Министерство образования и науки РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторная работа №1

по языкам программирования и методам трансляции

Факультет: ПМИ

Группа: ПМ-01

Студент: Конев А.М., Ряховский М.И.

Вариант: 6

Преподаватель: Еланцева И.Л., Полетаева И.А.

Новосибирск

2013

# Цель работы

Получить представление о видах таблиц, используемых при трансляции программ. Изучить множество операций с таблицами и особенности реализации этих операций для таблиц, используемых на этапе лексического анализа. Реализовать классы таблиц, используемых сканером.

# Документация классов

## 2.1 Класс для работы с постоянными таблицами

*Имя класса*: const\_table

*Примечание*: в силу того, что постоянные таблицы могут иметь разную природу, класс является шаблоном. Параметр шаблона Table\_type – тип элементов таблицы. Например, символы или строки.

*Тип структуры данных*: упорядоченное множество.

Методы класса:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя метода | Входные данные | Описание | Пример использования |
| const\_table | string f\_name | Конструктор, генерирует таблицу из файла с именем f\_name. Если файл успешно открыт возвращает true, иначе - false | const\_table<char> char\_t("table1.txt"); |
| generate\_from\_file | string f\_name | Генерирует таблицу из файла с именем f\_name | char\_t.generate\_from\_file("table1.txt"); |
| add\_element | Table\_type add\_elem | Добавляет в таблицу один элемент. | char\_t.add\_element('a'); |
| is\_element\_in\_table | Table\_type check | Проверяет наличие элемента в таблице. Возвращает true, если элемент есть в таблице, false- иначе. | char\_t. is\_element\_in\_table ('a'); |
| get\_elemen\_number | Table\_type check\_el, int& num | Возвращает в num номер искомого элемента. Если элемента в таблице нет функция возвращает false, иначе – true. | string\_t.get\_elemen\_number("int", n0); |
| get\_elemen\_value | int num, Table\_type& gets\_el | Возвращение элемента gets\_el по номеру num. Если элемента в таблице нет функция возвращает false, иначе – true. | string\_t.get\_elemen\_value(3, get\_s); |
| ~const\_table | - | Деструктор класса, отчищает таблицу | - |

Атрибутом класса является множество set<Table\_type> our\_table, которое и является упорядоченной таблицей.

## Класс для работы с переменными таблицами

*Имя класса*: variable\_table

*Тип структуры данных*: хэш-таблица с цепочками, каждая цепочка представляет собой не упорядоченное множество.

Методы класса:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя метода | Входные данные | Описание | Пример использования |
| variable\_table | - | Конструктор по умолчанию, создаёт хэш-таблицу | - |
| add\_ind | string ind\_name | Добавляет элемент с идентификатором ind\_name в таблицу. Возвращает false если элемент уже есть в таблице, иначе – true. | ind\_t.add\_ind("a"); |
| set\_ind\_type | string ind\_name, int type | Устанавливает значение типа для идентификатора ind\_name. Возвращает false, если элемента нет в таблице, true – иначе. | ind\_t.add\_ind\_type("ab", 2); |
| set\_ind\_val | string ind\_name, bool val | Устанавливает, инициализирован ли указанный идентификатор. Возвращает false, если элемента нет в таблице, true – иначе. | ind\_t.add\_ind\_val("a", true); |
| check\_ind\_in\_table | string ind\_name | Проверяет, есть ли указанный идентификатор в таблице. Возвращает true если есть, false – если нет. | check\_ind\_in\_table(ind\_name) |
| ind\_adrress | string ind\_name, int& hash, int& numb | Возвращает полный адрес идентификатора – номер цепочки и номер в цепочке. Возвращает false, если элемента нет в таблице. | int n1, n2;  ind\_t.ind\_adrress("a", n1, n2); |
| get\_lexeme | string ind\_name, lexeme& gets\_lexeme | Возвращает лексему по идентификатору. Возвращает false, если элемента нет в таблице. | vars\_t.get\_lexeme("aab", t1); |
| get\_lexeme | int hash, int num, lexeme& gets\_lexeme | Возвращает лексему по полному адресу. Возвращает false, если элемента нет в таблице. | vars\_t.get\_lexeme(n1, n2, t2); |
| set\_type\_address | int hash, int numb, int type | Изменяет значение типа лексемы, по полному адресу в таблице. Возвращает false, если элемента нет в таблице. | vars\_t.set\_type\_address(n1,n2, 3); |
| set\_value\_address | int hash, int numb, bool val | Изменяет значение инициализации лексемы, по полному адресу в таблице. Возвращает false, если элемента нет в таблице. | vars\_t.set\_value\_address(n1,n2, true); |
| calc\_hash | string ind\_name | Возвращает хэш для идентификатора, который одновременно является номером цепочки, в которой идентификатор хранится. Имеет статус private. | int ind\_hash = calc\_hash(ind\_name); |
| find\_in\_vector | int hash, string ind\_name | Возвращает номер элемента с идентификатором ind\_name в векторе с номером hash. Если он там отсутствует, возвращается -1. Имеет статус private. | int ind\_num = find\_in\_vector(ind\_hash, ind\_name); |
| ~variable\_table | - | Деструктор класса, отчищает таблицу | - |

Атрибутом класса является массив векторов (цепочек) vector<lexeme> \*our\_table.

# Функция хеширования

Вычисления хэша производится по первому символу идентификатора, в следующем порядке: сначала идут цифры, затем строчные буквы латинского алфавита, затем заглавные буквы латинского алфавита.

# Исходные данные

Исходными данными для постоянных таблиц является файл, где перечислены элементы этих таблиц. Объём файла ограничивается техническими возможностями компьютера. В файле каждый идентификатор начинается с новой строки.

Для переменных таблиц явных исходных данных нет. Они формируются динамически.

# Тесты

## Тест для постоянных таблиц

Исходный файл с именем “table2.txt”:

int

void

main

float

char

|  |  |  |
| --- | --- | --- |
| Шаг | Описание | Результат |
| const\_table<string> string\_t("table2.txt"); | Создание таблицы из файла | our\_table[5]("char","float","int","main","void")  [0] "char"  [1] "float"  [2] "int"  [3] "main"  [4] "void" |
| string\_t.add\_element("if"); | Добавление элемента в таблицу | our\_table [6]("char","float","if","int","main","void")  [0] "char"  [1] "float"  [2] "if"  [3] "int"  [4] "main"  [5] "void" |
| bool check1 = string\_t.is\_element\_in\_table("int");  bool check2 = string\_t.is\_element\_in\_table("double"); | Проверка наличия элементов в таблице | check1 true  check2 false |
| int n0;  string\_t.get\_elemen\_number("int", n0);  string get\_s;  string\_t.get\_elemen\_value(n0, get\_s); | Нахождение элемента по номеру и по значению | n0 3  get\_s "int" |

## Тест для переменных таблиц

|  |  |  |
| --- | --- | --- |
| Шаг | Описание | Результат |
| vars\_t.add\_ind("a");  vars\_t.add\_ind("b");  vars\_t.add\_ind("aab"); | Добавление элементов в таблицу | vars\_t.our\_table[10]  [0] {name="a" type=0 value=false }  [1] {name="aab" type=0 value=false }  vars\_t.our\_table[11]  [0] {name="b" type=0 value=false } |
| bool check3 = vars\_t.add\_ind("a") | Попытка повторного добавления в таблицу | check3 false  vars\_t.our\_table[10]  [0]{name="a" type=0 value=false }  [1]{name="aab" type=0 value=false } |
| bool check4 = vars\_t.check\_ind\_in\_table("aab");  bool check5 = vars\_t.check\_ind\_in\_table("c"); | Проверка наличия элементов в таблице | check4 true  check5 false |
| int n1, n2;  vars\_t.ind\_adrress("b",n1, n2); | Получение адреса элемента | n1 11  n2 0 |
| vars\_t.add\_ind\_type("a", 1);  vars\_t.add\_ind\_val("aab", true); | Установка значений | vars\_t.our\_table[10]  [0]{name="a" type=1 value=false }  [1]{name="aab" type=0 value=true } |
| lexeme t1, t2;  vars\_t.get\_lexeme("aab", t1);  vars\_t.get\_lexeme(n1, n2, t2); | Получение токена | t1 {name="aab" type=0 value=true }  t2 {name="b" type=0 value=false } |

# Текст программы

## Файл «lexeme.h»

#pragma once

#include <string>

using namespace std;

//Стуркутра, являющаяся програмным предстовлением токена.

struct lexeme{

string name; //имя индендификатора

int type;

//Тип индендификатора

//0 - не опрерделён

//1 - int

//2 - char

//3 - float

bool value; //Инциализирован ли индедификатор. true - инициализирован, false - нет.

//Констукторы, для уменьшения кода

lexeme(){

}

lexeme(string ind\_name){

name = ind\_name;

type = 0;

value = false;

}

//Для сета

bool operator < (const lexeme& b) const{

return name < b.name;

}

bool operator == (lexeme b){

return name == b.name;

}

};

## Файл «const\_table.h»

#pragma once

#include "lexeme.h"

#include <iostream>

#include <fstream>

#include <set>

//Класс, релазиующий хранение постоянных таблиц

//Поскольку у нас могут быть как целые строки, так и символы чтобы не тратить память - делаем шаблон

template <typename Table\_type> class const\_table{

public:

//Два конструктора. Скучный - по умлочанию, и для генерации из файла

const\_table(){

}

const\_table(string f\_name){

generate\_from\_file(f\_name);

}

//И деструктор, для приличия

~const\_table(){

our\_table.clear();

}

//Генерирует таблицу из файла, возаращает true если файл открыли, false если нет

bool generate\_from\_file(string f\_name){

ifstream file\_stream(f\_name.c\_str()); //поток чтения из файла

if(file\_stream.is\_open()){

Table\_type read\_el; //считываемый элемент

while(!file\_stream.eof()){

file\_stream >> read\_el;

our\_table.insert(read\_el);

};

file\_stream.close();

return true;

}

else

return false;

}

//Добавление одного элемента в табилцу. Малоли, вдруг пригодится.

void add\_element(Table\_type add\_elem){

our\_table.insert(add\_elem);

}

//Проверка - есть ли элемент в таблицу, true если есть, false если нет.

bool is\_element\_in\_table(Table\_type check\_el){

set<Table\_type>::iterator el\_numb = our\_table.find(check\_el); // находим номер элемента

return el\_numb != our\_table.end() ? true : false; //элемента нет в таблице, если его "номер" равен концу структуры...STL же

}

//Поиск номера элемента. Если такого нет - false.

bool get\_elemen\_number(Table\_type check\_el, int& num){

if(is\_element\_in\_table(check\_el)){

num = distance(our\_table.begin(), our\_table.find(check\_el));

return true;

}

else

return false;

}

//Поиск элемента по номеру. Если такого нет - false.

bool get\_elemen\_value(int num, Table\_type& gets\_el){

if(num >= 0 && num < our\_table.size()){

set<Table\_type>::iterator el\_iter = our\_table.begin();

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

el\_iter++;

gets\_el = \*el\_iter;

return true;

}

else

return false;

}

private:

set<Table\_type> our\_table; //собственно, наша табличка

};

## Файл «variable\_table.h»

#pragma once

#include "const\_table.h"

#include <vector>

//Класс, для хранения переменных таблиц, в виде хеш-таблиц

//Реализация в виде хеш-таблицы с цепочками. Хеширование производится по первому символу индендификатора

const int hash\_N = 26\*2 + 10; //Максимальное значение хэша

class variable\_table{

public:

//Конструктор, для выделения нужной памяти

variable\_table(){

our\_table = new vector<lexeme> [hash\_N];

}

//Деструктор

~variable\_table(){

for(int i = 0; i < hash\_N; i++)

our\_table[i].clear();

delete[] our\_table;

}

bool add\_ind(string ind\_name); //Функция добавления индендификатора, true - если добавили, false - не добавили.

bool set\_ind\_type(string ind\_name, int type); //Функция добавления значения типа индендификатору.

bool set\_ind\_val(string ind\_name, bool val); //Функция измения типо инициализации индендификатора.

bool check\_ind\_in\_table(string ind\_name); //Проверка, входит ли индендификатор в таблицу, true - входит, false - нет.

bool ind\_adrress(string ind\_name, int& hash, int& numb); //Возврашает полный адрес идентификатора, либо false если его нет в таблице.

bool get\_lexeme(string ind\_name, lexeme& gets\_lexeme); //Получаем токен по идентификатору, если такого нет - false.

bool get\_lexeme(int hash, int numb, lexeme& gets\_lexeme); //Получаем токен по его полному адресу, если такого нет - false.

bool set\_type\_address(int hash, int numb, int type); //Установка типа по полному адресу

bool set\_value\_address(int hash, int numb, bool val); //Установка значения по полному адресу

private:

vector<lexeme> \*our\_table; //наша табличка

int calc\_hash(string ind\_name); //Функция вычисления хэша. Сначала - числа, потом строчные буквы, потом заглавные

int find\_in\_vector(int hash, string ind\_name);

## };

## Файл «variable\_table.cpp»

#include "variable\_table.h"

bool variable\_table::add\_ind(string ind\_name){

int ind\_hash = calc\_hash(ind\_name);

bool check = check\_ind\_in\_table(ind\_name);

if(!check){

our\_table[ind\_hash].push\_back(lexeme(ind\_name));

return true;

}

else

return false;

}

bool variable\_table::set\_ind\_type(std::string ind\_name, int type){

int ind\_hash = calc\_hash(ind\_name);

if(check\_ind\_in\_table(ind\_name)){

int ind\_num = find\_in\_vector(ind\_hash, ind\_name); //ищим наш элемент

our\_table[ind\_hash][ind\_num].type = type;

return true;

}

else

return false;

}

bool variable\_table::set\_ind\_val(std::string ind\_name, bool val){

int ind\_hash = calc\_hash(ind\_name);

if(check\_ind\_in\_table(ind\_name)){

int ind\_num = find\_in\_vector(ind\_hash, ind\_name); //ищим наш элемент

our\_table[ind\_hash][ind\_num].value = val;

return true;

}

else

return false;

}

bool variable\_table::set\_type\_address(int hash, int numb, int type){

if(hash >= 0 && hash < hash\_N && numb >= 0 && numb < our\_table[hash].size()){

our\_table[hash][numb].type = type;

return true;

}

return false;

}

bool variable\_table::set\_value\_address(int hash, int numb, bool val){

if(hash >= 0 && hash < hash\_N && numb >= 0 && numb < our\_table[hash].size()){

our\_table[hash][numb].value = val;

return true;

}

return false;

}

bool variable\_table::check\_ind\_in\_table(string ind\_name){

int ind\_hash = calc\_hash(ind\_name);

int num = find\_in\_vector(ind\_hash,ind\_name); //номер элемента

return num != -1 ? true : false;

}

int variable\_table::find\_in\_vector(int hash, string ind\_name){

lexeme find\_t(ind\_name);

for(int i = 0; i < our\_table[hash].size(); i++)

if(find\_t == our\_table[hash][i])

return i;

return -1;

}

bool variable\_table::ind\_adrress(string ind\_name, int& hash, int& numb){

hash = calc\_hash(ind\_name);

numb = find\_in\_vector(hash, ind\_name);

return numb != -1 ? true : false;

}

bool variable\_table::get\_lexeme(string ind\_name, lexeme& gets\_lexeme){

int hash, numb;

bool check = ind\_adrress(ind\_name, hash, numb);

if(check){

gets\_lexeme = our\_table[hash][numb];

return true;

}

return false;

}

bool variable\_table::get\_lexeme(int hash, int numb, lexeme& gets\_lexeme){

if(hash >= 0 && hash < hash\_N && numb >= 0 && numb < our\_table[hash].size()){

gets\_lexeme = our\_table[hash][numb];

return true;

}

else

return false;

}

int variable\_table::calc\_hash(string ind\_name){

if('0' <= ind\_name[0] && ind\_name[0] <= '9')

return ind\_name[0] - '0';

if('a' <= ind\_name[0] && ind\_name[0] <= 'z')

return ind\_name[0] - 'a' + 10;

if('A' <= ind\_name[0] && ind\_name[0] <= 'Z')

return ind\_name[0] - 'A' + 36;

return 0;

}

## Файл «launch.cpp»

#include "variable\_table.h"

int main(){

const\_table<string> string\_t("table2.txt");

string\_t.add\_element("void");

string\_t.add\_element("if");

bool check1 = string\_t.is\_element\_in\_table("int");

bool check2 = string\_t.is\_element\_in\_table("double");

int n0;

string\_t.get\_elemen\_number("int", n0);

string get\_s;

string\_t.get\_elemen\_value(n0, get\_s);

variable\_table vars\_t;

vars\_t.add\_ind("a");

vars\_t.add\_ind("b");

vars\_t.add\_ind("aab");

bool check3 = vars\_t.add\_ind("a");

bool check4 = vars\_t.check\_ind\_in\_table("aab");

bool check5 = vars\_t.check\_ind\_in\_table("c");

int n1, n2;

vars\_t.ind\_adrress("b",n1, n2);

vars\_t.set\_ind\_type("a", 1);

vars\_t.set\_ind\_val("aab", true);

lexeme t1, t2;

vars\_t.get\_lexeme("aab", t1);

vars\_t.get\_lexeme(n1, n2, t2);

return 0;

}