Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторная работа №1

По дисциплине: «Языки программирования и методы трансляции»

**проектирование и реализация таблиц, используемых в трансляторе**

Факультет: ПМИ

Группа: ПМ-43

Студенты: Иост А.В.

Мошкина А.Д.

Преподаватель: Еланцева И.Л.

1. Вариант: 5

Новосибирск

2017

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;

• инструкции описания переменных;

• операторы присваивания, switch любой вложенности и в любой последовательности;

• операции +, – , \*, = =, != , < .

1. Структура таблиц

3.1 Структуры для работы с постоянными таблицами

Класс ключевых слов: class KeyWords

Класс знаков операций: class SgnOperat

Класс разделителей: class Separator

Каждый из классов содержит в себе функцию поиска заданного элемента в таблице, которая возвращает индекс элемента, если он найден, иначе -1.  
 int KeyWords::FindWord(string word)

int SgnOperat::FindSign(string sign)

int Separator::FindSymbol(char symbol)

Также каждый класс содержит в себе функцию получения некоторого элемента из таблицы по заданному индексу.

string KeyWords::GetWord(int numb)

string SgnOperat::GetSign(int numb)

char Separator::GetSymbol(int numb)

3.2 Структуры для работы с переменными таблицами

Структура константа: struct constanta

Содержит в себе строковое значение и целочисленный тип.

Класс констант: class Const

Структура идентификатор: struct Identificator

Содержит в себе строковое имя, целочисленный тип и целочисленное значение.

Класс идентификатор: class Identific

Элементами класса являются вектора соответствующих структур.

Каждый из классов содержит в себе:  
функцию поиска элемента в таблице, которая возвращает индекс элемента, если он найден, иначе -1;  
 int Const::SearchConst(constanta c)

int Identific::SearchIdent(Identificator id)

функцию добавления элемента в таблицу;

void Const::AddConst(constanta c)

**void Identific::AddIdent(Identificator id)**

функцию получения некоторого элемента из таблицы по заданному индексу;

**constanta Const::GetElem(int numb)**

**Identificator Identific::GetElem(int numb)**

функцию изменения элемента таблицы по номеру;

**void Const::ChangeConst(int numb, constanta c)**

**void Identific::ChangeIdent(int numb, Identificator id)**

1. **Исходные данные**

Исходными данными для постоянных таблиц являются вектора, в которых перечислены элементы этих таблиц.

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

1. **Тесты**

**5.1 Тестирование для постоянных таблиц**

|  |  |  |
| --- | --- | --- |
| Шаг | Описание | Результат |
| fill\_const\_table(f,&KeyWords); | Заполнение таблицы  (таблица Ключевых Слов) | [0] "main"  [1] "while"  [2] "int"  [3] "do"  [4] "void" |
| KeyWords.Add("if"); | Добавление элемента в таблицу | [0] "main"  [1] "while"  [2] "int"  [3] "do"  [4] "void"  [5] "if" |
| k = KeyWords.Search("main");  k = KeyWords.Search("double"); | Проверка наличия элементов в таблице | k=0 элемент есть в таблице  k=-1элемента нет в таблице |

**5.1 Тестирование для переменных таблиц**

|  |  |  |
| --- | --- | --- |
| Шаг | Описание | Результат |
| id = TableIdentifictors.Add("abc");  id = TableIdentificators.Add("b");  id = TableIdentificators.Add("c");  id = TableIdentificators.Add("b"); | Добавление элементов в таблицу | Id = 1  Id = 1  Id = 1  Id = 0  [0] 0 0 c  [7] 0 0 abc  [8] 0 0 b |
| id = TableIdenfitors.Search("abc",&i, &j);  id = TableIdentifcaors.Search("a",&i, &j); | Проверка наличия элементов в таблице | Id = 1  i = 7 j = 0  Id = 0  i = -1 j = -1 |
| id = TableIdentificators.Search("abc",&i, &j);  if(id == 1){  TableIdentiftors.ChangeAtributs(i,j,1,1);  } | Установка значений | [0] 0 0 c  [7] 11abc  [8] 0 0 b |

1. **Текст программы**

**/\*static\_table.h\*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#include <string>

#include <vector>

#include <array>

using namespace std;

namespace static\_table {

//ключевые слова

class KeyWords {

array <string, 9> key\_words;

public:

KeyWords();

~KeyWords();

int FindWord(string word);

string GetWord(int numb);

};

//знаки операций

class SgnOperat {

array <string, 8> sgn\_operat;

public:

SgnOperat();

~SgnOperat();

int FindSign(string sign);

string GetSign(int numb);

};

//разделители

class Separator {

array <char, 9> separator;

public:

Separator();

~Separator();

int FindSymbol(char symbol);

char GetSymbol(int numb);

};

}

**/\*static\_table.cpp\*/**

#include "static\_table.h"

namespace static\_table {

KeyWords::KeyWords() {

key\_words[0] = "int";

key\_words[1] = "main";

key\_words[2] = "void";

key\_words[3] = "switch";

key\_words[4] = "case";

key\_words[5] = "break";

key\_words[6] = "default";

key\_words[7] = "if";

key\_words[8] = "else";

}

KeyWords::~KeyWords() {}

int KeyWords::FindWord(string word) {

for (int i = 0; i < key\_words.size(); i++)

if (key\_words[i] == word) return i;

else return -1;

}

string KeyWords::GetWord(int numb) {

return key\_words[numb];

}

SgnOperat::SgnOperat() {

sgn\_operat[0] = "+";

sgn\_operat[1] = "-";

sgn\_operat[2] = "\*";

sgn\_operat[3] = "=";

sgn\_operat[4] = "==";

sgn\_operat[5] = "!=";

sgn\_operat[6] = "<";

sgn\_operat[7] = ">";

}

SgnOperat::~SgnOperat() {}

int SgnOperat::FindSign(string sign) {

for (int i = 0; i < sgn\_operat.size(); i++)

if (sgn\_operat[i] == sign) return i;

else return -1;

}

string SgnOperat::GetSign(int numb) {

return sgn\_operat[numb];

}

Separator::Separator() {

separator[0] = ' ';

separator[1] = '(';

separator[2] = ')';

separator[3] = '{';

separator[4] = '}';

separator[5] = ';';

separator[6] = ':';

separator[7] = '\t';

separator[8] = '\n';

}

Separator::~Separator() {}

int Separator::FindSymbol(char symbol) {

for (int i = 0; i < separator.size(); i++)

if (separator[i] == symbol) return i;

else return -1;

}

char Separator::GetSymbol(int numb) {

return separator[numb];

}

}

**/\*dynamic\_table.h\*/**

#pragma once

#include "static\_table.h"

namespace dynamic\_table {

//константы

struct constanta {

string value;

int type;

};

class Const {

vector <constanta> consts;

public:

Const();

~Const();

int SearchConst(constanta c);

void AddConst(constanta c);

constanta GetElem(int numb);

void ChangeConst(int numb, constanta c);

};

//идентификаторы

struct Identificator {

string name;

int type;

int value;

};

class Identific {

vector <Identificator> identif;

public:

Identific();

~Identific();

int SearchIdent(Identificator id);

void AddIdent(Identificator id);

Identificator GetElem(int numb);

void ChangeIdent(int numb, Identificator id);

};

}

**/\*dynamic\_table.cpp\*/**

#include "dynamic\_table.h"

namespace dynamic\_table {

Const::Const() {}

Const::~Const() {}

int Const::SearchConst(constanta c) {

for (int i = 0; i < consts.size(); i++)

if (consts[i].type == c.type && consts[i].value == c.value) return i;

return -1;

}

void Const::AddConst(constanta c) {

consts.push\_back(c);

}

constanta Const::GetElem(int numb) {

return consts[numb];

}

void Const::ChangeConst(int numb, constanta c) {

consts[numb] = c;

}

Identific::Identific() {}

Identific::~Identific() {}

int Identific::SearchIdent(Identificator id) {

for (int i = 0; i < identif.size(); i++)

if (identif[i].name == id.name && identif[i].type == id.type

&& identif[i].value == id.value) return i;

return -1;

}

void Identific::AddIdent(Identificator id) {

identif.push\_back(id);

}

Identificator Identific::GetElem(int numb) {

return identif[numb];

}

void Identific::ChangeIdent(int numb, Identificator id) {

identif[numb] = id;

}

}