Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ
НЭТИ

Кафедра прикладной математики

Языки программирования и методы трансляции

Лабораторная работа №1

Факультет: ПМИ
Группа: ПМ71
Студенты: Петровичев А.
Камынин А.

Преподаватель: Еланцева Е.Л.

Новосибирск
2020

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.

2. 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.

3. Структура Таблиц

1) Постоянные таблицы

Для работы с постоянными таблицами используем класс ConstTable наследуемый от SortedSet<string> из System.Collection.Generic

Методы класса:

Имя метода	Описание	Пример использования
`Bool ReadFrom(string filename)`	Генерирует таблицу из файла с именем filename	`MyTable.ReadFrom("input.txt");`
`Bool Add(string element)`	Добавляет в таблицу один элемент.	`MyTable.Add("main");`
`Contains(string element)`	Проверяет наличие элемента в таблице. Возвращает true, если элемент есть в таблице, false- иначе.	MyTable.Contains(main)
`Bool TryGetValue(string equalvalue, out string actualvalue)`	Проверяет, содержится ли equalvalue element в таблице, в случае успеха значение элемента возвращается в actualvalue. Функция возвращает значение, указывающее, найден ли элемент	`MyTable.TryGetValue("maint", out Lex);`

2) Переменные таблицы

Имя класса: VariableTable

Элементы в переменных таблицах будут представлены Класом Lexem

Свойства:

String Name – имя идентификатора, или значение константы
Enum ValueType{undef, integer, flt}
List<bool> isInit – массив, указывающий определено ли значение
Int Dimension – размерность массива
Свойства инкапсулированы и могут быть изменены динамически
Тип структуры данных: Класс использующий реализацию Dictionary<string, Lexem>
При большом кол-ве значений Dictionary реализован, как хэш тейбл, для подсчёта хэша используется стандартная определённая для типа Ключа функция (в нашем случае для string), описание в примечании.

Методы класса:

Имя метода	Описание	Пример использования
`VariableTable`	Конструктор по умолчанию, создаёт таблицу	-
Void Add(key,value)	Добавляет элемент с идентификатором ind_name в таблицу. Возвращает false если элемент уже есть в таблице, иначе – true.	MyTable.Add("main", new Lexem("main"));
`MyTable[key]`	Получение элемента по ключу для дальнейших модификаций	Lexem MyLex = MyTable[“main”];
Bool Contains(key)	Проверяет есть ли элемент в таблице	`MyTable.Contains(“main”);`
`Bool TryGetValue(key,out value)`	Присваивает значение value, если найден элемент key, возвращает информацию об успешности выполнения	`MyTable.TryGetValue(“main”,Value);`

4. Текст программ

Lexem.cs:

```
using System.Collections.Generic;

namespace ConsoleApplication1
{
    enum ValueType
    {
        undef,
        integer,
        flt
    }
    class Lexem
    {
        private string name;
        private int dimension = 0; //размерность: 1 - для переменных и констант
        private List<bool> isInit = new List<bool>(); //определено ли значение
```

```
        public Lexem(string name)
        {
            this.name = name;
        }

        public string Name
        {
            get => name;
        }

        public int Dimension
        {
            get => isInit.Count;
        }

        public List<bool> IsInit
        {
            get => isInit;
            set => isInit = value;
        }

        public ValueType Type1
        {
            get => Type;
            set => Type = value;
        }
        public ValueType Type = (ValueType)0;

        public override string ToString()
        {
            string ToReturn = $"name = {name}, type = {Type}, dimension = {Dimension}|
initialized:";
            for (int i = 0; i < isInit.Count; i++)
            {
                if (isInit[i])
                {
                    ToReturn += $" {i} ,";
                }
            }

            return ToReturn;
        }

        public override bool Equals(object obj)
        {
            Lexem p = (Lexem) obj;
            return this.name == p.name;
        }
    };
}
```

VariableTable.cs:

```
using System.Collections.Generic;
using System.ComponentModel;

namespace ConsoleApplication1
{
```

```
    class VariableTable : Dictionary<string, Lexem>
    {
    }
}
```

Main.cs

5. Тесты

1) Создание ConstTable из файла:

```
string workingDirectory = Environment.CurrentDirectory;
string projectDirectory = Directory.GetParent(workingDirectory).Parent.FullName;
var path = projectDirectory+ "\\input.txt";
cTable.ReadFrom(path);
foreach (var item in cTable)
{
    Console.WriteLine(item.ToString());

}
```

float

if

int

main

void

2) Проверка методов:

```
bool check;
check = cTable.Contains("main"); //true
check = cTable.Contains("double"); //false
string a;
check = cTable.TryGetValue("main", out a); // a = “main”
```

```
Lexem first = new Lexem("x"); //Лексема создана
VariableTable vTable = new VariableTable(); //Создана таблица
vTable.Add("x", first); //добавление лексемы
Lexem second = vTable["x"]; //получение лексемы
second.IsInit.Add(true); //изменение параметров инициализации
second.Type = ValueType.integer; //определение типа
Console.WriteLine(vTable["x"].ToString()); //вывод содержимого
```

name = x, type = integer, dimension = 1| initialized: 0 ,