

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-01 | | Пецеля Артем Володимирович | | (*прізвище, ім’я, по батькові*) |   варіант №12 |  | 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**

*Загальні вимоги до завдання*

Реалізувати консольну програму для розбору тексту і пошуку заданих сутностей.

Сутності за варіантом: **неповний запис літералів дробових чисел - ціла частина** - починаються на непусту послідовність десяткових цифр і обов'язково закінчуються на символ точки.

*Частина 1. Символи*

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. Рядки символів*

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*:

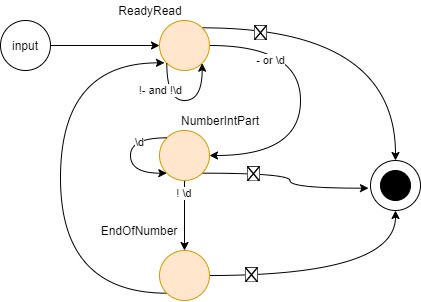
1. Якщо булева опція задана декілька раз - це нормально і вона вважається включеною
2. Якщо опція зі значенням задана декілька раз - це нормально і враховувати останнє таке значення.
3. Якщо задано декілька вільних аргументів командного рядка - поведінка програми на ваш розсуд: помилка, обробляти перший, 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**

Граф використаного автомату для зчитування рядка:

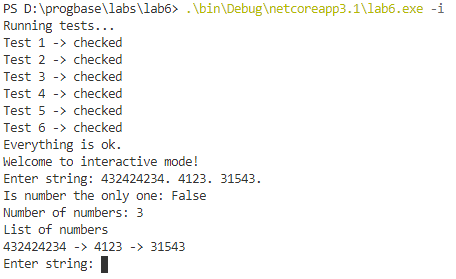
****

**Тексти коду програм**

|  |
| --- |
| **Program.cs** |
| using System;  using static System.Console;  using System.Diagnostics;  using static System.IO.File;  namespace lab6  {  class SLList  {  public Node head;  public class Node  {  public string data;  public Node next;  public Node(string data)  {  this.data = data;  }  public Node(string data, Node next)  {  this.data = data;  this.next = next;  }  }  public SLList(string data)  {  head = new Node(data);  }  public int GetLength()  {  Node current = head;  int counter = 0;  while (current != null)  {  counter++;  current = current.next;  }  return counter;  }  public string GetValue(int num)  {  Node current = head;  for (int i = 0; i < num; i++)  {  current = current.next;  }  return current.data;  }  public void AddLast(string data)  {  if (head == null)  {  head = new Node(data);  }  else  {  Node current = head;  while (current.next != null)  {  current = current.next;  }  current.next = new Node(data);  }  }  public void DeleteFirst()  {  if (head != null)  head = head.next;  }  public void DeleteLast()  {  if (head != null && head.next != null)  {  Node current = head;  while (current.next.next != null)  {  current = current.next;  }  current.next = null;  }  else head = null;  }  public void Print()  {  Node current = head;  while (current != null)  {  Console.Write(current.data);  if (current.next != null) Console.Write(" -> ");  current = current.next;  }  WriteLine();  }  public string GetString()  {  string output = "";  Node current = head;  while (current != null)  {  output += current.data;  if (current.next != null)  output += " -> ";  current = current.next;  }  return output;  }  }  enum State  {  Stop,  ReadyRead,  NumberIntPart,  EndOfNumber,  }  struct Options  {  public bool isInteractiveMode; // for -i boolean option  public string inputFile; // for the independent option  public string outputFile; // for -o value option  // for errors  public bool hasParsingError;  public string parsingError;  }  class Program  {  static State state;  static string buffer = "";  static SLList output;  static void RunTests()  {  WriteLine("Running tests...");  Write("Test 1 -> ");  string input1 = "4324.";  Debug.Assert(CheckIntegerPart(input1) == true);  Debug.Assert(CountIntegerPart(input1) == 1);  SLList expectedOutput1 = new SLList("4324");  GetAllNumbers(input1);  Debug.Assert(CompareSLLists(output, expectedOutput1) == true);  WriteLine("checked");  Write("Test 2 -> ");  string input2 = "412345. fds 12345. 5421. 564";  Debug.Assert(CheckIntegerPart(input2) == false);  Debug.Assert(CountIntegerPart(input2) == 2);  SLList expectedOutput2 = new SLList("12345");  expectedOutput2.AddLast("5421");  GetAllNumbers(input2);  Debug.Assert(CompareSLLists(output, expectedOutput2) == true);  WriteLine("checked");  Write("Test 3 -> ");  string input3 = "dhsiufh fdksjnfsu fdsjnfeed. 432. fdjdjgfdo 3242.";  Debug.Assert(CheckIntegerPart(input3) == false);  Debug.Assert(CountIntegerPart(input3) == 1);  SLList expectedOutput3 = new SLList("3242");  GetAllNumbers(input3);  Debug.Assert(CompareSLLists(output, expectedOutput3) == true);  WriteLine("checked");  Write("Test 4 -> ");  string input4 = "1234. abc 435.";  Debug.Assert(CheckIntegerPart(input4) == false);  Debug.Assert(CountIntegerPart(input4) == 1);  SLList expectedOutput4 = new SLList("435");  GetAllNumbers(input4);  Debug.Assert(CompareSLLists(output, expectedOutput4) == true);  WriteLine("checked");  Write("Test 5 -> ");  string input5 = "4a2.";  Debug.Assert(CheckIntegerPart(input5) == false);  Debug.Assert(CountIntegerPart(input5) == 1);  SLList expectedOutput5 = new SLList("2");  GetAllNumbers(input5);  Debug.Assert(CompareSLLists(output, expectedOutput5) == true);  WriteLine("checked");  Write("Test 6 -> ");  string input6 = "4 .";  Debug.Assert(CheckIntegerPart(input6) == false);  Debug.Assert(CountIntegerPart(input6) == 0);  GetAllNumbers(input6);  Debug.Assert(output.head == null);  WriteLine("checked");  WriteLine("Everything is ok.");  }  static void Main(string[] args)  {  RunTests();  Options options = ParseOptions(args);  if (options.hasParsingError)  {  WriteLine("Error: {0}", options.parsingError);  }  else if (options.isInteractiveMode)  {  WriteLine("Welcome to interactive mode!");  bool exit = false;  while (!exit)  {  Write("Enter string: ");  string str = ReadLine();  if (str == "")  exit = true;  else  {  Write("Is number the only one: ");  WriteLine(CheckIntegerPart(str));  Write("Number of numbers: ");  WriteLine(CountIntegerPart(str));  GetAllNumbers(str);  WriteLine("List of numbers");  output.Print();  }  }  }  else  {  string str = ReadAllText(options.inputFile);  string outputString = "Is number the only one: ";  if (CheckIntegerPart(str))  {  outputString += "True\n";  }  else  {  outputString += "False\n";  }  outputString += String.Format("Number of numbers: {0}{1}", CountIntegerPart(str), "\n");  GetAllNumbers(str);  outputString += "List of numbers:\n" + output.GetString();  if (options.outputFile != null)  {  WriteAllText(options.outputFile, outputString);  }  else  {  WriteLine(outputString);  }  }  }  static bool CompareSLLists(SLList list1, SLList list2)  {  int length1 = list1.GetLength();  int length2 = list2.GetLength();  if (length1 != length2)  return false;  bool flag = false;  for (int i = 0; i < length1; i++)  {  string value1 = list1.GetValue(i);  string value2 = list2.GetValue(i);  if (value1 != value2)  {  flag = true;  break;  }  }  if (flag)  return false;  else  return true;  }  // func 1  static bool CheckIntegerPart(string input)  {  state = State.ReadyRead;  int scanIndex = 0;  bool error = false;  bool isDot = false;  while (state != State.Stop && error == false)  {  switch (state)  {  case State.ReadyRead:  OnReadyReadOneNumber(input, ref scanIndex, ref error);  break;  case State.NumberIntPart:  ReadOneNumberInt(input, ref scanIndex, ref error, ref isDot);  break;  }  }  if (error)  return false;  else  return true;  }  static void OnReadyReadOneNumber(string input, ref int scanIndex, ref bool error)  {  if (scanIndex == input.Length)  {  state = State.Stop;  }  else if (!Char.IsDigit(input[scanIndex]))  {  error = true;  }  else  {  state = State.NumberIntPart;  }  }  static void ReadOneNumberInt(string input, ref int scanIndex, ref bool error, ref bool isDot)  {  if (scanIndex == input.Length && isDot)  {  state = State.Stop;  }  else if (scanIndex == input.Length && !isDot)  {  error = true;  }  else if (Char.IsDigit(input[scanIndex]))  {  scanIndex += 1;  }  else if (input[scanIndex] == '.')  {  isDot = true;  scanIndex += 1;  }  else  {  error = true;  }  }  // func 2  static int CountIntegerPart(string input)  {  state = State.ReadyRead;  int scanIndex = 0;  int counter = 0;  bool spaceBetween = false;  bool dotAfter = false;  while (state != State.Stop)  {  switch (state)  {  case State.ReadyRead:  OnReadyRead(input, ref scanIndex);  break;  case State.NumberIntPart:  dotAfter = false;  spaceBetween = false;  ReadNumberToCount(input, ref scanIndex);  break;  case State.EndOfNumber:  ReadEndOfNumber(input, ref scanIndex, ref counter, ref spaceBetween, ref dotAfter);  break;  }  }  return counter;  }  static void OnReadyRead(string input, ref int scanIndex)  {  if (scanIndex == input.Length)  {  state = State.Stop;  }  else if (Char.IsDigit(input[scanIndex]))  {  state = State.NumberIntPart;  }  else if (input[scanIndex] == ' ')  {  scanIndex += 1;  }  else  {  scanIndex++;  }  }  static void ReadNumberToCount(string input, ref int scanIndex)  {  if (scanIndex == input.Length)  {  state = State.Stop;  }  else if (Char.IsDigit(input[scanIndex]))  {  scanIndex++;  }  else  {  state = State.EndOfNumber;  }  }  static void ReadEndOfNumber(string input, ref int scanIndex, ref int counter, ref bool spaceBetween, ref bool dotAfter)  {  if (scanIndex == input.Length)  {  state = State.Stop;  counter++;  }  else if (input[scanIndex] == '.' && !dotAfter)  {  dotAfter = true;  scanIndex++;  }  else if (input[scanIndex] != '.' && !dotAfter)  {  if (counter != 0)  counter--;  state = State.ReadyRead;  }  else if ((input[scanIndex] == ' ' || Char.IsPunctuation(input[scanIndex])) && !spaceBetween)  {  scanIndex++;  spaceBetween = true;  }  else if (input[scanIndex] == ' ' || Char.IsPunctuation(input[scanIndex]))  {  scanIndex++;  }  else if (Char.IsDigit(input[scanIndex]) && spaceBetween && dotAfter)  {  counter++;  state = State.NumberIntPart;  }  else if (Char.IsDigit(input[scanIndex]) && !spaceBetween)  {  state = State.NumberIntPart;  }  else  {  state = State.ReadyRead;  }  }  // func 3  static void GetAllNumbers(string input)  {  output = new SLList("");  state = State.ReadyRead;  int scanIndex = 0;  bool spaceBetween = false;  bool dotAfter = false;  while (state != State.Stop)  {  switch (state)  {  case State.ReadyRead:  OnReadyRead(input, ref scanIndex);  break;  case State.NumberIntPart:  spaceBetween = false;  dotAfter = false;  ReadNumber(input, ref scanIndex);  break;  case State.EndOfNumber:  ReadEndOfNumber(input, ref scanIndex, ref spaceBetween, ref dotAfter);  break;  }  }  if (output.GetValue(0) == "")  output.DeleteFirst();  }  static void ReadNumber(string input, ref int scanIndex)  {  if (scanIndex == input.Length)  {  state = State.Stop;  }  else if (Char.IsDigit(input[scanIndex]))  {  SaveToBuffer(input[scanIndex]);  scanIndex++;  }  else  {  state = State.EndOfNumber;  }  }  static void SaveToBuffer(char c)  {  buffer += c;  }  static void ClearBuffer()  {  buffer = "";  }  static void Dump()  {  output.AddLast(buffer);  buffer = "";  }  static void ReadEndOfNumber(string input, ref int scanIndex, ref bool spaceBetween, ref bool dotAfter)  {  if (scanIndex == input.Length)  {  state = State.Stop;  Dump();  }  else if (input[scanIndex] == '.' && !dotAfter)  {  dotAfter = true;  scanIndex++;  }  else if (input[scanIndex] != '.' && !dotAfter)  {  ClearBuffer();  if (output.GetValue(0) != "")  output.DeleteLast();  state = State.ReadyRead;  }  else if ((input[scanIndex] == ' ' || Char.IsPunctuation(input[scanIndex])) && !spaceBetween)  {  scanIndex++;  spaceBetween = true;  }  else if (input[scanIndex] == ' ' || Char.IsPunctuation(input[scanIndex]))  {  scanIndex++;  }  else if (Char.IsDigit(input[scanIndex]) && spaceBetween && dotAfter)  {  Dump();  state = State.NumberIntPart;  }  else if (Char.IsDigit(input[scanIndex]) && !spaceBetween)  {  ClearBuffer();  state = State.NumberIntPart;  }  else  {  ClearBuffer();  state = State.ReadyRead;  }  }  static Options ParseOptions(string[] args)  {  Options options = new Options {isInteractiveMode = false, inputFile = "", outputFile = ""};  bool[] isParsedArr = new bool[args.Length];  for (int i = 0; i < args.Length; i++)  {  if (args[i].Length > 1)  {  if (args[i] == "-i")  {  options.isInteractiveMode = true;  isParsedArr[i] = true;  }  else if (args[i] == "-o")  {  if (i + 1 < args.Length)  {  if (args[i + 1].Contains(".txt"))  {  options.outputFile = args[i + 1];  isParsedArr[i] = true;  isParsedArr[i + 1] = true;  }  else  {  options.hasParsingError = true;  options.parsingError = "argument -o has wrong text file";  return options;  }  }  else  {  options.hasParsingError = true;  options.parsingError = "argument -o has no params";  return options;  }  }  }  else  {  options.hasParsingError = true;  options.parsingError = String.Format("unrecognized argument {0}", args[i]);  return options;  }  }  if (CountFalses(isParsedArr) == 1)  {  for (int i = 0; i < args.Length; i++)  {  if (!isParsedArr[i])  {  if (args[i].Contains(".txt"))  {  options.inputFile = args[i];  break;  }  else  {  options.hasParsingError = true;  options.parsingError = "unrecognized input file";  return options;  }  }  }  }  else if (CountFalses(isParsedArr) == 0)  {  options.inputFile = "";  }  else  {  options.hasParsingError = true;  options.parsingError = "too many args";  return options;  }  if (options.isInteractiveMode || options.inputFile != "")  {  return options;  }  else  {  options.hasParsingError = true;  options.parsingError = "there must be either interactive or standart mode activated";  return options;  }  }  static int CountFalses(bool[] array)  {  int counter = 0;  foreach (bool b in array)  {  if (!b)  counter++;  }  return counter;  }  }  } |

**Приклади результатів**

Приклад 1. Робота програми в інтерактивному режимі



Приклад 2. Робота програми в стандартному режимі



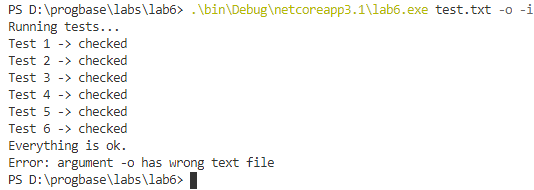
Вміст файлу test.txt

|  |
| --- |
| **test.txt** |
| 4324. 4324. 432423. |

Вміст файлу outputText.txt

|  |
| --- |
| **outputText.txt** |
| Is number the only one: False  Number of numbers: 3  List of numbers:  4324 -> 4324 -> 432423 |

Приклад 3. Некоректний ввід аргументів у консоль



**Висновки**

Виконавши дану лабораторну роботу було створено програму, що обробляє заданий рядок за допомогою скінченного автомата та виводить результат у вигляді списку сутностей, а саме цілої частини числа. Також було застосовано аргументи командного рядка. Ввід користувача перевірявся та після перевірки програма виконувала задані вказівки.

Компіляція всього коду відбувалася за допомогою утиліти dotnet.