|  |  |  |
| --- | --- | --- |
| 1 | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Практическое задание № 1 | | |
| по дисциплине «Языки программирования и методы трансляции» | | |
| **Проектирование и реализация таблиц, используемых в трансляторе** | | |
|  | | |
|  | Бригада 6 | Плешкова дарья |
| Группа ПМ-04 | Сальников дмитрий |
| Вариант 2 |  |
|  |  |
|  |  |
|  |  |
| Преподаватели | Еланцева ирина леонидовна |
|  |  |
| Новосибирск, 2023 | | |

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. **Описание класса ConstTable и его методов**

Объекты данного класса представляют из себя шаблоны постоянных таблиц для создания и использования таблиц ключевых слов, знаков операторов и разделителей.

Входные данные записаны в файле с расширением .txt и представляют собой список ключевых слов, знаков операторов или разделителей.

* 1. Метод CreateTable

Создает постоянную таблицу, считывая ее элементы из указанного файла.

void ConstTable::CreateTable(const string& fileName)

* 1. Метод IndexByWord

Определяет индекс элемента в таблице: если элемент в таблице найден, возвращает индекс, если не найден – возвращает значение -1.

int ConstTable::IndexByWord(const string& text)

* 1. Метод WordByIndex

Определяет элемент таблицы по его индексу: если элемент есть в таблице, то возвращает строку с элементом, если нет – возвращает пустую строку.

string ConstTable::WordByIndex(int Index)

* 1. Метод PrintTable

Выводит таблицу в формате [<индекс> <элемент>] в указанный файл.

void ConstTable::PrintTable(const string& outFileName)

**Примеры:**

|  |  |  |
| --- | --- | --- |
| Запрос | Файл «Keywords.txt» | Результат |
| cout << t.IndexByWord("while"); |  | 2 |
| cout << t.IndexByWord("do"); | -1 |
| cout << t.WordByIndex(3); | if |
| cout << t.WordByIndex(7); | /пустая строка/ |
| t.PrintTable(Keywords + OutFileFormat + Format); | Файл «KeywordsOut.txt» |

1. **Описание класса VariableTable и его методов**

Объекты данного класса представляют из себя шаблоны переменных таблиц для создания и использования таблиц идентификаторов и констант. Для таблиц данного шаблона нет входных данных, все элементы добавляются при помощи функций поиска и добавления.

* 1. Метод CreateEmptyTable

Создает пустую переменную таблицу.

void VariableTable::CreateEmptyTable()

* 1. Метод SearchByWord

Определяет индекс элемента в таблице: если элемент в таблице найден, возвращает индекс, если не найден – добавляет элемент в конец таблицы и возвращает индекс.

int VariableTable::SearchByWord(const string& NameVariable)

* 1. Метод SearchByNumber

Определяет элемент таблицы по его индексу: если элемент есть в таблице, то возвращает строку с именем элемента, если нет – возвращает пустую строку.

string VariableTable::SearchByNumber(int Index)

* 1. Метод SetAttribute

Устанавливает тип и значение указанного идентификатора.

void VariableTable::SetAttribute(const string& NameVariable, bool TypeVariable, int ValueVariable)

void SetAttribute(int Index, bool Type);

void SetAttribute(int Index, int Value);

* 1. Метод GetAttribute

Возвращает тип и значение идентификатора с указанным индексом.

pair<bool, int> VariableTable::GetAttribute(int Index)

* 1. Метод PrintTable

Выводит таблицу в формате [<индекс> <название> <тип> <значение элемента>] в указанный файл.

void VariableTable::PrintTable(const string& outFileName)

**Примеры:**

|  |  |  |
| --- | --- | --- |
| Запрос | Файл «KeywordsOut.txt» | Результат |
| int x = t2.SearchByWord("Value1");  cout << x << endl;  t2.PrintTable(Keywords + OutFileFormat + Format); |  | 0 |
| int x = t2.SearchByWord("Value1");  cout << x << endl;  t2.SetAttribute("Value1", true, 100);  t2.PrintTable(Keywords + OutFileFormat + Format); |  | 0 |
| int x = t2.SearchByWord("Value1");  t2.SetAttribute("Value1", true, 100);  pair<bool, int> p = t2.GetAttribute(x);  cout << p.first << endl << p.second;  t2.PrintTable(Keywords + OutFileFormat + Format); |  | 1  100 |

1. **Текст программы**

#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

using namespace std;

static const string Format = ".txt";

static const string Keywords = "Keywords";

static const string Separators = "Separators";

static const string OperationSigns = "OperationSigns";

static const string VariableFile = "VariableTable";

static const string OutFileFormat = "Out";

#pragma region ConstTable

class ConstTable

{

public:

ConstTable(const string& fileName)

{

CreateTable(fileName);

};

~ConstTable()

{

\_table.clear();

};

int IndexByWord(const string& text);

string WordByIndex(int Index);

void PrintTable(const string& outFileName);

private:

vector<string> \_table;

int IsContains(int Index);

void CreateTable(const string& fileName);

};

void ConstTable::CreateTable(const string& fileName)

{

ifstream fin(fileName);

if (!fin.is\_open())

cerr << "File didn't openned";

string s;

while (fin >> s)

\_table.push\_back(s);

fin.close();

}

int ConstTable::IndexByWord(const string& text)

{

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

if (\_table[i] == text)

return i;

return -1;

}

string ConstTable::WordByIndex(int Index)

{

return IsContains(Index) == -1 ? string() : \_table[Index];

}

int ConstTable::IsContains(int Index)

{

return Index >= 0 && Index < \_table.size() ? Index : -1;

}

void ConstTable::PrintTable(const string& outFileName)

{

ofstream fout(outFileName);

int i = 0;

for (auto& el : \_table)

fout << ++i << " " << el << endl;

fout.close();

}

#pragma endregion

#pragma region VariableTable

ostream& operator << (ostream& out, pair<bool, int> el)

{

return out << "{ " << el.first << ", " << el.second << " }" << endl;

}

struct Lexeme

{

string name;

bool type = false;

int value = -1;

Lexeme(const string& Name)

{

name = Name;

}

Lexeme() {}

friend bool operator == (const Lexeme& leftOperand, const Lexeme& rightOperand)

{

return

leftOperand.name == rightOperand.name &&

leftOperand.type == rightOperand.type &&

leftOperand.value == rightOperand.value;

}

friend bool operator != (const Lexeme& leftOperand, const Lexeme& rightOperand)

{

return !(leftOperand == rightOperand);

}

friend ostream& operator << (ostream& out, const Lexeme& el)

{

if (!el.name.empty())

{

return out << "{ "

<< el.name << ", "

<< (el.type ? string("int") : string("Unknow type")) << ", "

<< (el.value == -1 ? -1 : el.value) << " }" << endl;

}

return out << "The element doesn`t exist!";

}

};

class VariableTable

{

public:

VariableTable()

{

CreateEmptyTable();

};

~VariableTable()

{

\_table.clear();

};

int SearchByWord(const string& NameVariable);

Lexeme SearchByIndex(int Index);

void SetAttribute(const string& NameVariable, bool TypeVariable, int ValueVariable);

void SetAttribute(int Index, bool Type);

void SetAttribute(int Index, int Value);

pair<bool, int> GetAttribute(int Index);

string SearchByNumber(int Index);

void PrintTable(const string& outFileName);

private:

vector<Lexeme> \_table;

void CreateEmptyTable();

void AddingElement(Lexeme El);

int IsContains(const string& NameEl);

int IsContains(const int& Index);

};

void VariableTable::CreateEmptyTable()

{

\_table.resize(0);

}

void VariableTable::AddingElement(Lexeme El)

{

\_table.push\_back(El);

}

void VariableTable::SetAttribute(const string& NameVariable, bool TypeVariable, int ValueVariable)

{

auto indexElement = SearchByWord(NameVariable);

\_table[indexElement].type = TypeVariable;

\_table[indexElement].value = ValueVariable;

}

void VariableTable::SetAttribute(int Index, bool Type)

{

if (IsContains(Index) == -1)

return;

\_table[Index].type = Type;

}

void VariableTable::SetAttribute(int Index, int Value)

{

if (IsContains(Index) == -1)

return;

\_table[Index].value = Value;

}

void VariableTable::PrintTable(const string& outFileName)

{

ofstream fout(outFileName);

int i = 0;

for (auto& el : \_table)

fout << "[" << ++i << "] " << el << endl;

fout.close();

}

int VariableTable::IsContains(const string& NameEl)

{

if (\_table.size() == 0)

return -1;

auto x = find(\_table.begin(), \_table.end() - 1, NameEl);

if (\*x != \_table.back() || \_table.back().name == NameEl)

return distance(\_table.begin(), x);

return -1;

}

int VariableTable::IsContains(const int& Index)

{

return Index >= 0 && Index < \_table.size() ? Index : -1;

}

int VariableTable::SearchByWord(const string& NameVariable)

{

int indexElement = IsContains(NameVariable);

if (indexElement == -1)

{

AddingElement(Lexeme(NameVariable));

return static\_cast<unsigned long long>(indexElement) + \_table.size();

}

return indexElement;

}

Lexeme VariableTable::SearchByIndex(int Index)

{

int indexElement = IsContains(Index);

if (indexElement != -1)

{

return \_table[indexElement];

}

return Lexeme({});

}

pair<bool, int> VariableTable::GetAttribute(int Index)

{

auto Element = SearchByIndex(Index);

return make\_pair(Element.type, Element.value);

}

string VariableTable::SearchByNumber(int Index)

{

return SearchByIndex(Index).name;

}

#pragma endregion

int main()

{

// ConstTable t(Keywords + Format);

// t.PrintTable(Keywords + OutFileFormat + Format);

VariableTable t2;

int x = t2.SearchByWord("Value1");

t2.SetAttribute(x, 150);

x = t2.SearchByWord("Value2");

t2.SetAttribute(x, true);

// cout << x << endl;

t2.PrintTable(VariableFile + OutFileFormat + Format);

}