Министерство науки и высшего образования Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №2

по дисциплине

«Языки программирования и методы трансляции»

Факультет прикладной математики и информатики

Группа ПМ-63

Студенты Майер В.

Шепрут И.

Преподаватели Еланцева И.Л.

Вариант 8

Новосибирск 2019

1. Цель работы

Изучить методы лексического анализа. Получить представление о методах обработки лексических ошибок. Научиться проектировать сканер на основе детерминиро

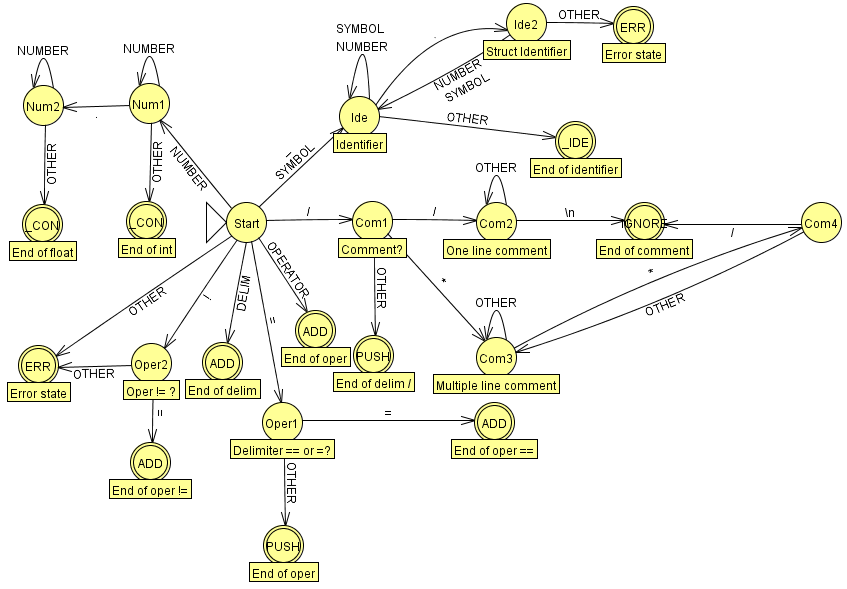
1. Задание

Подмножество языка С++ включает:

* данные типа int, float, struct из элементов указанных типов
* инструкции описания переменных
* операторы присваивания в любой последовательности
* операции +, -, \*, = =, !=, <, >.

В соответствии с выбранным вариантом задания к лабораторным работам разработать и реализовать лексический анализатор на основе детерминированных конечных автоматов. Исходными данными для сканера является программа на языке С++ и постоянные таблицы, реализованные в лабораторной работе №1. Результатом работы сканера является создание файла токенов, переменных таблиц (таблицы символов и таблицы констант) и файла сообщений об ошибках.

1. Детерминированный конечный автомат



Круги с двойным контуром обозначают конечное состояние, круги с одинарным контуром обозначают переходное. Треугольник показывает точку входа.

Состояния:

Start – начальное состояние детерминированного автомата

ADD – Конечное состояние без отката последнего символа

PUSH – Конечное состояние с откатом последнего символа

ERR – Конечное состояние при котором программа обрабатывает ошибку и вызывает исключение с описанием ошибки.

\_CON – Добавление в таблицу констант

\_IDE – Добавление в таблицу идентификаторов

IGNORE – Пропуск

Ide – состояние идентификатора.

Ide2 – состояние идентификатора структуры с точкой, но без поля (a.)

Com1 – знак /, после чего будет определен означает ли это знак делить или комментарий.

Com2 – однострочный комментарий.

Com3 – многострочный комментарий.

Com4 – состояние после символа \* в многострочном комментарии.

Del1 – переходное состояние между знаком равно или проверкой на равенство.

Del2 – переходное состояние между проверкой на неравенство.

Num1 – состояние целых чисел.

Num2 – состояние вещественных чисел.

1. **Алгоритм разбора**

Принцип алгоритма заключается в том чтобы, считывать последовательность символов по одному, считывание происходит до того момента, пока детерминированный автомат не достигнет одного из конечных состояний.

Если конечное состояние Push, IDE или CON, то происходит откат последнего символа, это нам позволяет точнее разделять на токены, и предотвращать некоторые ошибки на следующем уровне обработки.

Если конченое состояние IDE или CON будет добавлена одна из таблиц идентификаторов или констант

Структура будет добавляться в таблицу структур при синтаксическом анализе.

Идентификатор может состоять из точек, это позволяет хранить аргументы структуры в таблице идентификаторов, и возможность использовать структуру внутри структуры.

Алгоритм прекращает работу после первой ошибки.

***Подробный алгоритм:***

1. Считать строку, перейти к шагу 2.
2. Составить токен текущего слова, обнулить текущее слово, проверить следующий символ, если это буква перейти к шагу 3, если символ цифра перейти к шагу 5, если оператор(кроме /) или разделитель перейти к текущему состоянию, если знак равно, перейти в состояние 7, если восклицательный знак, топ перейти в состояние 8, если знак делить, то перейти в состояние 9, иначе перейти в состояние ERR.
3. Проверить следующий символ, если это буква или цифра, перейти к текущему состоянию, если точка, то к состоянию 4, если другой символ, перейти к состоянию 2, добавив текущее слово в таблицу идентификатор и перейти к предыдущему символу.
4. Если следующий символ буква или цифра перейти к состоянию 3, если точка, то к шагу ERR.
5. Проверить следующий символ, если это число, перейти к текущему состоянию, если точка перейти в состояние 6, иначе перейти к шагу 2, добавить текущее слово в таблицу констант(целое число) и перейти к предыдущему символу.
6. Проверить следующий символ, если это число, перейти к текущему состоянию, иначе перейти к шагу 2, добавить текущее слово в таблицу констант(действительное число) и перейти к предыдущему символу.
7. Если следующий символ ровно, то перейти состояние 2, если другое, то перейти в состояние 2 с переходом на предыдущий символ.
8. Если следующий символ ровно, то перейти состояние 2, если другое, то перейти в состояние ERR.
9. Если следующий символ, знак делить перейти в состояние 10, если умножить перейти в состояние 11, иначе перейти в состояние 2 с переходом к предыдущему символу.
10. Если следующий символ перенос строки, то перейти в состояние 2(без составления токена), иначе перейти к текущему состоянию.
11. Если следующий символ знак умножить перейти в состояние 12, иначе перейти к текущему состоянию.
12. Если следующий символ знак делить, перейти в состояние 2(без составления токена), иначе перейти в состояние 11.

ERR. Составить сообщение об ошибке из текущего слова и номера строки.

1. **Тесты**
2. *Лексически верный исходный код*

|  |  |
| --- | --- |
| Parsing of string:  struct a  {  float b;  };  int main()  {  int c;  c = 5+6\*(1/3);  float d;  a t;  t.b = 0.5;  d = 1564\*15.15 - 0.6464;  e = c != d - c == d;  /\* aoeusnthoaeu float: a\*b \*/  d = d;  return 0;  }  KEYWORDS.92.0 = "struct"  DELIMITERS.95.0 = " "  IDENTIFIERS.60.0 = "a"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.86.0 = "{"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  KEYWORDS.97.0 = "float"  DELIMITERS.95.0 = " "  IDENTIFIERS.61.0 = "b"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.88.0 = "}"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  KEYWORDS.18.0 = "int"  DELIMITERS.95.0 = " "  IDENTIFIERS.108.0 = "main"  DELIMITERS.3.0 = "("  DELIMITERS.4.0 = ")"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.86.0 = "{"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  DELIMITERS.72.0 = "TAB"  IDENTIFIERS.63.0 = "d"  DELIMITERS.95.0 = " "  OPERATIONS.24.0 = "="  DELIMITERS.95.0 = " "  CONSTANTS.247.0 = "1564"  OPERATIONS.5.0 = "\*"  CONSTANTS.25.0 = "15.15"  DELIMITERS.95.0 = " "  OPERATIONS.8.0 = "-"  DELIMITERS.95.0 = " "  CONSTANTS.401.0 = "0.6464"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  IDENTIFIERS.64.0 = "e"  DELIMITERS.95.0 = " "  OPERATIONS.24.0 = "="  DELIMITERS.95.0 = " "  IDENTIFIERS.62.0 = "c"  DELIMITERS.95.0 = " "  OPERATIONS.73.0 = "!="  DELIMITERS.95.0 = " "  IDENTIFIERS.63.0 = "d"  DELIMITERS.95.0 = " "  OPERATIONS.8.0 = "-"  DELIMITERS.95.0 = " " | KEYWORDS.18.0 = "int"  DELIMITERS.95.0 = " "  IDENTIFIERS.62.0 = "c"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  IDENTIFIERS.62.0 = "c"  DELIMITERS.95.0 = " "  OPERATIONS.24.0 = "="  DELIMITERS.95.0 = " "  CONSTANTS.16.0 = "5"  OPERATIONS.6.0 = "+"  CONSTANTS.17.0 = "6"  OPERATIONS.5.0 = "\*"  DELIMITERS.3.0 = "("  CONSTANTS.12.0 = "1"  OPERATIONS.10.0 = "/"  CONSTANTS.14.0 = "3"  DELIMITERS.4.0 = ")"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  DELIMITERS.95.0 = " "  KEYWORDS.97.0 = "float"  DELIMITERS.95.0 = " "  IDENTIFIERS.63.0 = "d"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  IDENTIFIERS.60.0 = "a"  DELIMITERS.95.0 = " "  IDENTIFIERS.79.0 = "t"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  IDENTIFIERS.291.0 = "t.b"  DELIMITERS.95.0 = " "  OPERATIONS.24.0 = "="  DELIMITERS.95.0 = " "  CONSTANTS.154.0 = "0.5"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  IDENTIFIERS.62.0 = "c"  DELIMITERS.95.0 = " "  OPERATIONS.9.0 = "=="  DELIMITERS.95.0 = " "  IDENTIFIERS.63.0 = "d"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  IDENTIFIERS.63.0 = "d"  DELIMITERS.95.0 = " "  OPERATIONS.24.0 = "="  DELIMITERS.95.0 = " "  IDENTIFIERS.63.0 = "d"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.72.0 = "TAB"  KEYWORDS.71.0 = "return"  DELIMITERS.95.0 = " "  CONSTANTS.11.0 = "0"  DELIMITERS.22.0 = ";"  DELIMITERS.73.0 = "ENTER"  DELIMITERS.88.0 = "}"  Для продолжения нажмите любую .клавишу . . . |

1. *Использование неверных символов*

Parsing of string:

struct a

{

float ~b;

};

int main()

{

int c;

c = 5+6\*(1/3);

float d;

d = 1564\*15.15 - 0.6464;

e = c != d - c == d;

/\* aoeusnthoaeu float: a\*b \*/

d = d;

return 0;

}

Error around symbol "~"

In 3 line

Для продолжения нажмите любую клавишу . . .

1. *Лексически неверный код*

*Parsing of string:*

*struct a*

*{*

*float b;*

*};*

*int main()*

*{*

*int c;*

*c = 5+6\*(1/3);*

*float d;*

*d = 1564\*15.15 - 0.6464;*

*e = c !!= d - c == d;*

*/\* aoeusnthoaeu float: a\*b \*/*

*d = d;*

*return 0;*

*}*

*Error around symbol "!!"*

*In 11 line*

*Для продолжения нажмите любую клавишу . . .*

1. **Текст программы**

/\* lex\_analyzer.h \*/

#pragma once

#include <vector>

#include <tables.h>

std::vector<Token> parse(const std::string& str, Tables& tables);

/\* lex\_analyzer.cpp \*/

#include <string>

#include <lex\_analyzer.h>

#include <string>

#include <lex\_analyzer.h>

enum AddType

{

ADD\_IDENTIFIER,

ADD\_CONSTANT,

ADD\_KEYWORD\_OR\_DELIMITER,

IGNORE,

ERROR,

};

enum StateType

{

Start, // Стартовое состояние

// Завершающие состояния, при достижении их, конечный автомат должен завершать свою работу

ADD, // Все считанные символы, начиная со Start, до текущего момента ялвяются лексемой

PUSH, // ADD + Поместить предыдущий считанный символ обратно в ввод

NEXT, // Данная лексема бесполезна для трансляции, и её можно игнорировать

ERR, // Ошибочное состояние

\_IDE, // PUSH, только для таблицы идентификаторов

\_CONS, // PUSH, только для таблицы констант

// Промежуточные состояния

Ide, Ide2,

Com1, Com2, Com3, Com4,

Num1, Num2,

Oper1, Oper2,

};

enum SymbolType

{

SYMBOL, // "\_a-zA-Z"

NUMBER, // "0-9"

DELIM, // " \t;,{}()"

OPER, // "+-<>"

DOT, // "."

EXCL, // "!"

EQUAL, // "="

DIV, // "/"

MUL, // "\*"

EOLN, // "\n"

OTHER,

};

SymbolType getType(char symbol);

// Здесь надо запрогать конечный автомат, который по текущему состоянию и считанному символу возвращает следующее состояние

StateType automatonNext(StateType state, SymbolType symbol);

class Automaton

{

public:

Automaton(const std::string& str);

bool next(void);

bool isEndState(void) const;

AddType getState(void) const;

std::string getStr(void) const;

int getPos(void) const;

int getLine(void) const;

private:

const std::string& str;

int i, n;

std::string sum;

StateType state;

};

//=============================================================================

//=============================================================================

//=============================================================================

//-----------------------------------------------------------------------------

SymbolType getType(char c) {

if (c == '.') return DOT;

if (c == '!') return EXCL;

if (c == '=') return EQUAL;

if (c == '/') return DIV;

if (c == '\*') return MUL;

if (c == '\n') return EOLN;

if (c == '\_' || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

return SYMBOL;

if (c >= '0' && c <= '9')

return NUMBER;

if (c == ' ' || c == '\t' || c == ';' || c == ',' || c == '{' || c == '}' || c == '(' || c == ')')

return DELIM;

if (c == '+' || c == '-' || c == '<' || c == '>')

return OPER;

return OTHER;

}

//-----------------------------------------------------------------------------

StateType automatonNext(StateType state, SymbolType sym) {

switch (state) {

case Start: {

if (sym == EQUAL) return Oper1;

if (sym == EXCL) return Oper2;

if (sym == SYMBOL) return Ide;

if (sym == DELIM) return ADD;

if (sym == OPER) return ADD;

if (sym == NUMBER) return Num1;

if (sym == DIV) return Com1;

if (sym == DOT) return ERR;

if (sym == MUL) return ADD;

if (sym == EOLN) return ADD;

return ERR;

} break;

case Ide: {

if (sym == SYMBOL) return Ide;

if (sym == NUMBER) return Ide;

if (sym == DOT) return Ide2;

return \_IDE;

} break;

case Ide2: {

if (sym == SYMBOL) return Ide;

if (sym == NUMBER) return Ide;

return ERR;

} break;

case Com1: {

if (sym == DIV) return Com2;

if (sym == MUL) return Com3;

return PUSH;

} break;

case Com2: {

if (sym == EOLN) return NEXT;

return Com2;

} break;

case Com3: {

if (sym == MUL) return Com4;

return Com3;

} break;

case Com4: {

if (sym == DIV) return NEXT;

return Com3;

} break;

case Num1: {

if (sym == NUMBER) return Num1;

if (sym == DOT) return Num2;

return \_CONS;

} break;

case Num2: {

if (sym == NUMBER) return Num2;

return \_CONS;

} break;

case Oper1: {

if (sym == EQUAL) return ADD;

return PUSH;

} break;

case Oper2: {

if (sym == EQUAL) return ADD;

return ERR;

} break;

case ADD:

case PUSH:

case ERR:

case \_IDE:

case \_CONS: {

throw std::exception("That is final state!");

} break;

}

}

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

Automaton::Automaton(const std::string& str) : str(str), i(0), n(1), state(Start) {

}

//-----------------------------------------------------------------------------

bool Automaton::next(void) {

if (state == PUSH || state == \_IDE || state == \_CONS)

if (str[--i] == '\n') n--;

if (isEndState()) {

state = Start;

sum = "";

}

char c = str[i];

//Надо либо automatonNext засовывать внуторь класса Automation, либо так

if (c == '\n') n++;

state = automatonNext(state, getType(c));

sum += c;

i++;

return i != str.size();

}

//-----------------------------------------------------------------------------

bool Automaton::isEndState(void) const {

return

state == ADD ||

state == PUSH ||

state == NEXT ||

state == ERR ||

state == \_IDE ||

state == \_CONS;

}

//-----------------------------------------------------------------------------

AddType Automaton::getState(void) const {

if (state == ADD || state == PUSH)

return ADD\_KEYWORD\_OR\_DELIMITER;

if (state == \_IDE)

return ADD\_IDENTIFIER;

if (state == \_CONS)

return ADD\_CONSTANT;

if (state == NEXT)

return IGNORE;

if (state == ERR)

return ERROR;

return ERROR;

}

//-----------------------------------------------------------------------------

std::string Automaton::getStr(void) const {

auto result = sum;

if (state == PUSH || state == \_IDE || state == \_CONS)

result.pop\_back();

return result;

}

//-----------------------------------------------------------------------------

int Automaton::getPos(void) const {

return i;

}

int Automaton::getLine(void) const {

return n;

}

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

std::vector<Token> parse(const std::string& str, Tables& tables) {

std::vector<Token> result;

Automaton automaton(str);

while (automaton.next()) {

if (automaton.isEndState()) {

auto to\_add = automaton.getStr();

switch (automaton.getState()) {

case ADD\_IDENTIFIER: {

auto token = tables.find(to\_add);

if (token) {

result.push\_back(token);

} else {

result.push\_back(tables.add(to\_add, TABLE\_IDENTIFIERS));

}

} break;

case ADD\_CONSTANT: {

auto token = tables.find(to\_add);

if (token) {

result.push\_back(token);

} else {

result.push\_back(tables.add(to\_add, TABLE\_CONSTANTS));

}

} break;

case ADD\_KEYWORD\_OR\_DELIMITER: {

auto token = tables.find(to\_add);

if (token) {

result.push\_back(token);

} else {

// Такого быть не может

throw std::exception();

}

} break;

case IGNORE: {} break;

case ERROR: {

// Надо как-то обработать эту ошибку

std::string error = "Error around symbol \"" + to\_add +

"\"\nIn " + std::to\_string(automaton.getLine()) + " line";

throw error; // std::exception();

break;}

}

}

}

return result;

}

/\*tables.h\*/

#pragma once

#include <vector>

#include <string>

#include <functional>

#include <hash\_table.h>

using namespace std;

enum TableType

{

TABLE\_NULL,

// Статические таблицы

TABLE\_KEYWORDS,

TABLE\_OPERATIONS,

TABLE\_DELIMITERS,

// Динамические таблицы

TABLE\_IDENTIFIERS,

TABLE\_CONSTANTS,

TABLE\_STRUCTURES,

};

struct Token

{

TableType table;

hash\_table\_pos pos;

operator bool() const {

return bool(pos);

}

};

std::ostream& operator<<(std::ostream& out, const Token& t);

enum Type

{

TYPE\_INT,

TYPE\_FLOAT,

TYPE\_STRUCT // Используется только в StructElem

};

struct Identifier

{

Type type;

bool isInitialized;

};

struct Constant

{

Type type;

union {

float f;

int i;

} value;

};

struct Structure

{

struct StructElem

{

Type type;

std::string name;

Token structToken;

};

std::vector<StructElem> elems;

};

class Tables

{

public:

Tables();

// Удаление из таблицы идентификаторов нужно потому что на этапе лексического анализа невозможно отличить структуры от имен идентификаторов. И все имена, даже имена структур будут заноситься в таблицу идентификаторов

void deleteFromIdentifiers(const Token& token);

Token find(const std::string& str) const;

Token add(const std::string& str, const TableType& type);

std::string getStr(const Token& token) const;

template<class T>

T& get(const Token& token);

template<class T>

const T& get(const Token& token) const;

private:

struct void\_struct {};

hash\_table<void\_struct> m\_keywords;

hash\_table<void\_struct> m\_delimiters;

hash\_table<void\_struct> m\_operations;

hash\_table<Identifier> m\_identifiers;

hash\_table<Constant> m\_constants;

hash\_table<Structure> m\_structures;

};

//=============================================================================

//=============================================================================

//=============================================================================

//-----------------------------------------------------------------------------

template<class T>

T& Tables::get(const Token& token) {

if constexpr (std::is\_same<T, Identifier>::value)

return m\_identifiers[token.pos];

if constexpr (std::is\_same<T, Constant>::value)

return m\_constants[token.pos];

if constexpr (std::is\_same<T, Structure>::value)

return m\_structures[token.pos];

}

//-----------------------------------------------------------------------------

template<class T>

const T& Tables::get(const Token& token) const {

if constexpr (std::is\_same<T, Identifier>::value)

return m\_identifiers[token.pos];

if constexpr (std::is\_same<T, Constant>::value)

return m\_constants[token.pos];

if constexpr (std::is\_same<T, Structure>::value)

return m\_structures[token.pos];

}

/\*tables.cpp\*/

#include <sstream>

#include <tables.h>

Tables::Tables() :

m\_keywords(100),

m\_delimiters(100),

m\_operations(100),

m\_identifiers(500),

m\_constants(500),

m\_structures(500) {

{

std::vector<string> add = {"int", "float", "struct"};

for (auto& i : add)

m\_keywords.add(i, {});

}

{

std::vector<string> add = {" ", ";", "\n", "\t", "{", "}", "(", ")"};

for (auto& i : add)

m\_delimiters.add(i, {});

}

{

std::vector<string> add = {"=", "+", "-", "\*", "/", "==", "!=", "<", ">"};

for (auto& i : add)

m\_operations.add(i, {});

}

}

//-----------------------------------------------------------------------------

void Tables::deleteFromIdentifiers(const Token& token) {

if (token.table != TABLE\_IDENTIFIERS)

throw std::exception();

m\_identifiers.erase(token.pos);

}

//-----------------------------------------------------------------------------

Token Tables::find(const std::string& str) const {

{ auto token = m\_keywords.find(str); if (token) return {TABLE\_KEYWORDS, token}; }

{ auto token = m\_delimiters.find(str); if (token) return {TABLE\_DELIMITERS, token}; }

{ auto token = m\_operations.find(str); if (token) return {TABLE\_OPERATIONS, token}; }

{ auto token = m\_identifiers.find(str); if (token) return {TABLE\_IDENTIFIERS, token}; }

{ auto token = m\_constants.find(str); if (token) return {TABLE\_CONSTANTS, token}; }

{ auto token = m\_structures.find(str); if (token) return {TABLE\_STRUCTURES, token}; }

return {TABLE\_NULL, hash\_table\_pos::getNullPos()};

}

//-----------------------------------------------------------------------------

Token Tables::add(const std::string& str, const TableType& type) {

switch (type) {

case TABLE\_IDENTIFIERS: return {TABLE\_IDENTIFIERS, m\_identifiers.add(str, {})}; break;

case TABLE\_CONSTANTS: return {TABLE\_CONSTANTS, m\_constants.add(str, {})}; break;

case TABLE\_STRUCTURES: return {TABLE\_STRUCTURES, m\_structures.add(str, {})}; break;

}

}

//-----------------------------------------------------------------------------

std::string Tables::getStr(const Token& token) const {

switch (token.table) {

case TABLE\_KEYWORDS: return m\_keywords.str(token.pos); break;

case TABLE\_OPERATIONS: return m\_operations.str(token.pos); break;

case TABLE\_DELIMITERS: return m\_delimiters.str(token.pos); break;

case TABLE\_IDENTIFIERS: return m\_identifiers.str(token.pos); break;

case TABLE\_CONSTANTS: return m\_constants.str(token.pos); break;

case TABLE\_STRUCTURES: return m\_structures.str(token.pos); break;

}

}

//-----------------------------------------------------------------------------

std::ostream& operator<<(std::ostream& out, const Token& t) {

std::stringstream sout;

if (t) {

switch (t.table) {

case TABLE\_KEYWORDS: sout << "KEYWORDS."; break;

case TABLE\_OPERATIONS: sout << "OPERATIONS."; break;

case TABLE\_DELIMITERS: sout << "DELIMITERS."; break;

case TABLE\_IDENTIFIERS: sout << "IDENTIFIERS."; break;

case TABLE\_CONSTANTS: sout << "CONSTANTS."; break;

case TABLE\_STRUCTURES: sout << "STRUCTURES."; break;

}

sout << t.pos.line << "." << t.pos.pos;

} else {

sout << "not exists";

}

out << sout.str();

return out;

}

/\*hash\_table.h\*/

#pragma once

#include <functional>

#include <vector>

#include <list>

// Хеш-функция. https://sohabr.net/post/219139/

struct HashH37

{

size\_t operator()(const std::string& str) const {

size\_t hash = 2139062143;

for (auto& i : str)

hash = 37 \* hash + size\_t(i);

return hash;

}

};

// Позиция в хеш-таблице

struct hash\_table\_pos

{

size\_t line;

size\_t pos;

// Возвращает true, если позиция действительная, и false если такой позиции не существует

operator bool() const {

return !(line == std::numeric\_limits<size\_t>::max() && pos == line);

}

static hash\_table\_pos getNullPos() {

return {std::numeric\_limits<size\_t>::max(), std::numeric\_limits<size\_t>::max()};

}

};

// Хеш-таблица. Можно обращаться как по строке, так и по позиции.

//template<typename T, typename F = std::hash<std::string>>

template<typename T, typename F = HashH37>

class hash\_table

{

public:

hash\_table(int size) : m\_table(size), m\_size(size) {

}

hash\_table\_pos add(const std::string& str, const T& t) {

if (find(str))

throw std::exception();

size\_t line = f(str) % m\_size;

m\_table[line].push\_back({str, t});

return {line, m\_table[line].size() - 1};

}

hash\_table\_pos find(const std::string& str) const {

size\_t line = f(str) % m\_size;

auto& list = m\_table[line];

for (auto i = list.begin(); i != list.end(); ++i) {

if (i->first == str) {

size\_t pos = std::distance(list.begin(), i);

return {line, pos};

}

}

return hash\_table\_pos::getNullPos();

}

void erase(const hash\_table\_pos& pos) {

m\_table[pos.line].erase(std::next(m\_table[pos.line].begin(), pos.pos));

}

std::string str(const hash\_table\_pos& pos) const {

return std::next(m\_table[pos.line].begin(), pos.pos)->first;

}

T& operator[](const std::string& str) {

auto pos = find(str);

return operator[](pos);

}

const T& operator[](const std::string& str) const {

auto pos = find(str);

return operator[](pos);

}

T& operator[](const hash\_table\_pos& pos) {

return std::next(m\_table[pos.line].begin(), pos.pos)->second;

}

const T& operator[](const hash\_table\_pos& pos) const {

return std::next(m\_table[pos.line].begin(), pos.pos)->second;

}

private:

std::vector<std::list<std::pair<std::string, T>>> m\_table;

int m\_size;

F f;

};

/\*test\_lex\_analyzer.cpp\*/

#include <iostream>

#include <iomanip>

#include <lex\_analyzer.h>

int main() {

std::string file = "struct a\n{\n\tfloat b;\n};\nint main()\n{\n\tint c;\n\tc = 5+6\*(1/3);\n\t float d;\n\ta t;\n\tt.b = 0.5;\n\td = 1564\*15.15 - 0.6464;\n\te = c != d - c == d;\n\t/\* aoeusnthoaeu float: a\*b \*/\n\td = d;\n\treturn 0;\n}";

file += ' ';

std::cout << "Parsing of string: \n" << file << std::endl;

try

{

Tables tables;

auto result = parse(file, tables);

string str;

for (auto& i : result)

{

str = tables.getStr(i);

if (str == "\n")

str = "ENTER";

if (str == "\t")

str = "TAB";

std::cout << std::setw(18) << std::left << i << " = \"" << str << "\"" << std::endl;

}

}

catch (string &e)

{

std::cerr << e << endl;

}

system("pause");

}