Министерство образования и науки Российской Федерации
Новосибирский государственный технический университет
Кафедра прикладной математики

Языки программирования и методы трансляции

Лабораторная работа №1

Факультет	прикладной математики и информатики
Группа	ПМ-01
Студенты	Александров М.Е.
	Жигалов П.С.
Преподаватели	Еланцева И.Л.
	Полетаева И.А.
Вариант	7

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.

Задание

Подмножество языка C++ включает:

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

С использованием средств объектно-ориентированного программирования

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;

2) разработать структуру переменных таблиц с вычисляемым входом для хранения идентификаторов и констант (вид хеш-функции и метод рехеширования задает разработчик);

3) реализовать для переменных таблиц алгоритмы поиска/добавления лексемы, поиска/добавления атрибутов лексемы;

4) 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

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 = distance(table.begin(), table.find(elem));
        return true;
    }
    // Поиск значения по номеру
    bool get_val(int num, type &elem)
    {
        if(num < 0 || num >= 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
```

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
```

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();
}
```

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
```

table_var.cpp

```
#include "table_var.h"
// 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 < name.size(); i++)
        hash += name[i];
    return hash % hashnum;
}

// Подсчет номера в цепочке
int table_var::get_chain(string name)
{
    for(int i = 0, h = get_hash(name); i < table[h].size(); i++)
        if(name == table[h][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[hash][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[hash][chain].dimension = dimension;
    table[hash][chain].is_init.resize(dimension);
    for(int i = 0; i < dimension; i++)
        table[hash][chain].is_init[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[hash][chain].is_init[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[hash][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
```

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(\"double\") = " << a.contains("double") << 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;
}
```

Тестовые примеры

Файл reserved_words.txt:

```
int
float
main
return
```

Вывод теста из main.cpp

```
a.contains("int") = 1
a.contains("double") = 0
a.get_num("return", num): num = 3
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
```