|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Лабораторная работа № 1 | | |
| по дисциплине «Языки программирования и методы трансляции» | | |
| **Проектирование и реализация таблиц, используемых в трансляторе** | | |
|  | | |
|  | Студенты | Исакин даниил |
|  | вострецова екатерина |
| Группа ПМ-13 |  |
| Бригада 10 |  |
|  |  |
|  |  |
| Преподаватель | Дворецкая Виктория константиновна |
|  |  |
| Новосибирск, 2024 | | |

1. **Цель**

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.

1. **Задание**
2. Подмножество языка С++ включает:

* данные типа **int**;
* инструкции описания переменных;
* операторы присваивания в любой последовательности;
* полный набор арифметических, логических операций и операций сравнения.

1. разработать структуру постоянных таблиц для хранения алфавита языка, зарезервированных слов, знаков операций, разделителей и пр.;

реализовать для постоянных таблиц алгоритм поиска элемента в упорядоченной таблице;

1. разработать структуру переменных таблиц с вычисляемым входом для хранения идентификаторов и констант (вид хеш-функции и метод рехеширования задает разработчик);
2. реализовать для переменных таблиц алгоритмы поиска/добавления лексемы, поиска/добавления атрибутов лексемы;
3. разработать программу для тестирования и демонстрации работы программ пп.1-3.

Исходные данные

Для постоянных таблиц источником исходных данных является текстовый файл, в котором находятся элементы соответствующего типа, каждый элемент находится на отдельной строке. Типы элементов: char (символ) и string (строка). На размер файла не накладывается никаких определенных ограничений, кроме аппаратных и программных ограничений компьютера. Также элементы могут быть добавлены из программы, без участия файла.

Для переменных таблиц добавление происходит из программы по имени идентификатора или по значению константы. На количество элементов не накладывается никаких определенных ограничений, кроме аппаратных и программных ограничений компьютера.

Структура таблиц

Класс постоянных таблиц

Название класса: table\_const.

Внутреннее представление: set (упорядоченное множество).

Содержимое контейнера: произвольный шаблон, подставляется char (символ) или string (строка).

Методы класса:

table\_const() - конструктор по умолчанию

~table\_const() - деструктор

inline void add(type elem) - добавление элемента elem в таблицу

bool read\_file(string name) - чтение таблицы из файла name

bool contains(type elem) - проверка есть ли элемент elem в таблице

bool get\_num(type elem, int &num) - поиск номера num по значению elem

bool get\_val(int num, type &elem) - поиск значения elem по номеру num

Класс переменных таблиц

Название класса: table\_var

Внутреннее представление: массив из элементов типа vector (вектор) из элементов типа lexem (лексемы).

Содержимое контейнера: элементы типа lexem

class lexeme

{

public:

// Имя идентификатора или значение константы

string name;

// Тип, 0 - не определен, 1 - int, 2 - float

int type;

// Массив флагов "илициализировано ли" размерности dimension

vector<bool> is\_init;

// Размерность массива, для переменных и констант - 1.

int dimension;

// Конструктор по умолчанию

lexeme();

// Конструктор с заданием имени идентификатора или значения константы

lexeme(string new\_name);

// Деструктор

~lexeme();

// Оператор присваивания

lexeme &operator = (const lexeme &other);

};

Хэш-функция: остаток от деления суммы элементов строкового представления имени идентификатора или значения константы на размер хэш-таблицы.

Методы класса:

int get\_hash(string name); - подсчет хэша для name

int get\_chain(string name); - подсчет номера в цепочке для name

table\_var(); - конструктор с размером таблицы по умолчанию

table\_var(int new\_hashnum); - конструктор с пользовательским размером таблицы new\_hashnum

~table\_var(); - деструктор

bool get\_location(string name, int &hash, int &chain); - определение хэша hash и номера в цепочке chain для name

inline bool contains(string name); - проверка есть ли элемент с name в таблице

bool add(string name); - добавление нового имени идентификатора или значения константы name

bool set\_type(int hash, int chain, int type); - задание типа type по хэшу hash и номеру в цепочке chain

bool set\_type(string name, int type); - задание типа type по имени идентификатора или значению константы name

bool set\_dimension(int hash, int chain, int dimension); - задание размерности dimension по хэшу hash и номеру в цепочке chain

bool set\_dimension(string name, int dimension); - задание размерности dimension по имени идентификатора или значению константы name

bool set\_is\_init(int hash, int chain, bool is\_init); - задание флага инициализации is\_init по хэшу hash и номеру в цепочке chain

bool set\_is\_init(string name, bool is\_init); - задание флага инициализации is\_init по имени идентификатора или значению константы name

bool set\_is\_init(int hash, int chain, bool is\_init, int init\_index); - задание флага инициализации is\_init[init\_index] для массивов по хэшу hash и номеру в цепочке chain

bool set\_is\_init(string name, bool is\_init, int init\_index); - задание флага инициализации is\_init[init\_index] для массивов по имени идентификатора или значению константы name

bool get\_lexeme(int hash, int chain, lexeme &lexeme); - получение структуры lexeme по хэшу hash и номеру в цепочке chain

bool get\_lexeme(string name, lexeme &lexeme); - получение структуры lexeme по имени идентификатора или значению константы name

**Текст программы**

**lexeme.cpp**

#include "lexeme.h"

// Конструктор по умолчанию

lexeme::lexeme() {}

// Конструктор с заданием имени идентификатора или значения константы

lexeme::lexeme(string new\_name)

{

name = new\_name;

type = 0;

is\_init.push\_back(false);

dimension = 1;

}

// Деструктор

lexeme::~lexeme()

{

is\_init.clear();

}

**lexeme.h**

#ifndef LEXEME\_H\_INCLUDED

#define LEXEME\_H\_INCLUDED

#include <string>

#include <vector>

using namespace std;

// Класс для хранения идентификаторов и констант

class lexeme

{

public:

// Имя идентификатора или значение константы

string name;

// Тип, 0 - не определен, 1 - int, 2 - float

int type;

// Массив флагов "илициализировано ли" размерности dimension

vector<bool> is\_init;

// Размерность массива, для переменных и констант - 1.

int dimension;

// Конструктор по умолчанию

lexeme();

// Конструктор с заданием имени идентификатора или значения константы

lexeme(string new\_name);

// Деструктор

~lexeme();

// Оператор присваивания

lexeme &operator = (const lexeme &other)

{

if(this != &other)

{

name = other.name;

type = other.type;

dimension = other.dimension;

is\_init = other.is\_init;

}

return \*this;

}

};

#endif // LEXEME\_H\_INCLUDED

**Main.cpp**

#include <iostream>

#include <stdio.h>

#include "table\_const.h"

#include "table\_var.h"

using namespace std;

int main()

{

table\_const<string> a;

a.read\_file("reserved\_words.txt");

cout << "a.contains(\"int\") = " << a.contains("int") << endl;

cout << "a.contains(\"main\") = " << a.contains("main") << endl;

int num;

a.get\_num("return", num);

cout << "a.get\_num(\"return\", num): num = " << num << endl;

string str;

a.get\_val(num, str);

cout << "a.get\_val(num, str): str = " << str << endl;

table\_var b;

b.add("avriable");

b.add("vairable");

b.add("vairalbe");

b.add("variable");

int hash, chain;

b.get\_location("variable", hash, chain);

cout << "b.get\_location(\"variable\", hash, chain): hash = " << hash << " chain = " << chain << endl;

b.set\_type("variable", 2);

b.set\_dimension("variable", 3);

b.set\_is\_init("variable", true);

b.set\_is\_init("variable", false, 1);

b.set\_is\_init("variable", true, 2);

lexeme c;

b.get\_lexeme("variable", c);

cout << "c.name = " << c.name << endl;

cout << "c.type = " << c.type << endl;

cout << "c.is\_init[0] = " << c.is\_init[0] << endl;

cout << "c.is\_init[1] = " << c.is\_init[1] << endl;

cout << "c.is\_init[2] = " << c.is\_init[2] << endl;

return 0;

}

**Table\_const.h**

#ifndef TABLE\_CONST\_H\_INCLUDED

#define TABLE\_CONST\_H\_INCLUDED

#include <fstream>

#include <string>

#include <set>

using namespace std;

// Класс постоянных таблиц

template <typename type> class table\_const

{

private:

set<type> table;

public:

// Конструктор по умолчанию

table\_const() {}

// Деструктор

~table\_const()

{

table.clear();

}

// Добавление элемента в таблицу

inline void add(type elem)

{

table.insert(elem);

}

// Чтение таблицы из файла

bool read\_file(string name)

{

ifstream fs(name.c\_str(), ios::in);

if(!fs.is\_open()) return false;

type elem;

while (!fs.eof())

{

fs >> elem;

add(elem);

}

return true;

}

// Проверка есть ли элемент в таблице

bool contains(type elem)

{

typename set<type>::iterator it = table.find(elem);

if(it == table.end()) return false;

return true;

}

// Поиск номера по значению

bool get\_num(type elem, int &num)

{

if(!contains(elem)) return false;

num = static\_cast<int>(distance(table.begin(), table.find(elem)));

return true;

}

// Поиск значения по номеру

bool get\_val(int num, type &elem)

{

if(num < 0 || num >= static\_cast<int>(table.size())) return false;

typename set<type>::iterator it = table.begin();

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

it++;

elem = \*it;

return true;

}

};

#endif // TABLE\_CONST\_H\_INCLUDED

**Table\_var.cpp**

#include "table\_var.h"

#include <cstdint>

// Размер хэш-таблицы по умолчанию

#define default\_hashnum 100

// Подсчет хэша

int table\_var::get\_hash(string name)

{

int hash = 0;

for(int i = 0; i < static\_cast<int>(name.size()); i++)

hash += name[static\_cast<uint64\_t>(i)];

return hash % hashnum;

}

// Подсчет номера в цепочке

int table\_var::get\_chain(string name)

{

for(int i = 0, h = get\_hash(name); i < static\_cast<int>(table[h].size()); i++)

if(name == table[static\_cast<uint64\_t>(h)][static\_cast<uint64\_t>(i)].name) return i;

return -1;

}

// Конструктор с размером таблицы по умолчанию

table\_var::table\_var()

{

hashnum=default\_hashnum;

table = new vector<lexeme> [hashnum];

}

// Конструктор с пользовательским размером таблицы

table\_var::table\_var(int new\_hashnum)

{

hashnum=new\_hashnum;

table = new vector<lexeme> [hashnum];

}

// Деструктор

table\_var::~table\_var()

{

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

table[i].clear();

delete [] table;

}

// Проверка есть ли элемент в таблице

inline bool table\_var::contains(string name)

{

if(get\_chain(name) != -1) return true;

return false;

}

// Добавление нового имени идентификатора или значения константы

bool table\_var::add(string name)

{

if(contains(name)) return false;

int h = get\_hash(name);

table[h].push\_back(lexeme(name));

return true;

}

// Задание типа по хэшу и номеру в цепочке

bool table\_var::set\_type(int hash, int chain, int type)

{

if(chain == -1) return false;

table[static\_cast<uint64\_t>(hash)][static\_cast<uint64\_t>(chain)].type = type;

return true;

}

// Задание типа по имени идентификатора или значению константы

bool table\_var::set\_type(string name, int type)

{

int hash = get\_hash(name), chain = get\_chain(name);

return set\_type(hash, chain, type);

}

// Задание размерности по хэшу и номеру в цепочке

bool table\_var::set\_dimension(int hash, int chain, int dimension)

{

if(chain == -1) return false;

table[static\_cast<uint64\_t>(hash)][static\_cast<uint64\_t>(chain)].dimension = dimension;

table[static\_cast<uint64\_t>(hash)][static\_cast<uint64\_t>(chain)].is\_init.resize(static\_cast<uint64\_t>(dimension));

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

table[static\_cast<uint64\_t>(hash)][static\_cast<uint64\_t>(chain)].is\_init[static\_cast<uint64\_t>(i)] = false;

return true;

}

// Задание размерности по имени идентификатора или значению константы

bool table\_var::set\_dimension(string name, int dimension)

{

int hash = get\_hash(name), chain = get\_chain(name);

return set\_dimension(hash, chain, dimension);

}

// Задание флага инициализации для массивов по хэшу и номеру в цепочке

bool table\_var::set\_is\_init(int hash, int chain, bool is\_init, int init\_index)

{

if(chain == -1) return false;

table[static\_cast<uint64\_t>(hash)][static\_cast<uint64\_t>(chain)].is\_init[static\_cast<uint64\_t>(init\_index)] = is\_init;

return true;

}

// Задание флага инициализации для массивов по имени идентификатора или значению константы

bool table\_var::set\_is\_init(string name, bool is\_init, int init\_index)

{

int hash = get\_hash(name), chain = get\_chain(name);

return set\_is\_init(hash, chain, is\_init, init\_index);

}

// Задание флага инициализации по хэшу и номеру в цепочке

bool table\_var::set\_is\_init(int hash, int chain, bool is\_init)

{

return set\_is\_init(hash, chain, is\_init, 0);

}

// Задание флага инициализации по имени идентификатора или значению константы

bool table\_var::set\_is\_init(string name, bool is\_init)

{

return set\_is\_init(name, is\_init, 0);

}

// Определение хэша и номера в цепочке

bool table\_var::get\_location(string name, int &hash, int &chain)

{

int h = get\_hash(name), c = get\_chain(name);

if(chain == -1) return false;

hash = h;

chain = c;

return true;

}

// Получение структуры lexeme по хэшу и номеру в цепочке

bool table\_var::get\_lexeme(int hash, int chain, lexeme &lexeme)

{

if(chain == -1) return false;

lexeme = table[static\_cast<uint64\_t>(hash)][static\_cast<uint64\_t>(chain)];

return true;

}

// Получение структуры lexeme по имени идентификатора или значению константы

bool table\_var::get\_lexeme(string name, lexeme &lexeme)

{

int hash = get\_hash(name), chain = get\_chain(name);

return get\_lexeme(hash, chain, lexeme);

}

#undef default\_hashnum

**Table\_var.h**

#ifndef TABLE\_VAR\_H\_INCLUDED

#define TABLE\_VAR\_H\_INCLUDED

#include <fstream>

#include <string>

#include <vector>

#include "lexeme.h"

using namespace std;

// Класс переменных таблиц

class table\_var

{

private:

// Размер таблицы

int hashnum;

// Указатель на массив цепочек

vector<lexeme> \*table;

// Подсчет хэша

int get\_hash(string name);

// Подсчет номера в цепочке

int get\_chain(string name);

public:

// Конструктор с размером таблицы по умолчанию

table\_var();

// Конструктор с пользовательским размером таблицы

table\_var(int new\_hashnum);

// Деструктор

~table\_var();

// Определение хэша и номера в цепочке

bool get\_location(string name, int &hash, int &chain);

// Проверка есть ли элемент в таблице

inline bool contains(string name);

// Добавление нового имени идентификатора или значения константы

bool add(string name);

// Задание типа по хэшу и номеру в цепочке

bool set\_type(int hash, int chain, int type);

// Задание типа по имени идентификатора или значению константы

bool set\_type(string name, int type);

// Задание размерности по хэшу и номеру в цепочке

bool set\_dimension(int hash, int chain, int dimension);

// Задание размерности по имени идентификатора или значению константы

bool set\_dimension(string name, int dimension);

// Задание флага инициализации по хэшу и номеру в цепочке

bool set\_is\_init(int hash, int chain, bool is\_init);

// Задание флага инициализации по имени идентификатора или значению константы

bool set\_is\_init(string name, bool is\_init);

// Задание флага инициализации для массивов по хэшу и номеру в цепочке

bool set\_is\_init(int hash, int chain, bool is\_init, int init\_index);

// Задание флага инициализации для массивов по имени идентификатора или значению константы

bool set\_is\_init(string name, bool is\_init, int init\_index);

// Получение структуры lexeme по хэшу и номеру в цепочке

bool get\_lexeme(int hash, int chain, lexeme &lexeme);

// Получение структуры lexeme по имени идентификатора или значению константы

bool get\_lexeme(string name, lexeme &lexeme);

};

#endif // TABLE\_VAR\_H\_INCLUDED

**Тест**

**Входныеданные**int

void

main

return

\*

+

-

/

>>

<<

&

|

**Выходные данные**

a.contains("int") = 1

a.contains("main") = 1

a.get\_num("return", num): num = 9

a.get\_val(num, str): str = return

b.get\_location("variable", hash, chain): hash = 38 chain = 3

c.name = variable

c.type = 2

c.is\_init[0] = 1

c.is\_init[1] = 0

c.is\_init[2] = 1