Министерство науки и высшего образования Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №1

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

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

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 с использованием средств объектно-ориентированного программирования:

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

Исходные данные представляют собой строчку(для статических таблиц) или одной из специальной структур(для динамических таблиц). Для создания или поиска элемента не нужно знать его представления, только для добавления/поиска аргументов. Статические и динамические функции имеют общие функции, кроме функций поиска и ввода аргументов.

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.

В данных момент источником исходных данный является вызываемые методы класса. Так же возможен ввод с файла, который осуществляется из вне класса.

Хэш-функция может задаваться, как лямда-функция, по умолчанию хэш-функцией является остаток от деления суммы элементов строки(имени идентификатора) на размер хэш-таблицы

Представления элементов Хэш-таблиц:

Элементами статических Хэш-таблиц является строка

Так же есть 3 динамических таблицы для Переменных, констант и структур

Принадлежность к ним можно определить по названию:

Переменная начинается со строчной буквы, константа с цифры или знака кавычек, а структура с заглавной буквы.

Представление в динамических таблицах:

struct Indint //Аргументы у индинтификатора

{

string name; //Имя индентификатора

union

{

float f;

int i;

} value;

int type; //0 - int, 1 - float

operator Hash::Const();

operator Hash::Struct();

operator bool();

Indint(string name); //Конструктор

};

struct Const //Аргументы у константы

{

string name;

int type;

int value;

operator Hash::Indint();

operator Hash::Struct();

operator bool();

Const(string name); //Конструктор

};

struct Struct

{

string name; //Название структуры

int nInt; //Колчиество целых чисел

int nFloat; //Количество вещественных чисел

vector<string> valueName; //Названия индинтификаторов в структуре

vector<int> valueI; //Динамический массив целых чисел(можно вектор сделать)

vector<float> valueF; //Динамический массив вещественныз чисел

operator Hash::Indint();

operator Hash::Const();

operator bool();

Struct(string name);

};

Публичные методы:

class Hash

{

public:

Hash(); //Конструктор

~Hash(); //Деструктор

struct Indint;

struct Const;

struct Struct;

struct Tocken

{

int nTable; //Номер таблицы

int nString; //Строка в хэш-таблице

int nPos; //Порядок в хэш-таблице

operator bool();

};

Tocken Find(string a); //Поиск, результатом будет значение true или false, так же можно обратиться к токену и посмотреть где нашли

bool Push(string b); //Добавление элемента в динамическую таблицу

//Добавление в статическую происходит в конструкторе приватной функцией

template<class T>

bool SetArg(Tocken tocken, T arg) //Добавдение аргументов

template<class T>

T GetArg(Tocken tocken) //Получение аргумента

void SetHeshFunc(function<int(string)> func); //Задание Хэш-функции

Приватные методы и параметры:

static const int Nkw = 10; //Количество строк в хэш-таблице ключевых слов

static const int No = 15; //Количество строк в хэш-таблице операций

static const int Nd = 10; //Количество строк в хэш-таблице разделителкй

static const int Ni = 255; //Количество строк в хэш-таблице индинтификаторов

static const int Nc = 63; //Количество строк в хэш-таблице констант

static const int Ns = 15; //Количество строк в хэш-таблице структур

static const int n = 6; //Количество хэш-таблиц

enum Table

{

Keywords = 0,

Operation = 1,

Delimiters = 2,

Indintificator = 3,

Constant = 4,

Structures = 5

};

bool SetArgIndint(Indint ind); //Добавление аргументов у индинтификаторов

bool SetArgConst(Const con); //Добавление аргументов у констант

bool SetArgStruct(Struct str); //Добавление аргументов у структур

Indint GetArgIndint(string name); //Получить аргументы у индентификатора

Const GetArgConst(string name); //Получить аргументы у константы

Struct GetArgStruct(string name); //Получить аргументы у структуры

void Add(int nTable, string name); //Добавление элемента в таблицу

Tocken Check(int nTable, string name); //Проверка наличия элемента в таблице

vector<string> keywords[Nkw]; //Ключевые слова

vector<string> operation[No]; /Знаки операции

vector<string> delimiters[Nd]; ///Разделители

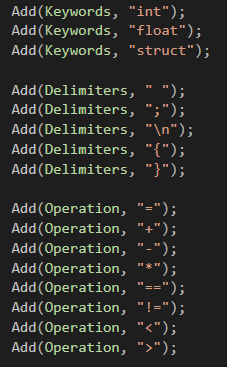
vector<Indint> indintificator[Ni]; //Идентификаторы

vector<Const> constant[Nc]; //Константы

vector<Struct> structures[Ns]; //Структуры

function<int(string)> hashFunc; //Хэш функция

1. **Тесты**

Конструктор класса  


|  |  |  |  |
| --- | --- | --- | --- |
| Входные данные | Резльтат | | Пояснение теста |
| r1 = h.Find("{");  r2 = h.Find("`"); | 1 2 3 0  0 -1 -1 -1  Пояснение результата: 1 число типа bool означает нашлось или нет, 2 – номер таблицы, 3 – номер строки, 4 – номер последовательности в строке | | Поиск существующего токина  Поиск не существующего токина |
| r3 = h.Find("f");  r4 = h.Push("`");  r5 = h.Push("r");  r6 = h.Push("A");  r7 = h.Push("\"str\"");  r8 = h.Push("r");  r9 = h.Find() | 0  0  1  1  1  0  1  Пояснение: 1 – сигнализирует об успешном выполнение, 0 – о ошибке | | Поиск не существующего  Добавление невозможного  Добавление переменной  Добавление структуры  Добавление константы  Добавление уже существующего  Поиск добавленного |
| auto r10 = h.GetArg<Hash::Indint>("r");  cout << r10 << endl;  r10.type = 1;  float f = 0.3;  r10.velue = \*((int\*)&f);  auto r11 = h.SetArg(r10);  cout << r11 << endl;  auto r12 = h.GetArg<Hash::Indint>("r");  float f2 = \*((float\*)&r12.velue);  cout << r12 << endl;  cout << r12.type << " " << f2 << endl; | | 0  1  1  2 0.3 | Взятие не заданных аргументов  Добавление аргумента  Взятие заданных аргументов  Вывод значений |

1. **Текст программы**

/\* Includes.h \*/

#include <iostream>

#include <vector>

#include <functional>

/\*Hash.h\*/

#pragma once

#include"Includes.h"

using namespace std;

class Hash

{

public:

Hash();

~Hash();

struct Indint;

struct Const;

struct Struct;

struct Tocken

{

int nTable; //Номер таблицы

int nString; //Строка в хэш-таблице

int nPos; //Порядок в хэш-таблице

operator bool();

};

struct Indint //Аргументы у индинтификатора

{

string name; //Имя индентификатора

union

{

float f;

int i;

} value;

int type; //0 - int, 1 - float

bool isInitialized;

operator Hash::Const();

operator Hash::Struct();

operator bool();

Indint(string name); //Конструктор

//По идее этого не должно еще быть,

//и они должны будут работать с токенами

//Indint operator=(Indint a);

//Indint operator+(Indint a);

//Indint operator-(Indint a);

//Indint operator\*(Indint a);

//bool operator==(Indint a);

//bool operator!=(Indint a);

//bool operator<(Indint a);

//bool operator>(Indint a);

};

struct Const //Аргументы у константы

{

string name;

int type;

int value;

operator Hash::Indint();

operator Hash::Struct();

operator bool();

Const(string name); //Конструктор

};

//Разделил чтобы можно было задавать разные индентификаторы

struct Struct

{

string name; //Название структуры

int nInt; //Колчиество целых чисел

int nFloat; //Количество вещественных чисел

vector<string> valueName; //Названия индинтификаторов в структуре

vector<int> valueI; //Динамический массив целых чисел(можно вектор сделать)

vector<float> valueF; //Динамический массив вещественныз чисел

operator Hash::Indint();

operator Hash::Const();

operator bool();

Struct(string name);

};

Tocken Find(string a); //Поиск

bool Push(string b); //Добавление элемента в динамическую таблицу

template<class T>

bool SetArg(Tocken tocken, T arg) //Добавдение аргументов

{

if (is\_same<T, Hash::Indint>::value)

return SetArgIndint(tocken, arg);

if (is\_same<T, Hash::Const>::value)

return SetArgConst(tocken, arg);

if (is\_same<T, Hash::Struct>::value)

return SetArgStruct(tocken, arg);

return false;

}

template<class T>

T GetArg(Tocken tocken) //Получение аргумента

{

if (is\_same<T, Hash::Indint>::value)

return GetArgIndint(tocken);

if (is\_same<T, Hash::Const>::value)

return GetArgConst(tocken);

if (is\_same<T, Hash::Struct>::value)

return GetArgStruct(tocken);

return T(0);

}

void SetHeshFunc(function<int(string)> func); //Задание Хэш-функции

private:

static const int Nkw = 10; //Количество строк в хэш-таблице ключевых слов

static const int No = 15; //Количество строк в хэш-таблице операций

static const int Nd = 10; //Количество строк в хэш-таблице разделителкй

static const int Ni = 255; //Количество строк в хэш-таблице индинтификаторов

static const int Nc = 63; //Количество строк в хэш-таблице констант

static const int Ns = 15; //Количество строк в хэш-таблице структур

static const int n = 6; //Количество хэш-таблиц

enum Table

{

Keywords = 0,

Operation = 1,

Delimiters = 2,

Indintificator = 3,

Constant = 4,

Structures = 5

};

bool SetArgIndint(Tocken tocken, Indint ind); //Добавление аргументов у индинтификаторов

bool SetArgConst(Tocken tocken, Const con); //Добавление аргументов у констант

bool SetArgStruct(Tocken tocken, Struct str); //Добавление аргументов у структур

Indint GetArgIndint(Tocken name); //Получить аргументы у индентификатора

Const GetArgConst(Tocken name); //Получить аргументы у константы

Struct GetArgStruct(Tocken name); //Получить аргументы у структуры

void Add(int nTable, string name); //Добавление элемента в таблицу

Tocken Check(int nTable, string name); //Проверка наличия элемента в таблице

vector<string> keywords[Nkw]; //Ключевые слова

vector<string> operation[No]; //Знаки операции

vector<string> delimiters[Nd]; //Разделители

vector<Indint> indintificator[Ni]; //Идентификаторы

vector<Const> constant[Nc]; //Константы

vector<Struct> structures[Ns]; //Структуры

function<int(string)> hashFunc; //Хэш функция

};

/\*Hash.cpp\*/

#include "Hash.h"

#include "Includes.h"

using namespace std;

Hash::Hash()

{

hashFunc = [](string str)

{

int sum = 0;

for (size\_t i = 0; i < str.size(); i++)

sum += str[i];

return sum;

};

Add(Keywords, "int");

Add(Keywords, "float");

Add(Keywords, "struct");

Add(Delimiters, " ");

Add(Delimiters, ";");

Add(Delimiters, "\n");

Add(Delimiters, "{");

Add(Delimiters, "}");

Add(Operation, "=");

Add(Operation, "+");

Add(Operation, "-");

Add(Operation, "\*");

Add(Operation, "==");

Add(Operation, "!=");

Add(Operation, "<");

Add(Operation, ">");

}

Hash::~Hash()

{

}

Hash::Tocken Hash::Find(string name)

{

Tocken b;

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

{

b = Check(i, name);

if (b)

return b;

}

return Hash::Tocken{ -1, -1, -1 };

}

//!!!!!!УСЛОВИЕ!!!!!!!

//структуры с большой буквы

//индентификаторы с маленькой

//констатнты с цифы или ковычки

bool Hash::Push(string b)

{

if (!b.size())

return 0;

int n;

if (b[0] == '\"' || b[0] == '\'' || b[0] >= 0 && b[0] <= 9)

n = Constant;

else

{

if (b[0] >= 'A' && b[0] <= 'Z')

n = Structures;

else

if (b[0] >= 'a' && b[0] <= 'z')

n = Indintificator;

else

return 0;

}

if (!Check(n, b))

{

Add(n, b);

return 1;

}

else

return 0;

}

bool Hash::SetArgIndint(Tocken tocken, Indint ind)

{

if (tocken) {

string name = indintificator[tocken.nString][tocken.nPos].name;

ind.name = name;

indintificator[tocken.nString][tocken.nPos] = ind;

return true;

}

else

return false;

}

bool Hash::SetArgConst(Tocken tocken, Const con)

{

if (tocken) {

string name = constant[tocken.nString][tocken.nPos].name;

con.name = name;

constant[tocken.nString][tocken.nPos] = con;

return true;

}

else

return false;

}

bool Hash::SetArgStruct(Tocken tocken, Struct str)

{

if (tocken) {

string name = structures[tocken.nString][tocken.nPos].name;

str.name = name;

structures[tocken.nString][tocken.nPos] = str;

return true;

}

else

return false;

}

Hash::Indint Hash::GetArgIndint(Tocken name)

{

if (name)

return indintificator[name.nString][name.nPos];

Indint a("-1");

return a;

}

Hash::Const Hash::GetArgConst(Tocken name)

{

if (name)

return constant[name.nString][name.nPos];

Indint a("-1");

return a;

}

Hash::Struct Hash::GetArgStruct(Tocken name)

{

if (name)

return structures[name.nString][name.nPos];

Indint a("-1");

return a;

}

void Hash::SetHeshFunc(function<int(string)> func)

{

hashFunc = func;

}

void Hash::Add(int nTable, string name)

{

switch (nTable)

{

case 0:

if(!Check(0, name))

keywords[hashFunc(name) % Nkw].push\_back(name);

break;

case 1:

if(!Check(1, name))

operation[hashFunc(name) % No].push\_back(name);

break;

case 2:

if(!Check(2, name))

delimiters[hashFunc(name) % Nd].push\_back(name);

break;

case 3:

if (!Check(3, name))

{

Indint a(name);

indintificator[hashFunc(name) % Ni].push\_back(a);

}

break;

case 4:

if (!Check(4, name))

{

Const a(name);

constant[hashFunc(name) % Nc].push\_back(a);

}

break;

case 5:

if (!Check(5, name))

{

Struct a(name);

structures[hashFunc(name) % Ns].push\_back(a);

}

break;

}

}

Hash::Tocken Hash::Check(int nTable, string name)

{

switch (nTable)

{

case 0:

{

int b = hashFunc(name) % Nkw;

for (int i = 0; i < keywords[b].size(); i++)

{

if (name == keywords[b][i])

return Hash::Tocken{ 0, b, i };

}

break;

}

case 1:

{

int b = hashFunc(name) % No;

for (int i = 0; i < operation[b].size(); i++)

{

if (name == operation[b][i])

return Hash::Tocken{ 1, b, i };

}

break;

}

case 2:

{

int b = hashFunc(name) % Nd;

for (int i = 0; i < delimiters[b].size(); i++)

{

if (name == delimiters[b][i])

return Hash::Tocken{ 2, b, i };

}

break;

}

case 3:

{

int b = hashFunc(name) % Ni;

for (int i = 0; i < indintificator[b].size(); i++)

{

if (name == indintificator[b][i].name)

return Hash::Tocken{ 3, b, i };

}

break;

}

case 4:

{

int b = hashFunc(name) % Nkw;

for (int i = 0; i < constant[b].size(); i++)

{

if (name == constant[b][i].name)

return Hash::Tocken{ 4, b, i };

}

break;

}

case 5:

{

int b = hashFunc(name) % Nkw;

for (int i = 0; i < structures[b].size(); i++)

{

if (name == structures[b][i].name)

return Hash::Tocken{ 5, b, i };

}

break;

}

default:

return Hash::Tocken{-1, -1, -1};

}

return Hash::Tocken{ -1, -1, -1 };

}

Hash::Const::operator Hash::Indint()

{

Indint a(this->name);

return a;

}

Hash::Const::operator Hash::Struct()

{

Struct a(this->name);

return a;

}

Hash::Const::operator bool()

{

if (type == -1)

return 0;

return 1;

}

Hash::Const::Const(string name)

{

this->name = name;

this->type = -1;

this->value = -1;

}

Hash::Indint::operator Hash::Const()

{

Const a(this->name);

return a;

}

Hash::Indint::operator Hash::Struct()

{

Struct a(this->name);

return a;

}

Hash::Indint::operator bool()

{

if (type == -1)

return 0;

return 1;

}

Hash::Indint::Indint(string name)

{

this->name = name;

this->type = -1;

this->value.i = -1;

isInitialized = false;

}

Hash::Struct::operator Hash::Indint()

{

Indint a(this->name);

return a;

}

Hash::Struct::operator Hash::Const()

{

Const a(this->name);

return a;

}

Hash::Struct::operator bool()

{

if (this->nInt && this->nFloat)

return 1;

else

return 0;

}

Hash::Struct::Struct(string name)

{

this->name = name;

this->nFloat = 0;

this->nInt = 0;

}

Hash::Tocken::operator bool()

{

if (nTable == -1 || nString == -1 || nPos == -1)

return false;

return true;

}

/\*Main.cpp\*/

#include"Includes.h"

#include"Hash.h"

int main()

{

Hash h;

Hash::Tocken r1 = h.Find("{");

Hash::Tocken r2 = h.Find("`");

cout << "find { \t" << r1 << " " << r1.nTable << " " << r1.nString << " " << r1.nPos << endl;

cout << "find ` \t" << r2 << " " << r2.nTable << " " << r2.nString << " " << r2.nPos << endl;

auto r3 = h.Find("r");

auto r4 = h.Push("`");

auto r5 = h.Push("r");

auto r6 = h.Push("A");

auto r7 = h.Push("\"str\"");

auto r8 = h.Push("r");

auto r9 = h.Find("r");

cout << r3 << endl;

cout << r4 << endl;

cout << r5 << endl;

cout << r6 << endl;

cout << r7 << endl;

cout << r8 << endl;

cout << r9 << endl;

cout << endl;

auto r10 = h.GetArg<Hash::Indint>(r9);

cout << r10 << endl;

r10.type = 1;

float f = 0.3;

r10.value.f = 0.3;

auto r11 = h.SetArg(r9, r10);

cout << r11 << endl;

auto r12 = h.GetArg<Hash::Indint>(r9);

float f2 = r12.value.f;

cout << r12 << endl;

cout << r12.type << " " << f2 << endl;

Hash::Struct st = h.GetArg<Hash::Struct>(h.Find("A"));

cout << "param A \t" << "A" << " " << st.nInt << " " << st.nFloat << endl;

st.nFloat = 1;

st.valueF.push\_back(0.3);

st.valueName.push\_back("fs");

h.SetArg(h.Find("A"), st);

cout << "SetParam A \t" << endl;

Hash::Struct s2 = h.GetArg<Hash::Struct>(h.Find("A"));

cout << "param A \t" << "A" << " " << s2.nInt << " " << s2.nFloat << " " <<s2.valueF[s2.nFloat - 1]<< endl;

int a;

cin >> a;

return 0;

}