательно:

Token.cpp:

#include "stdafx.h"

namespace Tokens

{

// Этап разбиения кода на токены

TokenTable Tokenize(In::IN in)

{

TokenTable tokens = CreateTokenTable(TOKEN\_MAXSIZE);

char buffer[256];

bool isNegativeNum = false;

int NumOfCharRecorded = 0;

int CurrentLine = 0;

int LinePosition = 0;

for (int CharPointer = 0; in.text[CharPointer] != IN\_CODE\_ENDL; CharPointer++)

{

if (NumOfCharRecorded == 256)

throw ERROR\_THROW\_IN(132, CurrentLine, LinePosition - NumOfCharRecorded);

// Проверка на комментарий (комментарий начинается с //...)

// комментарии пропускаем

if (in.text[CharPointer] == '/' && in.text[CharPointer + 1] == '/')

{

while (in.text[CharPointer + 1] != '\n')

CharPointer++;

continue;

}

// Обработка сепараторов

if (IsSeparator(in.text[CharPointer]))

{

if (in.text[CharPointer] == ' ' && isNegativeNum)

continue;

// создание токена ключевого слова

if (NumOfCharRecorded)

{

buffer[NumOfCharRecorded] = IN\_CODE\_ENDL;

AddToken(tokens, buffer, CurrentLine, LinePosition - NumOfCharRecorded, NumOfCharRecorded);

NumOfCharRecorded = 0;

isNegativeNum = false;

}

if (in.text[CharPointer] == '!' && in.text[CharPointer + 1] == '=')

{

buffer[0] = '!';

buffer[1] = '=';

buffer[2] = IN\_CODE\_ENDL;

AddToken(tokens, buffer, CurrentLine, LinePosition - NumOfCharRecorded, 2);

CharPointer++;

continue;

}

if (in.text[CharPointer] == '=' && in.text[CharPointer + 1] == '=')

{

buffer[0] = '=';

buffer[1] = '=';

buffer[2] = IN\_CODE\_ENDL;

AddToken(tokens, buffer, CurrentLine, LinePosition - NumOfCharRecorded, 2);

CharPointer++;

continue;

}

if (in.text[CharPointer] == '>' && in.text[CharPointer + 1] == '=')

{

buffer[0] = '>';

buffer[1] = '=';

buffer[2] = IN\_CODE\_ENDL;

AddToken(tokens, buffer, CurrentLine, LinePosition - NumOfCharRecorded, 2);

CharPointer++;

continue;

}

if (in.text[CharPointer] == '<' && in.text[CharPointer + 1] == '=')

{

buffer[0] = '<';

buffer[1] = '=';

buffer[2] = IN\_CODE\_ENDL;

AddToken(tokens, buffer, CurrentLine, LinePosition - NumOfCharRecorded, 2);

CharPointer++;

continue;

}

if (in.text[CharPointer] == '\n')

{

CurrentLine++;

LinePosition = 0;

continue;

}

if (in.text[CharPointer] == '-')

{

if (!isdigit(tokens.table[tokens.size - 1].token[tokens.table[tokens.size - 1].length - 1]) &&

!isalpha(tokens.table[tokens.size - 1].token[tokens.table[tokens.size - 1].length - 1]) &&

tokens.table[tokens.size - 1].token[tokens.table[tokens.size - 1].length - 1] != ')')

{

isNegativeNum = true;

buffer[NumOfCharRecorded] = in.text[CharPointer];

NumOfCharRecorded++;

LinePosition++;

continue;

}

}

// Создание токенов для операторов состоящих из одного символа

if (in.text[CharPointer] != ' ' && in.text[CharPointer] != '\t' && in.text[CharPointer] != '\n')

AddToken(tokens, SepToken(in.text[CharPointer]), CurrentLine, LinePosition - NumOfCharRecorded, 1);

LinePosition++;

continue;

}

// создание токена для строкового литерала

if (in.text[CharPointer] == '\"')

{

if (NumOfCharRecorded)

throw ERROR\_THROW\_IN(120, CurrentLine, LinePosition);

do

{

if (in.text[CharPointer] == '\n')

throw ERROR\_THROW\_IN(121, CurrentLine, 0);

if (NumOfCharRecorded == 256)

throw ERROR\_THROW\_IN(130, CurrentLine, 0);

buffer[NumOfCharRecorded] = in.text[CharPointer];

CharPointer++;

NumOfCharRecorded++;

LinePosition++;

} while (in.text[CharPointer] != '\"');

buffer[NumOfCharRecorded] = in.text[CharPointer];

NumOfCharRecorded++;

buffer[NumOfCharRecorded] = IN\_CODE\_ENDL;

AddToken(tokens, buffer, CurrentLine, LinePosition - NumOfCharRecorded, NumOfCharRecorded);

NumOfCharRecorded = 0;

continue;

}

// создание токена для литерала char

if (in.text[CharPointer] == '\'')

{

if (NumOfCharRecorded)

throw ERROR\_THROW\_IN(140, CurrentLine, LinePosition);

buffer[NumOfCharRecorded++] = in.text[CharPointer++]; // Добавляем открывающую кавычку

LinePosition++;

if (in.text[CharPointer] == '\\') // Обработка escape-последовательностей

{

buffer[NumOfCharRecorded++] = in.text[CharPointer++]; // Добавляем '\'

LinePosition++;

}

if (in.text[CharPointer] == '\n' || in.text[CharPointer] == IN\_CODE\_ENDL)

throw ERROR\_THROW\_IN(141, CurrentLine, LinePosition);

buffer[NumOfCharRecorded++] = in.text[CharPointer++]; // Добавляем символ

LinePosition++;

if (in.text[CharPointer] != '\'') // Закрывающая кавычка

throw ERROR\_THROW\_IN(142, CurrentLine, LinePosition);

buffer[NumOfCharRecorded++] = in.text[CharPointer]; // Добавляем закрывающую кавычку

buffer[NumOfCharRecorded] = IN\_CODE\_ENDL;

AddToken(tokens, buffer, CurrentLine, LinePosition - NumOfCharRecorded, NumOfCharRecorded);

NumOfCharRecorded = 0;

continue;

}

buffer[NumOfCharRecorded] = in.text[CharPointer];

NumOfCharRecorded++;

LinePosition++;

}

return tokens;

}

// Проверка на сепараторы

bool IsSeparator(char ch)

{

char separators[] = TOKEN\_SEPARATORS;

int index = 0;

while (separators[index] != IN\_CODE\_ENDL)

{

if (ch == separators[index])

return true;

index++;

}

return false;

}

// Функция создания токенов для операторов состоящих из одного символа

char\* SepToken(char sep)

{

char\* separator = new char[2];

separator[0] = sep;

separator[1] = IN\_CODE\_ENDL;

return separator;

}

// Функция создания таблицы токенов

TokenTable CreateTokenTable(int size)

{

if (size > TOKEN\_MAXSIZE)

throw ERROR\_THROW(118);

TokenTable tokentable;

tokentable.maxsize = size;

tokentable.size = 0;

tokentable.table = new Token[tokentable.maxsize];

return tokentable;

}

// Функция добавления токенов

void AddToken(TokenTable& tokens, char\* token, int line, int linePosition, int length)

{

if (tokens.size > tokens.maxsize)

throw ERROR\_THROW(119);

Token item;

strcpy(item.token, token);

item.line = line;

item.linePosition = linePosition;

item.length = length;

tokens.table[tokens.size] = item;

tokens.size++;

}

}

Token.h:

#pragma once

#include "stdafx.h"

#include "In.h"

#define TOKEN\_SEPARATORS " \t\n+-\*/%=<>(){},;!"

#define TOKEN\_MAXSIZE 4096

namespace Tokens

{

// Структура токена

struct Token

{

char token[258];

int length;

int line;

int linePosition;

};

// Структура таблицы токенов

struct TokenTable

{

int maxsize;

int size;

Token\* table;

};

// Создание таблицы токенов

TokenTable CreateTokenTable(int size);

// Добавление токенов

void AddToken(TokenTable& tokens, char\* token, int line, int linePosition, int length);

// Процесс представления кода в виде потока токенов

TokenTable Tokenize(In::IN in);

// Проверка на сепаратор

bool IsSeparator(char ch);

// Создание токена состоящего из одного символа

char\* SepToken(char sep);

}  
  
Вот текст файла входного параметра in:  
main

{

writeline "hi world!!";

};