

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  Беліцький Олександр Сергійович  (*прізвище, ім’я, по батькові*)  варіант № 3 |  | 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: **неповний запис літералів дробових чисел - ціла частина** - починаються на непусту послідовність десяткових цифр і обов'язково закінчуються на символ точки.

#### **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.
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 - помилка.

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**

#### **Частина 1**

Для обробки рядка реалізувати алгоритми з використанням скінченних автоматів у чистих функціях:

*// function #1*

**static** **bool** **CheckIntegers**(**string** input) { }

*// function #2*

**static** **int** **CountIntegers**(**string** input) { }

*// function #3*

**static** **string**[] **GetAllIntegers**(**string** input) { }

Створити функцію:

**static** **void** **RunTests**() { }

У цій функції написати Debug.Assert() перевірки для кожної з трьох функцій першої частини завдання з різними вхідними даними.  
Викликати цю функцію як першу команду в Main()

У функції #3 створити великий масив фіксованого розміру зі змінною, що позначає заповнення масиву. Після завершення пошуку сутностей створити новий масив, що буде містити всі знайдені сутнсті і повернути його.  
Можна також замість великого масиву використати малий із перестворенням зі збільшенням розміру та копіюванням, або використати власну реалізацію зв'язного списку.

Обмеження реалізації функцій:

1. Кожен символ вхідного рядка перевіряється тільки 1 раз.
2. Можна доступатись тільки до одного символа з рядка за раз.
3. Заборонено перевіряти індекс символів (наприклад, if (index == 0)).
4. Заборонено використовувати стандартні функції для рядків (StartsWith, Contains, тощо).
5. Заборонено використовувати функцію #3 при реалізації #2.
6. Заборонено використовувати регулярні вирази в коді.
7. Не можна використовувати стандартні динамічні структури даних (List<T> і т.п.).

#### **Частина 2**

Описати структуру Options, поля якої відповідають опціям, що задані у завданні. Наприклад:

**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** **string** parsingError;

}

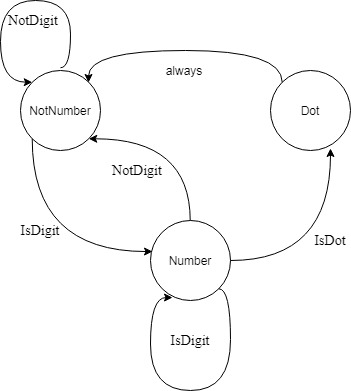
Реалізувати чисту функцію, що на основі вхідного масиву аргументів командного рядка повертає створене значення структури:

**static** Options **ParseOptions**(**string**[] args) { }

У випадку помилки розбору заповніть додаткові поля структури Options.

У Main(), після виклику ParseOptions(), перевіряйте наявність помилки та отримані опції програми.

Графічне представлення використаного скінченного автомата:



**Тексти коду програм**

|  |
| --- |
| **Program.cs** |
| using System;  using System.Diagnostics;  using System.IO;  namespace lab\_6.\_1\_try  {  class Program  {  enum State  {  NotNumber,  Dot,  Number  }  struct Options  {  public bool isInteractiveMode;  public string inputFile;  public string outputFile;  public string parsingError;  }  static void Main(string[] args)  {  RunTests();  Console.WriteLine("Command Line Arguments ({0}):", args.Length);  for (int i = 0; i < args.Length; i++)  {  Console.WriteLine("[{0}] \"{1}\"", i, args[i]);  }  Options options = ParseOptions(args);  if (options.parsingError != "")  {  Console.WriteLine(options.parsingError);  Environment.Exit(0);  }  else  {  Console.WriteLine($"Interactive mode: {options.isInteractiveMode}");  if (options.isInteractiveMode == false)  {  Console.WriteLine($"Input file: {options.inputFile}");  Console.WriteLine($"Output: {options.outputFile}");  }  }  if (options.isInteractiveMode == true)  {  do  {  Console.Write("Enter string of nums: ");  string input = Console.ReadLine();  if (input == null || input.Length == 0)  {  Console.WriteLine("Ending processing...");  Environment.Exit(0);  }  else  {  Console.WriteLine("Чи рядок повністю складається з однієї сутності - {0}", CheckIntegers(input));  Console.WriteLine("Кількість сутностей у вхідному рядку - {0}", CountIntegers(input));  if (CountIntegers(input) != 0)  {  PrintIntegers(input);  }  }  Console.WriteLine();  } while (true);  }  else  {  string inFilePath = $"/home/alexandr/progbase/progbase/labs/lab 6/{options.inputFile}";  var isExistInFile = File.Exists(inFilePath);  string outFilePath = $"/home/alexandr/progbase/progbase/labs/lab 6/{options.outputFile}";  var isExistOutFile = File.Exists(outFilePath);  if (isExistInFile)  {  string text = File.ReadAllText(options.inputFile);  if (isExistOutFile)  {  string[] integers = GetAllIntegers(text);  string newFileText = string.Join(" ", integers);  File.WriteAllText(options.outputFile, newFileText);  }  else  {  PrintIntegers(text);  }  }  else  {  Console.WriteLine("Error: Input file doesn't exist");  }  }  }  static void RunTests()  {  //CountIntegers  Debug.Assert(CountIntegers("") == 0, "Empty sring");  Debug.Assert(CountIntegers("gsdsf") == 0, "Only letters");  Debug.Assert(CountIntegers("444b.") == 0, "Can't contain letter");  Debug.Assert(CountIntegers("4334") == 0, "Must contain dot");  Debug.Assert(CountIntegers("44.44") == 0, "Can't contain fraction");  Debug.Assert(CountIntegers("44.bb") == 0, "Can't contain letters after dot");  Debug.Assert(CountIntegers("44. bb.") == 1, "Can't contain letters after dot");  Debug.Assert(CountIntegers("43. ") == 1, "Only one number");  Debug.Assert(CountIntegers("43. 4. 543453.") == 3, "Three numbers");  Debug.Assert(CountIntegers("..\*\*..$.%.sd.") == 0, "Must contain integer number");  Debug.Assert(CountIntegers(".......") == 0, "Must contain int number not only dots");  Debug.Assert(CountIntegers("43.,645. 8.") == 3, "Punctual is normal");  //CheckIntegers  Debug.Assert(CheckIntegers("33.") == true);  Debug.Assert(CheckIntegers("33. 43.") == false);  Debug.Assert(CheckIntegers("dds") == false);  Debug.Assert(CheckIntegers("43r.") == false);  Debug.Assert(CheckIntegers("33. dd.") == false);  Debug.Assert(CheckIntegers(".") == false);  Debug.Assert(CheckIntegers(".%>#$@)#$@\*$.") == false);  Debug.Assert(CheckIntegers("33.bb") == false);  Debug.Assert(CheckIntegers("33.,") == false);  //GetAllIntegers  Debug.Assert(CompareArrays(GetAllIntegers(""), new string[0] { }));  Debug.Assert(CompareArrays(GetAllIntegers("43. "), new string[1] { "43." }));  Debug.Assert(CompareArrays(GetAllIntegers("44. gg."), new string[1] { "44." }));  Debug.Assert(CompareArrays(GetAllIntegers("....#@$#Y$#@!I$O."), new string[0] { }));  Debug.Assert(CompareArrays(GetAllIntegers("44. 23."), new string[2] { "44.", "23." }));  Debug.Assert(CompareArrays(GetAllIntegers("324.,"), new string[1] { "324." }));  Debug.Assert(CompareArrays(GetAllIntegers("werwer."), new string[0] { }));  //Options  Debug.Assert(CompareOptions(ParseOptions(new string[] { "example.txt", "-i" }), new Options  {  isInteractiveMode = true,  inputFile = "",  outputFile = "",  parsingError = "",  }));  Debug.Assert(CompareOptions(ParseOptions(new string[] { "-o", "example.txt", "-i" }), new Options  {  isInteractiveMode = true,  inputFile = "",  outputFile = "",  parsingError = "",  }));  Debug.Assert(CompareOptions(ParseOptions(new string[] { "-i", "-o", "example.txt" }), new Options  {  isInteractiveMode = true,  inputFile = "",  outputFile = "",  parsingError = "",  }));  Debug.Assert(CompareOptions(ParseOptions(new string[] { "" }), new Options  {  isInteractiveMode = false,  inputFile = "",  outputFile = "",  parsingError = "",  }));  Debug.Assert(CompareOptions(ParseOptions(new string[] { "-o", "example.txt" }), new Options  {  isInteractiveMode = false,  inputFile = "",  outputFile = "",  parsingError = "Error: must be Interactive mode or input file",  }));  Debug.Assert(CompareOptions(ParseOptions(new string[] { "example.txt" }), new Options  {  isInteractiveMode = false,  inputFile = "example.txt",  outputFile = "",  parsingError = "",  }));  Debug.Assert(CompareOptions(ParseOptions(new string[] { "-i", "-i" }), new Options  {  isInteractiveMode = true,  inputFile = "",  outputFile = "",  parsingError = "",  }));  Debug.Assert(CompareOptions(ParseOptions(new string[] { "-o", "val1", "-o", "val2", "Val" }), new Options  {  isInteractiveMode = false,  inputFile = "Val",  outputFile = "val2",  parsingError = "",  }));  Debug.Assert(ParseOptions(new string[] { "-o" }).parsingError != "");  Debug.Assert(ParseOptions(new string[] { "-u" }).parsingError != "");  Debug.Assert(ParseOptions(new string[] { "-o", "-i" }).parsingError == "");  Debug.Assert(ParseOptions(new string[] { "example1", "examlpe2" }).parsingError != "");  }  static bool CheckIntegers(string str)  {  bool ifOne = false;  int counter = CountIntegers(str);  string[] integers = new string[100];  string[] structures = new string[counter];  integers = str.Split(' ', ',', ';');  State state = State.NotNumber;  if (integers.Length != 1)  {  return ifOne;  }  else  {  for (int i = 0; i < integers.Length; i++)  {  string stringInt = integers[i];  state = State.NotNumber;  for (int j = 0; j < integers[i].Length; j++)  {  char c = stringInt[j];  if (state == State.NotNumber)  {  if (char.IsDigit(c))  {  state = State.Number;  }  }  else if (state == State.Number)  {  if ((c == '.') && (j == integers[i].Length - 1))  {  state = State.Dot;  ifOne = true;  }  else if (!char.IsDigit(c))  {  state = State.NotNumber;  }  }  else if (state == State.Dot)  {  state = State.NotNumber;  }  }  }  }  return ifOne;  }  static int CountIntegers(string str)  {  int counter = 0;  string[] integers = new string[100];  integers = str.Split(' ', ',', ';');  State state = State.NotNumber;  for (int i = 0; i < integers.Length; i++)  {  string stringInt = integers[i];  state = State.NotNumber;  int dots = 0;  for (int j = 0; j < integers[i].Length; j++)  {  char c = stringInt[j];  if (c == '.')  dots++;  }  if (dots != 1)  {  continue;  }  for (int j = 0; j < integers[i].Length; j++)  {  char c = stringInt[j];  if (state == State.NotNumber)  {  if (char.IsDigit(c))  {  state = State.Number;  }  }  else if (state == State.Number)  {  if ((c == '.') && (j == integers[i].Length - 1))  {  state = State.Dot;  counter++;  }  else if (!char.IsDigit(c))  {  state = State.NotNumber;  }  }  else if (state == State.Dot)  {  state = State.NotNumber;  }  }  }  return counter;  }  static string[] GetAllIntegers(string str)  {  int counter = CountIntegers(str);  string[] integers = new string[100];  string[] structures = new string[counter];  integers = str.Split(' ', ',', ';');  State state = State.NotNumber;  int k = 0;  for (int i = 0; i < integers.Length; i++)  {  string stringInt = integers[i];  state = State.NotNumber;  int dots = 0;  for (int j = 0; j < integers[i].Length; j++)  {  char c = stringInt[j];  if (c == '.')  dots++;  }  if (dots != 1)  {  continue;  }  for (int j = 0; j < integers[i].Length; j++)  {  char c = stringInt[j];  if (state == State.NotNumber)  {  if (char.IsDigit(c))  {  state = State.Number;  }  }  else if (state == State.Number)  {  if ((c == '.') && (j == integers[i].Length - 1))  {  state = State.Dot;  structures[k] = integers[i];  k++;  }  else if (!char.IsDigit(c))  {  state = State.NotNumber;  }  }  else if (state == State.Dot)  {  state = State.NotNumber;  }  }  }  return structures;  }  static void PrintIntegers(string str)  {  string[] array = GetAllIntegers(str);  int counter = CountIntegers(str);  for (int i = 0; i < counter; i++)  {  Console.WriteLine("[{0}] -> {1}", i, array[i]);  }  }  static bool CompareArrays(string[] arr1, string[] arr2)  {  if (arr1.Length != arr2.Length)  {  return false;  }  for (int i = 0; i < arr1.Length; i++)  {  if (arr1[i] != arr2[i])  {  return false;  }  }  return true;  }  static bool CompareOptions(Options o1, Options o2)  {  return o1.isInteractiveMode == o2.isInteractiveMode  && o1.inputFile == o2.inputFile  && o1.outputFile == o2.outputFile  && o1.parsingError == o2.parsingError;  }  static Options ParseOptions(string[] args)  {  Options options = new Options()  {  isInteractiveMode = false,  inputFile = "",  outputFile = "",  parsingError = "",  };  if (args.Length == 0)  {  return options;  }  bool[] isUsed = new bool[args.Length];  while (true)  {  bool hasUnUsed = false;  foreach (bool isUsedItem in isUsed)  {  if (isUsedItem == false)  {  hasUnUsed = true;  break;  }  }  if (hasUnUsed == false)  {  break;  }  for (int i = 0; i < args.Length; i++)  {  if (isUsed[i])  {  continue;  }  if (args[i] == "-i")  {  options.isInteractiveMode = true;  isUsed[i] = true;  }  }  if (options.isInteractiveMode == true)  break;  for (int i = 0; i < args.Length; i++)  {  if (isUsed[i])  {  continue;  }  if (args[i] == "-o")  {  if (i == args.Length - 1 || args[i + 1].StartsWith('-'))  {  options.parsingError = "Error: `-o` опція із аргументом, задайте назву файлу";  return options;  }  options.outputFile = args[i + 1];  isUsed[i] = true;  isUsed[i + 1] = true;  }  else if (args[i].StartsWith('-'))  {  options.parsingError = $"error: unrecognized option '{args[i]}'";  return options;  }  else  {  if (options.inputFile != "")  {  options.parsingError = "error: there are two input files";  return options;  }  options.inputFile = args[i];  isUsed[i] = true;  }  }  }  if (args[0] == "")  {  return options;  }  else if ((options.isInteractiveMode == false) && (options.inputFile == ""))  {  options.parsingError = "Error: must be Interactive mode or input file";  options.isInteractiveMode = false;  options.inputFile = "";  options.outputFile = "";  }  return options;  }  }  } |

**Приклади результатів**

|  |
| --- |
| **examplein.txt - input file** |
|  |

|  |
| --- |
| **./bin/Debug/netcoreapp3.1/"lab 6" -o -i -g** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" examplein.txt** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" -o exampleout.txt** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" -o exampleout.txt examplein.txt** |
| *exampleout.txt* before and after |
| **./bin/Debug/netcoreapp3.1/"lab 6" unknownfile.txt** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" -o unknownfile.txt examplein.txt** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" -o exampleout.txt -o exampleout1.txt examplein.txt** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" -p** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" -o** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" examplein.txt examplein1.txt** |
|  |
| **./bin/Debug/netcoreapp3.1/"lab 6" -i //empty string** |
|  |

**Висновки**

Скінченний автомат (finite-state machine) — особливий різновид автомата, що використовується для опису шляху зміни стану об'єкта в залежності від поточного стану та інформації отриманої ззовні. Його особливістю є скінченність множини станів автомату, оскільки будь-який пристрій може мати тільки скінченну кількість станів.

В C# для налагодження коду використовується Debug.Assert. Перевіряється умова. Якщо умова має значення false, з'являється повідомлення із відображенням повідомлення зі стеком викликів. Належить до простору імен System.Diagnostics.

Компіляція всього коду відбувалася за допомогою утиліти dotnet.