

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 6

з дисципліни “Основи програмування”

# тема “Скінченні автомати”

|  |  |  |
| --- | --- | --- |
| Виконав  студент I курсу  групи КП-02  Жученко Андрій Сергійович  (*прізвище, ім’я, по батькові*)  варіант № 5 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

Київ 2020

**Мета роботи**

Застосувати скінченний автомат для реалізації алгоритмів обробки тексту.

**Постановка завдання**

Реалізувати консольну програму для розбору тексту і пошуку заданих сутностей.

Сутності за варіантом: **рядкові літерали без екранування, що починаються на символ @** - починаються на послідовність символів @" та закінчуються на символ ", між цими двома обов'язковими символами можуть бути будь-які символи окрім ". (Зауваження: @ та " не вважати символом пунктуації при розділенні двох літералів)..

#### Частина 1. Інтерактивний режим

Запитувати в циклі у користувача ввід рядка символів.

Ввід пустого рядка обриває цикл і зупиняє програму.

Розібрати введений рядок і одразу вивести 3 результати:

1. Чи вхідний рядок повністю складається з однієї сутності за варіантом без зайвих символів.
2. Кількість сутностей у вхідному рядку (див. правило розділення сутностей далі).
3. Масив всіх сутностей у вхідному рядку (див. правило розділення сутностей далі).

**Правило розділення сутностей**: для розділення двох шуканих значень обов'язково використовуються пусті символи або символи пунктуації (хоча б один такий символ). Якщо після знайденої сутності розташований будь-який інший символ - попередню сутність не враховувати при обробці.

#### Частина 2. Аргументи командного рядка

Додати обробку аргументів командного рядка для підтримки таких опцій роботи програми:

1. inputFileName.txt - перший вільний аргумент командного рядка, задає назву вхідного файлу для *стандартного режиму*.
2. -o outputFileName.txt - опція із аргументом, що задає назву вихідного файлу для *стандартного режиму*. Якщо не задано - виводити у консоль.
3. -i - (маленька літера I) булева опція без аргумента, її наявність включає *інтерактивний режим* і інші опції ігноруються.
4. Наявність будь-яких інших опцій є помилкою, що завершує роботу програми з попереднім виводом першої з цих опцій у консоль (наприклад, error: unrecognized command line option ‘-K’).
5. Якщо не задано ні назви вхідного файлу ні опції інтерактивного режиму - вивести про це помилку.

*Зауваження*:

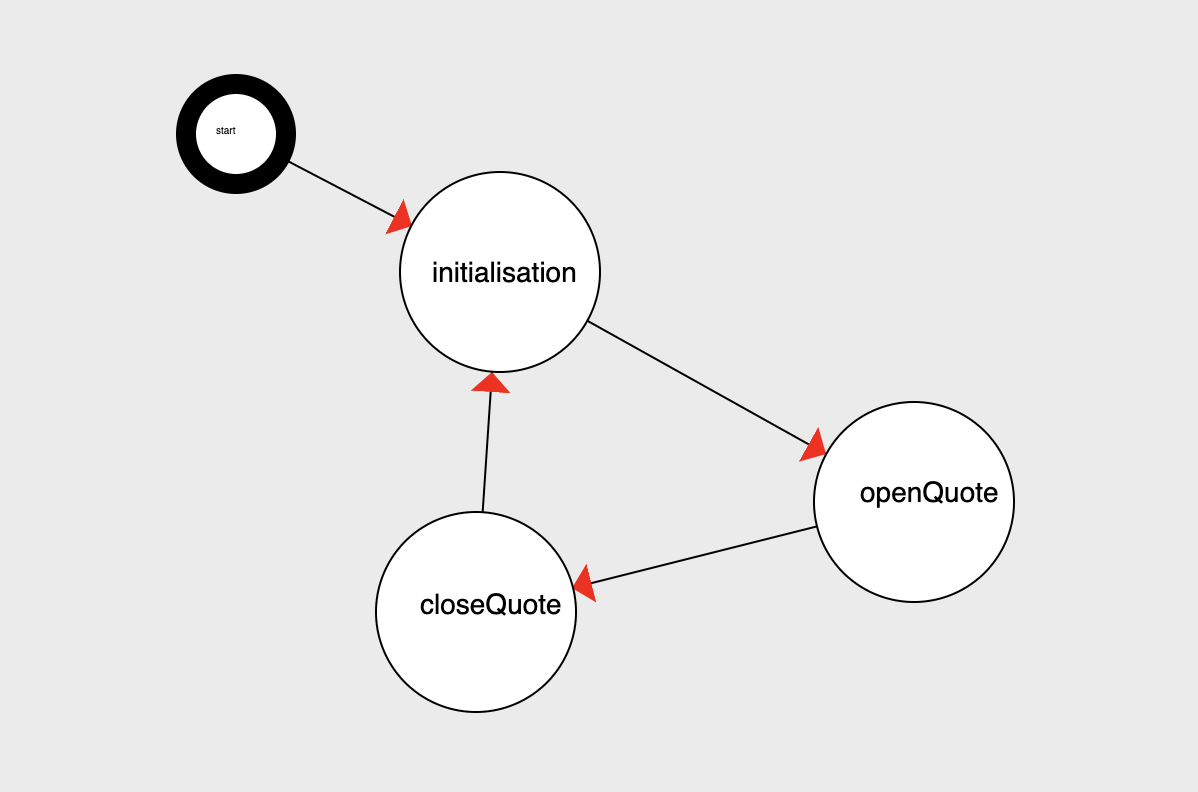
1. Якщо булева опція задана декілька раз - це нормально і вона вважається включеною
2. Якщо опція зі значенням задана декілька раз - це нормально і враховувати останнє таке значення.
3. Якщо задано декілька вільних аргументів командного рядка - поведінка програми на ваш розсуд: помилка, обробляти перший, останній або всі значення (наприклад, зчитати текст зі всіх заданих файлів і об'єднати його для обробки).

Режими роботи:

1. **Стандартний режим** - зчитати весь текст із вхідного файлу, виконати його обробку відповідно до третьої функції першої частини завдання та отримати масив шуканих сутностей. Вивести масив у файловий потік виводу (консоль або файл), кожен елемент з нового рядка тексту.
2. **Інтерактивний режим** - програма працює як у першій частині завдання

**Аналіз вимог і проектування**

Графове представлення скінченних автоматів (1 частина): .



**Тексти коду програм**

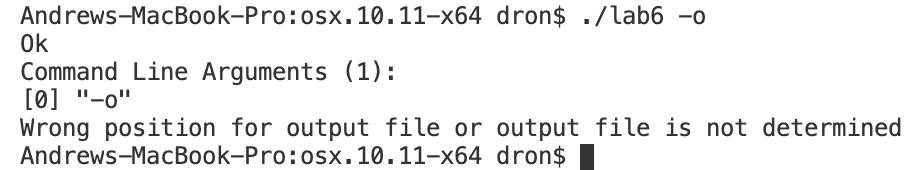
|  |
| --- |
| **Program.cs** |
| using System;  using static System.Console;  using System.IO;  using System.Diagnostics;  namespace lab6  {  enum State  {  initialization,  openQuote,  closeQuote,  chars,  }  struct Options  {  public bool isInteractiveMode; // for -i boolean option  public string inputFile; // for the independent option  public string outputFile; // for -o value option  public string parsingError; // for errors  }  class Program  {  static void Main(string[] args) // main part  {  RunTests();  if(args.Length == 0) // checking are there any arguments inputed  {  WriteLine("No arguments were inputed.");  return;  }  Console.WriteLine("Command Line Arguments ({0}):", args.Length); // printing arguments them  for (int i = 0; i < args.Length; i++)  {  Console.WriteLine("[{0}] \"{1}\"", i, args[i]);  }  Options options = ParseOptions(args); // parsing command line input into struct options  if(options.parsingError != "")  {  WriteLine(options.parsingError);  return;  }  string input;  if(options.isInteractiveMode == true)  {  WriteLine("Interactive mode is ON");  do  {  WriteLine("Input your string");  input = ReadLine();  if(CheckString(input) == false)  {  WriteLine("Threse is not one string entity");  }  else  {  WriteLine("There is one string entity");  }  WriteLine("Number of strings in your input is {0}", CountStrings(input));  string[] arrayOfStrings = GetAllStrings(input);  for(int i = 0; i < arrayOfStrings.Length; i++)  {  WriteLine("{0}", arrayOfStrings[i]);  }  }  while(input != "");  }  else  {  if(options.inputFile == "")  {  WriteLine("You should chose interactive mode or input input file");  return;  }  else if(options.inputFile != "" && options.outputFile == "")  {  string text = File.ReadAllText(options.inputFile);  string[] arrayForText = GetAllStrings(text);  for (int i = 0; i < arrayForText.Length; i++)  {  WriteLine("{0} = '{1}'", i, arrayForText[i]);  }  }  else if(options.inputFile != "" && options.outputFile != "")  {  string text = File.ReadAllText(options.inputFile);  string[] arrayForText = GetAllStrings(text);  string line = "";  for (int i = 0; i < arrayForText.Length; i++)  {  line = line + arrayForText[i];  }  File.WriteAllText(options.outputFile, line);  }  }  }  static bool CheckString(string input)  {  State state = State.initialization;  int counterOfStrings = 0;  int counterOfQuotes = 0;  int counterOfSymbols = 0;  foreach(char symbol in input)  {  if(state == State.initialization)  {  if(symbol == '@')  {  state = State.openQuote;  }  counterOfSymbols++;  }  else if(state == State.openQuote)  {  if(symbol != '"')  {  counterOfStrings = 0;  break;  }  else  {  counterOfQuotes++;  counterOfSymbols++;  state = State.chars;  }  }  else if(state == State.chars)  {  if(symbol == '"')  {  if(counterOfSymbols != input.Length - 1)  {  counterOfStrings = 0;  break;  }  else if(symbol == '@')  {  counterOfStrings = 0;  break;  }  else  {  counterOfStrings++;  }  }  counterOfSymbols++;  }  }  if(counterOfStrings == 1)  {  return true;  }  else  {  return false;  }  }  static int CountStrings(string input)  {  State state = State.initialization;  int counter = 0;  foreach(char symbol in input)  {  if(state == State.initialization)  {  if(symbol == '@')  {  state = State.openQuote;  }  else if(symbol == '"' || char.IsPunctuation(symbol) == false && char.IsSeparator(symbol) == false  && symbol != '\t' && symbol != '\n')  {  counter--;  }  }  else if(state == State.openQuote)  {  if(symbol == '"')  {  state = State.closeQuote;  }  }  else if(state == State.closeQuote)  {  if(symbol == '"')  {  counter++;  state = State.initialization;  }  }  }  if(counter < 0)  {  return 0;  }  return counter;  }  static string[] GetAllStrings(string input)  {  State state = State.initialization;  string currentString = "";  string[] arrayOfStrings = new string[CountStrings(input)];  int counter = 0;  foreach(char symbol in input)  {  if(state == State.initialization)  {  if(symbol == '@' || char.IsPunctuation(symbol) || char.IsSeparator(symbol) || symbol == '\t' || symbol == '\n')  {  if(currentString != "")  {  arrayOfStrings[counter] = currentString;  currentString = "";  counter++;  }  state = State.openQuote;  currentString = currentString + symbol;  }  else if(symbol == '"' || char.IsPunctuation(symbol) == false && char.IsSeparator(symbol) == false  && symbol != '\t' && symbol != '\n')  {  currentString = "";  }  }  else if(state == State.openQuote)  {  if(symbol == '"')  {  state = State.closeQuote;  currentString = currentString + symbol;  }  else  {  state = State.initialization;  currentString = "";  }  }  else if(state == State.closeQuote)  {  if(symbol == '"')  {  state = State.initialization;  currentString = currentString + symbol;  }  else  {  currentString = currentString + symbol;  }  }  }  return arrayOfStrings;  }  static Options ParseOptions(string[] array) // function which parse command line input into struct options  {  Options options = new Options(); // variable for new options from command line  {  options.isInteractiveMode = false;  options.inputFile = "";  options.outputFile = "";  options.parsingError = "";  }  for(int i = 0; i < array.Length; i++)  {  if(array[i] == "-i")  {  options.isInteractiveMode = true;  }  else if(array[i] == "-o")  {  if(i != array.Length - 1 && array[i + 1].StartsWith('-') == false)  {  options.outputFile = array[i + 1];  }  else  {  options.parsingError = "Wrong position for output file or output file is not determined";  }  i++;  }  else if(array[i].StartsWith("-") && array[i].StartsWith("-o") == false && array[i].StartsWith("-i") == false)  {  options.parsingError = "Unknown command";  }  else  {  options.inputFile = array[i];  }  }  return options;  }  static bool CompareArrays(string[] array1, string[] array2)  {  if (array1.Length != array2.Length)  {  return false;  }  for (int i = 0; i < array1.Length; i++)  {  if (array1[i] != array1[i])  {  return false;  }  }  return true;  }  static bool CompareOptions(in Options options1, in Options options2)  {  return options1.isInteractiveMode == options2.isInteractiveMode  && options1.inputFile == options2.inputFile  && options1.outputFile == options2.outputFile;  }  static void RunTests()  {  Debug.Assert(CheckString("@\"123\"") == true);  Debug.Assert(CheckString("fwrgg") == false);  Debug.Assert(CheckString("gkks@\"12") == false);  Debug.Assert(CountStrings("@\"friri\"@\"fkkg\"@fkrkf\"") == 3);  Debug.Assert(CountStrings("@\"fjgk@\"") == 1);  Debug.Assert(CountStrings("@\"124lg")== 0);  Debug.Assert(CompareArrays(GetAllStrings("\'@1234\""),new string[] {}));  Debug.Assert(CompareArrays(GetAllStrings("@\"243\"@\"ferfr\"@111\""),new string[] {@"243", @"ferfr"}));  Debug.Assert(CompareArrays(GetAllStrings("@\"fwlefwel\" "), new string[] {@"fwleflwel"}));  Debug.Assert(CompareOptions(ParseOptions(new string[]{"-o"}), new Options  {  isInteractiveMode = false,  inputFile = "",  outputFile = "",  parsingError = ""  }));  Debug.Assert(CompareOptions(ParseOptions(new string[]{"test.txt"}), new Options  {  isInteractiveMode = false,  inputFile = "test.txt",  outputFile = "",  parsingError = ""  }));  Debug.Assert(CompareOptions(ParseOptions(new string[] {"-k"}), new Options  {  isInteractiveMode = false,  inputFile = "",  outputFile = "",  parsingError = "Unknown command"  }));  WriteLine("Ok");  }  }  } |

**Приклади результатів програм**

****

**Програма проходить усі тести.**

**Ще приклад роботи програми для некорректних агрументів рядка:**

****

**Висновки**

Протягом виконання цієї лабораторної роботи, скінчені автомати були використані для реалізації декількох алгоритмів обробки тексту. Також велася робота з аргументами командного рядка, використання яких було необхідно при розв’язанні завдань другої частини лабораторної.