МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

ОТЧЕТ

по лабораторной работе №2

по дисциплине

Теория языков программирования и методов трансляции

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кузнецов Г.Д.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сапожников В.О.

19-В-1

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2022

**Задание**

Добавить в программу, разработанную в лабораторной работе 1, функции формирования таблицы идентификаторов и поиска в ней по методу цепочек и открытой адресации. Также сравнить эти методы.

**Теоретическая часть**

Заполнение таблицы идентификаторов происходит на разных этапах компиляции. Лексический анализатор создает записи в таблице, как только встречает имя в исходной программе, а другие атрибуты заполняются по мере обработки объявления этого имени. Но часто одно и то же имя используется в программе для обозначения разных объектов, иногда даже внутри одного декларационного блока.

Компилятор часто обращается к таблице идентификаторов, поэтому вопрос минимизации времени поиска по таблице идентификаторов является важным. Поскольку хэш-таблицы могут обеспечить константное время поиска (О(1)), они и являются наиболее распространенным способом организации таблиц идентификаторов. Для каждого имени в программе с помощью некоторой выбранной хэш-функции h рассчитывается целое число n, которое и будет индексом записи в таблице идентификаторов для хранения этого имени и всей сопутствующей ему информации. Так как время размещения элемента в таблице и время его поиска определяются только временем, затрачиваемым на вычисление хэш-функции, ее вычисление не должно занимать много времени. Кроме того, она не должна приводить к частым коллизиям, ситуациям, когда двум и более идентификаторам соответствует одно и то же значение хэш-функции.

Но когда возникают коллизии, то их нужно как-то разрешать. И вот некоторые варианты их разрешения:

1) **Метод цепочек** исходит из того, что хэш-функция h производит коллизии. При этом h разделяет все множество входных ключей на фиксированное количество множеств или карманов. Каждый карман содержит линейный список идентификаторов, для которых вычисленное значение хэш-функции совпало, т.е. произошла коллизия.

2) **Открытая адресация**, или **рехэширование**, в случае возникновения коллизии пересчитывает значение хэш-функции для идентификатора. Функция Insert(name) вычисляет h(name), если ячейка таблицы с полученным номером пуста, происходит добавление идентификатора в эту ячейку. Если ячейка уже занята, Insert вычисляет значение g(name) хэш-функции, задающей инкремент для вставки: (h(name) + g(name)) mod S, где S – размер таблицы. И так до тех пор, пока не будет найдена пустая ячейка. Если в результате рехэширования Insert вернется к значению h(name), это означает, что пустых ячеек в таблице больше нет.

Рассмотрим плюсы и минусы. С одной стороны метод «Цепочек» весьма прост в реализации, да и сами таблицы будут занимать меньше места в памяти, чем в методе рехэширования, т.к. количество пустых ячеек будет в разы меньше. А с другой стороны вынимать нужные нам идентификаторы будет существенно дольше, т.к. нам придется перебирать список, в то время как в методе рехэширования всего-то нужно вычислить значение нескольких хэш-функций. В любом случае каждый из этих методов зависит от размера таблицы, чем она больше, то тем больше она занимает места, но время поиска существенно возрастет в обоих методах. Чем меньше таблица, то тем дольше будет поиск нужного нам элемента.

**Практическая часть**

**Язык программирования: Kotlin**

Классы LexemeTable.kt и TokenType.kt не претерпели изменений с ЛР1.

**Table.kt**

Интерфейс, описывающие поведения для таблиц идентификаторов.

package Work2.Tables  
  
import Work2.Entry  
import java.io.PrintWriter  
  
*/\*\*  
 \* Общий итерфейс таблиц идентфиикаторов  
 \*/*interface Table {  
  
 */\*\*  
 \* Переход на новый лексический уровень  
 \*/* fun initializeScope()  
  
 */\*\*  
 \* Переход на предыдущий лексический уровень  
 \*/* fun finalizeScope()  
  
 */\*\*  
 \* Собственная реализация Хэш функции  
 \* @param key кллюч в хэш-таблице - идентификаторв  
 \*/* fun getHashCode(key: String): Int  
  
 */\*\*  
 \* Возвращает запись, хранящуюся в ячейке hashCode(name)  
 \* @param name идентификатор  
 \* @return найденная запись  
 \*/* fun lookUp(name: String): Entry?  
  
 */\*\*  
 \* Сохараняет идентфиикатор name в ячейке с номером h(name)  
 \* @param name идентификатор  
 \* @return true если запись ыбла сохранена  
 \*/* fun insert(name: String)  
  
 */\*\*  
 \* Метод вывода таблицы в файл  
 \* @param printWriter поток вывода в файл  
 \*/* fun write(printWriter: PrintWriter)  
}

**AddressTable.kt**

Реализация таблицы идентификаторов с методом открытой адресации

package Work2.Tables  
  
import Work2.Entry  
import org.apache.commons.lang3.StringUtils  
import java.io.PrintWriter  
  
class AddressTable: Table {  
 private val content = *arrayOfNulls*<Entry>(100)  
 private var level: String = "0"  
  
 */\*\*  
 \* Переход на новый лексический уровень  
 \*/* override fun initializeScope() {  
 level = (level.*last*() + 1).toString()  
 content.*forEach* **{** if (**it** != null) {  
 if (**it**.level.length == 1 && **it**.level == level) level += "a"  
 }  
 **}** }  
  
 */\*\*  
 \* Переход на предыдущий лексический уровень  
 \*/* override fun finalizeScope() {  
 if (level.length == 1)  
 {  
 level = (level.*last*() - 1).toString()  
 }  
 else {  
 level = level.*dropLast*(1)  
 level = (level.*last*() - 1).toString()  
 }  
 if (level.length == 1 && level.*last*().toInt() < 0) throw Exception("Scope violation: $level")  
 }  
  
 */\*\*  
 \* Собственная реализация Хэш функции  
 \* @param key кллюч в хэш-таблице - идентификаторв  
 \*/* override fun getHashCode(key: String): Int {  
 val c = 0.6180339887  
 var hash = 0  
 for (ch in key) hash = ((c \* hash + ch.*code*) % content.size).toInt()  
 return hash  
 }  
  
 private fun anotherHash(key: String, counter: Int): Int {  
 var hash = 0  
 for (ch in key) hash = ((counter \* hash + ch.*code*) % content.size)  
 return hash  
 }  
  
 */\*\*  
 \* Возвращает запись, хранящуюся в ячейке hashCode(name)  
 \* @param name идентификатор  
 \* @return найденная запись  
 \*/* override fun lookUp(name: String): Entry? {  
 val hash = getHashCode(name)  
 val newHash: Int  
 var entry: Entry? = content[hash]  
 var counter= 1  
  
 if (entry == null) {  
 throw Exception("Non-existent identifier: $name")  
 }  
  
 if (entry.identifier == name) {  
 return entry  
 }  
 else {  
 newHash = getHashCode(name) + anotherHash(name, ++counter) % content.size  
 while (newHash != hash) {  
 entry = content[newHash]  
  
 if (entry == null) {  
 throw Exception("Non-existent identifier: $name")  
 }  
 if (entry.identifier == name) break  
 }  
 }  
  
 return entry  
 }  
  
 */\*\*  
 \* Сохраняет идентификатор name в ячейке с номером h(name)  
 \* @param name идентификатор  
 \* @return true если запись была сохранена  
 \*/* override fun insert(name: String) {  
 val hash = getHashCode(name)  
 var newHash: Int  
 var counter = 1  
  
 if (content[hash] != null &&  
 content[hash]!!.identifier == name &&  
 content[hash]!!.level == level) {  
 content[hash] = Entry(name, level)  
 return  
 }  
  
 if (content[hash] == null) content[hash] = Entry(name, level)  
 else {  
 newHash = (getHashCode(name) + anotherHash(name, ++counter)) %

content.size  
 while (newHash != hash) {  
 if (content[newHash] == null) {  
 content[newHash] = Entry(name, level)  
 return  
 }  
 if (content[newHash] != null &&  
 content[newHash]!!.identifier == name &&  
 content[newHash]!!.level == level) {  
 content[newHash] = Entry(name, level)  
 return  
 }  
 newHash = (getHashCode(name) + anotherHash(name, ++counter)) % content.size  
 }  
 throw Exception("Identifier table is full")  
 }  
 }  
  
 */\*\*  
 \* Метод вывода таблицы в файл  
 \* @param printWriter поток вывода в файл  
 \*/* override fun write(printWriter: PrintWriter) {  
 val width = 25  
  
 printWriter.println(  
 String.*format*(  
 "%s\n%s",  
 StringUtils.center("Таблица идентификаторов", width),  
 StringUtils.center("Метод прямой адересации", width)  
 )  
 )  
 printWriter.println("-".*repeat*(width))  
 printWriter.println(  
 String.*format*(  
 "|%s|%s|%s|",  
 StringUtils.center("key", 3),  
 StringUtils.center("идентификатор", 13),  
 StringUtils.center("ур.", 5),  
 )  
 )  
 printWriter.println("-".*repeat*(width))  
 for (i in content.*indices*) {  
 if (content[i] == null) continue  
 printWriter.println(  
 String.*format*(  
 "|%s|%s|%s|",  
 StringUtils.center(i.toString(), 3),  
 StringUtils.center(content[i]!!.identifier, 13),  
 StringUtils.center(content[i]!!.level, 5)  
 )  
 )  
 }  
 printWriter.println("-".*repeat*(width))  
 printWriter.close()  
 }  
}

**ChainTable.kt**

Реализация таблицы идентификаторов с методом цепочек

package Work2.Tables  
  
import Work2.Entry  
import org.apache.commons.lang3.StringUtils  
import java.io.PrintWriter  
  
class ChainTable: Table {  
 private val content = *arrayOfNulls*<MutableList<Entry>>(100)  
 private var level: String = "0"  
  
 */\*\*  
 \* Переход на новый лексический уровень  
 \*/* override fun initializeScope() {  
 level = (level.*last*() + 1).toString()  
 content.*forEach* **{** list **->** list?.*forEach***{** if (**it**.level.length == 1 && **it**.level == level) level += "a"  
 **}  
 }** }  
  
 */\*\*  
 \* Переход на предыдущий лексический уровень  
 \*/* override fun finalizeScope() {  
 if (level.length == 1)  
 {  
 level = (level.*last*() - 1).toString()  
 }  
 else {  
 level = level.*dropLast*(1)  
 level = (level.*last*() - 1).toString()  
 }  
 if (level.length == 1 && level.*last*().toInt() < 0) throw Exception("Scope violation: $level")  
 }  
  
 */\*\*  
 \* Собственная реализация Хэш функции  
 \* @param key кллюч в хэш-таблице - идентификаторв  
 \*/* override fun getHashCode(key: String): Int {  
 val c = 0.6180339887  
 var hash = 0  
 for (ch in key) hash = ((c \* hash + ch.*code*) % content.size).toInt()  
 return hash  
 }  
  
 */\*\*  
 \* Возвращает запись, хранящуюся в ячейке hashCode(name)  
 \* @param name идентификатор  
 \* @return найденная запись  
 \*/* override fun lookUp(name: String): Entry? {  
 val hash = getHashCode(name)  
 return content[hash]?.*find* **{ it**.identifier == name **}** }  
  
 */\*\*  
 \* Сохраняет идентификатор name в ячейке с номером h(name)  
 \* @param name идентификатор  
 \* @return true если запись была сохранена  
 \*/* override fun insert(name: String) {  
 val hash = getHashCode(name)  
  
 if (content[hash] != null) {  
 for (i in content[hash]!!.*indices*) {  
 if (content[hash]!![i].identifier == name) {  
 if (content[hash]!![i].level == level) {  
 content[hash]!![i] = Entry(name, level)  
 }  
 else {  
 content[hash]!!.add(Entry(name, level))  
 }  
 }  
 else content[hash]!!.add(Entry(name, level))  
 }  
 }  
  
 if (content[hash] == null) content[hash] = *mutableListOf*(Entry(name, level))  
 }  
  
 */\*\*  
 \* Метод вывода таблицы в файл  
 \* @param printWriter поток вывода в файл  
 \*/* override fun write(printWriter: PrintWriter) {  
 val tableToPrint = content.*filterNotNull*().*map* **{** list **->** list.*distinctBy* **{ it**.level **} }** val width = tableToPrint.*maxByOrNull* **{ it**.size **}**.*toString*().length + 6  
  
 printWriter.println(  
 String.*format*(  
 "%s\n%s",  
 StringUtils.center("Таблица идентификаторов", width),  
 StringUtils.center("Метод Цепочек", width)  
 )  
 )  
 printWriter.println("-".*repeat*(width))  
 printWriter.println(  
 String.*format*(  
 "|%s|%s|",  
 StringUtils.center("key", 3),  
 StringUtils.center("идентификаторы", width-6)  
 )  
 )  
 printWriter.println("-".*repeat*(width))  
  
 for (i in tableToPrint.*indices*) {  
 printWriter.print(  
 String.*format*(  
 "|%s|%s%s",  
 StringUtils.center(i.toString(), 3),  
 StringUtils.left(tableToPrint[i].toString(), width-5),  
 " ".*repeat*(width-6-tableToPrint[i].toString().length)  
 )  
 )  
 printWriter.println("|")  
 printWriter.println("-".*repeat*(width))  
 }  
 printWriter.close()  
 }  
}

**TokenType.kt**

package Work1  
  
enum class TokenType {  
 *Identifier*,  
 *HexNumber*,  
 *Operations*,  
 *Assignment*,  
 *Separator*,  
 *Terminator*,  
}

**Entry.kt**

Класс – запись в таблице

package Work2  
  
*/\*\*  
 \* Запись в таблице идентификаторов  
 \* @param identifier сам идентификатор  
 \* @param level область видимости  
 \*/*data class Entry(  
 val identifier: String,  
 val level: String  
)

**LexicalAnalyzer.kt**

package Work2  
  
import Work2.Tables.Table  
  
*/\*\*  
 \* Лексический анализатор  
 \*/*data class LexicalAnalyzer(  
 val elements: List<Char>,  
 val operators: List<Char>,  
 val separators: List<Char>,  
 val table: Table  
) {  
 */\*\*  
 \* Удаление многострочных комментариев, т.к. они игнорируются  
 \* @param text текст для анализа  
 \* @return текст, в котором удалены многострочные комментарии  
 \*/* private fun removeMultiLineComment(text: String): String {  
 var result = text  
  
 while ("/\*" in result && "\*/" in result) {  
 result = result.*removeRange*(  
 result.*indexOf*("/\*"),  
 result.*indexOf*("\*/") + 2  
 )  
 }  
  
 if ("\*/" in result && "/\*" !in result) throw Exception("Expecting an element: " + text.*indexOf*("\*/"))  
  
 return result  
 }  
  
  
 */\*\*  
 \* Удаление однострочного комментария  
 \* @param text текст для анализа  
 \* @return текст, в котором удалены однострочные комментарии  
 \*/* private fun removeSingleLineComment(text: String): String {  
 var result = text  
  
 while ("//" in result) {  
 val start = result.*indexOf*("//")  
 var end = 0  
 for (i in start..text.length) {  
 if (text[i] == '\n') {  
 end = i  
 break  
 }  
 }  
  
 result = result.*removeRange*(start, end)  
 }  
  
 return result  
 }  
  
 */\*\*  
 \* Выделение шестнадцатиричного числа  
 \* @param str строка, из которой выделяется число  
 \* @return 16-тиричное число  
 \*/* private fun selectHexNumber(str: String): String {  
 var temp = ""  
  
 for (ch in str) {  
 if (ch in separators || ch == ';') break  
 else if (ch in elements) temp += ch  
 else {  
 throw Exception("Unresolved reference: $str")  
 }  
 }  
  
 return temp  
 }  
  
 */\*\*  
 \* Выделение идентификаторов  
 \* @param str строка, из которой выделяется идентификатор  
 \* @return идентификатор  
 \*/* private fun selectIdentifier(str: String): String {  
 var temp = ""  
  
 for (ch in str) {  
 if (ch in separators || ch == ';') break  
 if (ch.toString().*matches*("[A-z0-9\_]".*toRegex*())) temp += ch  
 else {  
 throw Exception("invalid character: ${str.*indexOf*(ch)}")  
 }  
 }  
  
 return temp  
 }  
  
 */\*\*  
 \* Выделение оператора  
 \* @param str строка, из которой выделяется оператор  
 \* @return оператор  
 \*/* private fun selectOperator(str: String): String {  
 var temp = ""  
  
 if (str[0] in operators) temp += str[0]  
 else {  
 throw Exception("invalid character: ${str[0]}")  
 }  
  
 return temp  
 }  
  
 */\*\*  
 \* Выделение присваивания  
 \* @param str строка, из которой выделяется присваивание  
 \* @return знак присваивания  
 \*/* private fun selectAssigment(str: String): String {  
 var temp = ""  
  
 if (str[0] == '=') temp += str[0]  
 else {  
 throw Exception("invalid character: ${str[0]}")  
 }  
  
 return temp  
 }  
  
 */\*\*  
 \* Ананлиз текста, построение таблицы лексем  
 \* @param text текст для анализа  
 \* @return таблицу лексем  
 \*/* fun analyze(text: String): LexemeTable {  
 val result = LexemeTable()  
 var typeOfPreviousElem = TokenType.*Assignment* var lexeme: String  
  
 *//Удаление одно- и многострочных коментариев  
 //И делим текст по пробелам* var strings = removeSingleLineComment(  
 removeMultiLineComment(text)  
 ).*split*("\\s".*toRegex*())  
  
 *//Заменяем табуляцию одинарным пробелом* strings.*forEach* **{ it**.*replace*("\t", " ") **}** strings = strings.*filter* **{ it**.*isNotEmpty*() **}** *//Анализ каждого выделенного элемента* strings.*forEach* **{** var temp = **it** *//пока выделенный элемент не пуст* while (temp.*isNotEmpty*()) {  
  
 *//удаляем пробел в начале элемента* temp = temp.*trimStart*(' ')  
  
 *//Выделение разделителей* if (temp[0] in separators)  
 {  
 lexeme = temp[0].toString()  
  
 *//Разделители ( )* if (lexeme == "(") {  
 if (")" !in text.*substring*(text.*indexOf*(lexeme), text.length)) {  
 throw Exception("Expecting an element: " + text.*indexOf*(temp))  
 }  
 }  
  
 *//Разделители { }* if (lexeme == "{") {  
 if ("}" !in text.*substring*(text.*indexOf*(lexeme), text.length)) {  
 throw Exception("Expecting an element: " + text.*indexOf*(temp))  
 }  
 table.initializeScope()  
 }  
 if (lexeme == "}") table.finalizeScope()  
  
 result.add(lexeme, TokenType.*Separator*)  
 temp = temp.*removePrefix*(lexeme)  
 typeOfPreviousElem = TokenType.*Separator* }  
  
 *//Выделение 16-тиричного числа* if (temp.*startsWith*("0x")) {  
 if (typeOfPreviousElem == TokenType.*HexNumber*) {  
 throw Exception("Expecting an element: " + text.*indexOf*(temp))  
 }  
 lexeme = selectHexNumber(temp)  
 result.add(lexeme, TokenType.*HexNumber*)  
 temp = temp.*removePrefix*(lexeme)  
 typeOfPreviousElem = TokenType.*HexNumber* }  
  
 *//Выделение идентификатора* if (Regex("""^[A-z\_]""").containsMatchIn(temp)) {  
 if (typeOfPreviousElem == TokenType.*Identifier* ||  
 typeOfPreviousElem == TokenType.*HexNumber* ) {  
 throw Exception("Expecting an element: " + text.*indexOf*(temp))  
 }  
 lexeme = selectIdentifier(temp)  
 result.add(lexeme, TokenType.*Identifier*)  
  
 table.insert(lexeme)  
  
 temp = temp.*removePrefix*(lexeme)  
 typeOfPreviousElem = TokenType.*Identifier* }  
  
 *//Выделение оператора* if (Regex("""^[+\-\*/]""").containsMatchIn(temp)) {  
 if (typeOfPreviousElem == TokenType.*Operations* ||  
 typeOfPreviousElem == TokenType.*Assignment* ) {  
 throw Exception("Expecting an element: " + text.*indexOf*(temp))  
 }  
 lexeme = selectOperator(temp)  
 result.add(lexeme, TokenType.*Operations*)  
 temp = temp.*removePrefix*(lexeme)  
 typeOfPreviousElem = TokenType.*Operations* }  
  
 *//Выделение знака присвоения* if (temp.*startsWith*("=")) {  
 if (typeOfPreviousElem == TokenType.*Assignment* ||  
 typeOfPreviousElem == TokenType.*HexNumber* ) {  
 throw Exception("Expecting an element: " + text.*indexOf*(temp))  
 }  
 lexeme = selectAssigment(temp)  
 result.add(lexeme, TokenType.*Assignment*)  
 temp = temp.*removePrefix*(lexeme)  
 typeOfPreviousElem = TokenType.*Assignment* }  
  
 *//Выделение терминального знака* if (temp.*startsWith*(";")) {  
 if (typeOfPreviousElem == TokenType.*Terminator*) {  
 throw Exception("Expecting an element: " + text.*indexOf*(temp))  
 }  
 lexeme = temp[0].toString()  
 temp = temp.*drop*(1)  
 result.add(lexeme, TokenType.*Terminator*)  
 typeOfPreviousElem = TokenType.*Terminator* }  
 }  
 **}** return result  
 }  
}

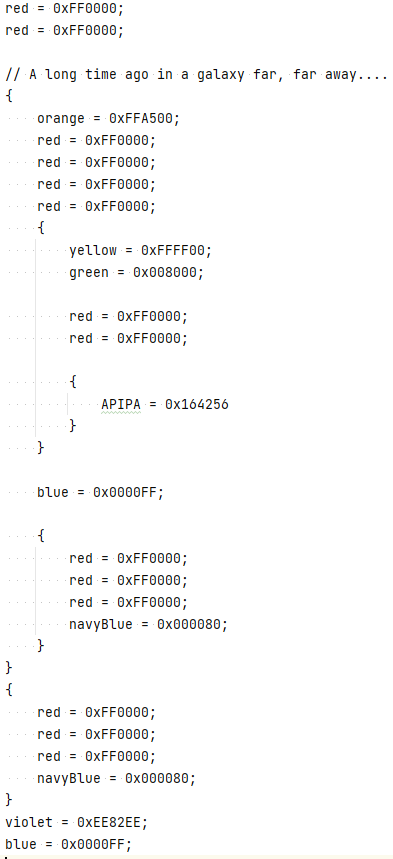
**Main.kt**

package Work2  
  
import Work2.Tables.AddressTable  
import Work2.Tables.ChainTable  
import java.io.File  
import java.io.PrintWriter  
  
*/\*\*  
 \* Теория языков и методов трансляции  
 \* 19-В-1 ИРИТ ВСТ  
 \*  
 \* Вариант 13:  
 \* Входной язык содержит арифметические выражения, разделённые символом «;».  
 \* Арифметические выражения состоят из идентификаторов, шестнадцатеричных чисел  
 \* (последовательность цифр и символов a, b, c, d, e, f, которые начинаются с 0x.  
 \* Например 0x00, 0xF0, 0xFF и т.д.), знака присваивания «=», знаков операций «+», «-»,  
 \* «\*», «/» и круглых скобок  
 \*  
 \* @author Vladislav Sapozhnikov  
 \*/*fun main(args: Array<String>) {  
 *//Входные данны читаются из файла  
 //имя файла передается аругментом командной строки* val inputText = *readFileDirectlyAsText*(args.*first*())  
  
  
 *//'Символы' доступные в 16-тиричной СС* val digits = *listOf*(  
 '0', 'x', '1', '2', '3', '4', '5', '6', '7', '8', '9',  
 'A', 'B', 'C', 'D', 'E', 'F'  
 )  
  
 *//операторы, оепредленные во входном языке* val operators = *listOf*(  
 '+', '-', '\*', '/'  
 )  
  
 val separators = *listOf*(  
 '{', '}', '(', ')'  
 )  
  
 *//Инцииализация лексического анализатора* val lexicalAnalyzerAddressTable = LexicalAnalyzer(  
 elements = digits,  
 operators = operators,  
 separators = separators,  
 table = AddressTable()  
 )  
  
 val lexicalAnalyzerWithChain = LexicalAnalyzer(  
 elements = digits,  
 operators = operators,  
 separators = separators,  
 table = ChainTable()  
 )  
  
 val addressFilePrinter = PrintWriter("OpenAddressing.txt")  
 val chainFilePrinter = PrintWriter("Chains.txt")  
  
 lexicalAnalyzerAddressTable.analyze(inputText).write(addressFilePrinter)  
 addressFilePrinter.print("\n\n\n")  
 lexicalAnalyzerAddressTable.table.write(addressFilePrinter)  
  
 lexicalAnalyzerWithChain.analyze(inputText).write(chainFilePrinter)  
 chainFilePrinter.print("\n\n\n")  
 lexicalAnalyzerWithChain.table.write(chainFilePrinter)  
  
 addressFilePrinter.close()  
 chainFilePrinter.close()  
}  
  
*/\*\*  
 \* Функция чтения файла  
 \* @param fileName имя файла, который необходимо прочитать  
 \* @return весь текст файла в виде одной строки  
 \*/*fun readFileDirectlyAsText(fileName: String): String  
 = File(fileName).*readText*(Charsets.UTF\_8)

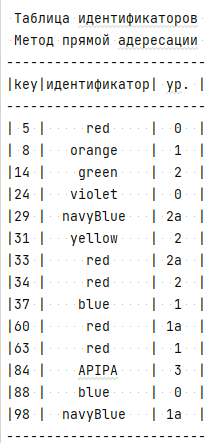
)  
  
 lexicalAnalyzer.analyze(inputText).print()  
}  
  
*/\*\*  
 \* Функция чтения файла  
 \* @param fileName имя файла, который необходимо прочитать  
 \* @return весь текст файла в виде одной строки  
 \*/*fun readFileDirectlyAsText(fileName: String): String  
 = File(fileName).*readText*(Charsets.UTF\_8)

*Результат работы программы*

Входной файл:



Результат, полученный методом открытой адресации:



Результат, полученный методом цепочек:

