|  |  |
| --- | --- |
|  | ы |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ПРАКТИЧЕСКАЯ РАБОТА № 2

по дисциплине «Теория автоматов и формальных языков»

«РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ»

Выполнил студент группы ИКБО-02-18 Благирев М.М.

Преподаватель: Зорина Н.В.

Практическая работа выполнена «19» октября 2020г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2020г.

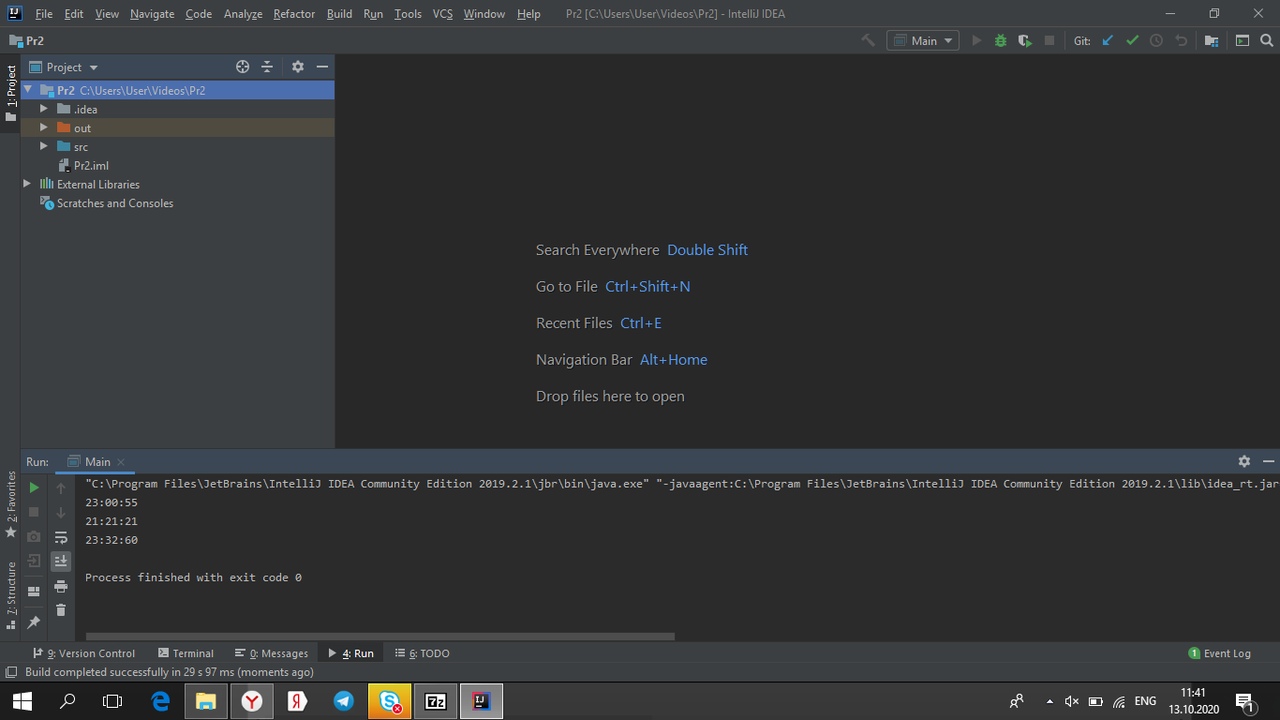
Москва 2020

Цель работы

Целью данной практической работы является ознакомления с конечными автоматами, и построение на их базе алгоритмов работы с регулярными выражениями

**Выполнение работы**

**Задание. Поиск времени вида “HH:MM:SS”**

****

**Рисунок 1 - пример работы программы**

*Листинг А — StateMachine*

|  |
| --- |
| package com.company;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  import java.util.Set;  /\*\*  \* Конечный автомат, описывающий регулярное выражение \*/  public class StateMachine {  /\*\*  \* Начальное состояние  \*/  private final int startState;  /\*\*  \* Множество конечных состояний  \*/  private final Set<Integer> endStates;  /\*\*  \* Хранение таблицы переходов для конечного автомата:  \* исходное состояние -> (входной символ -> состояние, в которое КА переходит)  \*/  private final Map<Integer, Map<Character, Integer>> transitions = new HashMap<>();  /\*\*  \* @param startState начальное состояние  \* @param endStates множество успешных конечных состояний  \*/  public StateMachine(int startState, Set<Integer> endStates) {  this.startState = startState;  this.endStates = endStates;  }  /\*\*  \* Добавление возможного перехода в таблицу переходов. \*  \*  \* @param from исходное состояние  \* @param ch входной символ  \* @param to состояние, в которое КА переходит  \*/  public void add(Integer from, Character ch, Integer to) {  Map<Character, Integer> fromMap = transitions.computeIfAbsent(from, k -> new HashMap<>());  fromMap.put(ch, to);  }  /\*\*  \* Добавление нескольких возможных переходов в таблицу переходов. \*  \*  \* @param from исходное состояние  \* @param chars входные символы  \* @param to состояние, в которое КА переходит  \*/  public void add(Integer from, List<Character> chars, Integer to) {  for (Character ch : chars) {  add(from, ch.charValue(), to);  }  }  /\*\*  \* Получение следующего состояния по текущему состоянию и входному  \* символу. \*  \*  \* @param state текущее состояние  \* @param ch входной символ  \* @return новое состояние (null, если в таблице нет переходов, т.е.  \* сопоставление с регулярным выражением  \* не может быть осуществлено для такого входного символа)  \*/  public Integer getNext(Integer state, Character ch) {  Map<Character, Integer> stateTransitions = transitions.get(state);  return stateTransitions.get(ch);  }  /\*\*  \* Сопоставление подстроки с регулярным выражением.  \*  \* @param str строка  \* @param from индекс в строке, с которого начинается сопоставление  \* @return -1 если подстрока начиная с позиции from не сопоставлена с  \* регулярным выражением; неотрицательное  \* значение, если подстрока сопоставлена с регулярным выражением - индекс в  \* строке символа, следующего за  \* сопоставленной подстрокой  \*/  public int match(String str, int from) {  int state = startState;  int i = from;  while (i < str.length()) {  char ch = str.charAt(i);  Integer nextState = getNext(state, ch);  if (nextState == null)  break;  state = nextState.intValue();  i++;  }  if (endStates.contains(state)) {  return i;  } else {  return -1;  }  }  /\*\*  \* Поиск и печать всех сопоставлений в строке.  \*/  public void findAll(String str) {  int i = 0;  while (i < str.length()) {  int match = match(str, i);  if (match < 0) {  i++;  } else {  System.out.println(str.substring(i, match));  i = match;  }  }  }  } |

*Листинг B — Main*

|  |
| --- |
| package com.company;  import java.util.Arrays;  import java.util.Collections;  import java.util.List;  import java.util.Set;  public class Main {  public static void main(String[] args) {  final int START = 0;  final int UPPER = 1;  String str = "QWERTYUIOPASDFGHJKLZXCVBNM";  List<Character> chars = new java.util.ArrayList<>(Collections.emptyList());  for (int i = 0; i < str.length(); i++)  chars.add(str.charAt(i));  Set<Integer> endStates = Collections.singleton(UPPER);  StateMachine sm = new StateMachine(START, endStates);  sm.add(START, chars, UPPER);  sm.add(UPPER, chars, UPPER);  // Поиск всех вхождений в строке:  String inputString = "\"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents/Home/bin/java\" \"-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea\_rt.jar=63457:/Applications/IntelliJ IDEA CE.app/";  sm.findAll(inputString);  }  } |

*Листинг С — MainVariant2*

|  |
| --- |
| package com.company;  import java.util.\*;  public class MainVariant2 {  public static void main(String[] args) {  final int START = 0;  final int H\_FIRST\_LIMITED = 1;  final int H\_FIRST = 2;  final int H\_SECOND = 3;  final int H\_DOT = 4;  final int M\_FIRST\_LIMITED = 5;  final int M\_FIRST = 6;  final int M\_SECOND = 7;  final int M\_DOT = 8;  final int S\_FIRST\_LIMITED = 9;  final int S\_FIRST = 10;  final int S\_SECOND = 11;  List<Character> lessThanTwo = Arrays.asList('0', '1');  List<Character> fourAndLess = Arrays.asList('0', '1', '2', '3', '4');  List<Character> lessThanSix = Arrays.asList('0', '1', '2', '3', '4', '5');  List<Character> allNumbers = Arrays.asList('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');  List<Character> two = Collections.singletonList('2');  List<Character> six = Collections.singletonList('6');  List<Character> zero = Collections.singletonList('0');  List<Character> dots = Collections.singletonList(':');  Set<Integer> endStates = Collections.singleton(S\_SECOND);  StateMachine sm = new StateMachine(START, endStates);  //часы  sm.add(START, two, H\_FIRST\_LIMITED);  sm.add(START, lessThanTwo, H\_FIRST);  sm.add(H\_FIRST\_LIMITED, fourAndLess, H\_SECOND);  sm.add(H\_FIRST, allNumbers, H\_SECOND);  sm.add(H\_SECOND, dots, H\_DOT);  //минуты  sm.add(H\_DOT, six, M\_FIRST\_LIMITED);  sm.add(H\_DOT, lessThanSix, M\_FIRST);  sm.add(M\_FIRST\_LIMITED, zero, M\_SECOND);  sm.add(M\_FIRST, allNumbers, M\_SECOND);  sm.add(M\_SECOND, dots, M\_DOT);  //секунды  sm.add(M\_DOT, six, S\_FIRST\_LIMITED);  sm.add(M\_DOT, lessThanSix, S\_FIRST);  sm.add(S\_FIRST\_LIMITED, zero, S\_SECOND);  sm.add(S\_FIRST, allNumbers, S\_SECOND);  //Костыль. Иначе StateMachine из методички будет выдавать ошибку раньше, чем распознает, что последний символ найден.  sm.add(S\_SECOND, ';', S\_SECOND);  // Поиск всех вхождений в строке:  String inputString = "\"27:09:65 23:00:55 22:65:05 21:21:21 23:32:77 23:32:60";  sm.findAll(inputString);  }  } |

**Вывод**

В результате выполнения практической работы мы ознакомились с конечными автоматами, научились строить на их базе алгоритмы для работы с регулярными выражениями

**Список литературных источников**

1. Зорина Н.В, Соболев О.В. Теория автоматов и формальных языков: Методические указания по выполнению практических работ для студентов, М. 2017, - 108 с.
2. Молчанов А.Ю. Системное программное обеспечение: Учебник для вузов. 3-е изд. – СПб, 2010. – 400 с.