захвата, и уменьшается его вероятность. Эти два конкурирующих эффекта приводят к появлению локального максимума на графике. Таким образом, с помощью моделирования можно подобрать самую подходящую энергию нейтронов для конкретной задачи облучения.

5. Заключение

Благодаря свойству бора поглощать нейтрон и распадаться на частицы, его можно использовать в лечении радиорезистентных опухолей, с которыми не могут справиться другие способы лечения. Проведенное моделирование показало, что нейтроны распространяются в теле хаотично, как броуновские частицы. Их поток резко ослабевает с увеличением расстояния от поверхности тела, поэтому такая терапия больше подходит для лечения не глубоко залегающих опухолей. Моделирование показало, что для выбранных геометрических параметров существует оптимальная энергия нейтронов, при которой происходит наибольшее число захватов. Малое количество бора в опухоли, достижимое на текущий момент, является основным недостатком терапии, так как хаотично движущиеся нейтроны создают заметную радиационную нагрузку на здоровые ткани.

Библиографический список

1. С. Ю. Таскаев. Бор-нейтронозахватная терапия (2021)
2. С.Ю. Таскаев. Разработка ускорительного источника эпитепловых нейтронов для бор-нейтронозахватной терапии (2019).
3. https://ru.vvikipedla.com/wiki/Neutron_capture_therapy_of_cancer
4. Като и другие. Эффективность борной нейтронно-захватной терапии при рецидивирующих злокачественных новообразованиях головы и шеи (2009).
5. Канкаанранта и другие. «Боронейтронозахватная терапия в лечении местного рецидива рака головы и шеи: окончательный анализ исследования фазы I / II» (2012).
6. Ван, Лин-Вэй; Лю, Йен-Ван Сюэ; (2018). «Клинические испытания лечения рецидивирующего рака головы и шеи с помощью борной нейтронно-захватной терапии с использованием реактора с открытым бассейном Tsing-Hua» (2018).
7. Касатова А.И., Каныгин В.В. Исследование биологической эффективности бор-нейтронозахватной терапии на клетках глиомы и меланомы человека (2020).
8. https://geant4.web.cern.ch/node/1
9. https://root.cern/
10. https://root.cern/root/vmc/

УДК 004.423.24

Греченко В.В.

ИСПОЛЬЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ В C#

USING REGULAR EXPRESSIONS IN C#

Филиал «Протвино» государственного университета «Дубна»
Секция «Информационные технологии»

Автор: Греченко Владимир Владимирович, студент 1-го курса направления «Информатика и вычислительная техника» филиала «Протвино» государственного университета «Дубна».

Научный руководитель: Кульман Татьяна Николаевна, кандидат технических наук, доцент кафедры информационных технологий филиала «Протвино» государственного университета «Дубна».

Author: Grechenko Vladimir Vladimirovich, 1st year student of the direction "Informatics and computer engineering" of the branch "Protvino" state University "Dubna".

Scientific adviser: Kulman Tatiana Nikolaevna, candidate of technical sciences, associate professor of the department information technology of the branch "Protvino" state University "Dubna".

Аннотация

Рассматривается реализация регулярных выражений в C# и их использование для выполнения различных операций над строками.

Abstract

Consider the realization of regular expressions in C# and their use for implementation different operations with strings.

Ключевые слова: регулярные выражения, класс Regex, метод, параметр, конструктор, шаблон, символ, метасимвол.

Keywords: regular expressions, Regex class, method, parameter, constructor, pattern, symbol, metacharacters.

Регулярные выражения – это часть небольшой технологической области, невероятно широко используемой в огромном диапазоне программ. Регулярные выражения представляют эффективный и гибкий метод по обработке больших текстов, позволяя в то же время существенно уменьшить объёмы программного кода по сравнению с использованием стандартных операций со строками [2].

Целью работы является написание программы для проверки вводимых пользователем данных на соответствие формату адреса электронной почты.

Актуальность: Наверное, каждый программист сталкивался с проблемой проверки входных данных на соответствие какому-то определённому формату или правилу записи. Многие прибегают к использованию методов и свойств классов string и char, однако не всегда этот способ является эффективным и удобным. Для тщательной обработки строк и больших блоков текстовой информации во многих языках программирования, в том числе и в C#, существует класс регулярных выражений – Regex, позволяющий значительно облегчить обработку строк.

Постановка задачи: написание программы с использованием класса регулярных выражений Regex для проверки пользовательских входных данных на соответствие формату адреса электронной почты. При вводе строки, несоответствующей формату e-mail адреса, программа должна вывести сообщение об ошибке.

Классы StringBuilder и String предоставляют достаточную функциональность для работы со строками. Однако платформа .NET предлагает ещё один мощный инструмент – регулярные выражения (regular expressions или regex) [5]. Регулярные выражения позволяют быстро анализировать большие объёмы текста в следующих целях:

- поиск определённых символов;
- проверка текста на соответствие предопределённому формату (например, для проверки номера мобильного телефона);
- извлечение, изменение, замена или удаление текстовых подстрок;

Для многих приложений, работающих со строками и анализирующих большие блоки текста, регулярные выражения являются незаменимым инструментом.

Главный компонент обработки текста с помощью регулярных выражений – это механизм регулярных выражений, представленный в .NET объектом System.Text.RegularExpressions.Regex [1]. Для обработки текста механизму необходимо предоставить как минимум два следующих элемента:

- шаблон регулярного выражения для определения текста;
- текст, который будет проанализирован на соответствие шаблону регулярного выражения;

Шаблон – это последовательность символов, определяющих регулярное выражение. Он может состоять из литералов, чисел, символов, операторов, конструкций, а также из метасимволов. Использование метасимволов является одним из главных преимуществ регулярных выражений. Они способны не только задавать команды, но и управлять последовательностями. Кроме строки шаблона конструктор класса Regex может принимать в качестве параметра другие необязательные параметры (например, параметр Multiline позволяет

использовать многострочный режим, где символы ^ и $ соответствуют началу и концу строки текста, а не началу и концу входной строки).

Пример простой программы с использованием класса Regex и метода Matches представлен на рисунке 1.

```
string s = "she sells sea shells at the sea shore, the shells " +
"she sells are the sea-shore shells, I'm sure";
            Regex regex = new Regex(@"(\w*)she(\w*)");
            MatchCollection matches = regex.Matches(s);
            if (matches.Count > 0)
            {
                Console.WriteLine("Словоформа -she- найдена в словах:");
                foreach (Match match in matches)
                    Console.WriteLine(match.Value);
            }
            else
            {
                Console.WriteLine("Совпадений не найдено!");
            }
            Console.ReadKey();
```

Рис. 1. Пример программы с использованием регулярного выражения

В данной программе мы находим в строке все слова, содержащие словоформу «she». В конструктор объекта Regex передаётся регулярное выражение “(\w*)she(\w*)” для поиска требуемых соответствий.

Метасимвол \w эквивалентен набору символов из диапазона [a-zA-z0-9_], а звёздочка указывает на неопределённое их количество – их может быть один, два, три, четыре и т.д. или может не быть вовсе.

Метод Matches класса Regex принимает строку, к которой надо применить регулярное выражение, и возвращает коллекцию найденных совпадений. Каждый элемент такой коллекции представляет объект Match, а его свойство Value возвращает найденное совпадение. На рисунке 2 представлен результат выполнения программы [4].

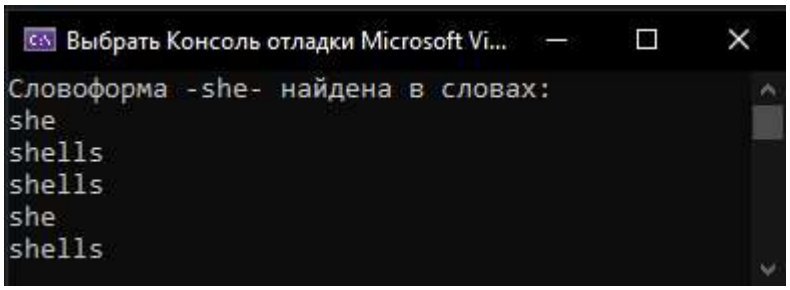

Рис. 2. Результат выполнения программы

Стоит отметить, что регулярные выражения не являются панацеей для каждой небольшой манипуляции со строкой. При простом синтаксическом анализе рекомендуется использовать методы, предоставляемые классом String и другими классами.

Перед написанием программы для проверки адреса электронной почты, был изучены формат записи e-mail адреса. Адрес должен состоять из двух частей, разделённых символом «@» (собака). Левая часть указывает имя почтового ящика, правая – доменное имя того сервера, на котором расположен почтовый ящик. E-mail адрес может состоять только из строчных и прописных букв латинского алфавита, цифр, специальных символов «-», «_» и точки «.». Пример правильной записи адреса электронной почты: grechenko.2002@mail.ru, где “grechenko.2002” – имя почтового ящика, задаваемое пользователем (логин), а “mail.ru” – доменное имя почтового ящика (имя сервера).

Далее, при помощи краткого справочника элементов языка регулярных выражений C#, представленного на официальном сайте компании Microsoft, был составлен шаблон регулярного выражения, состоящий из символов и метасимволов:

"^(([a-z0-9_-]+\.)*[a-z0-9_-]+@[a-z0-9-]+(\.[a-z0-9-]+)*\.[a-z]{2,6})$". Интерпретации шаблона показаны в следующей таблице [3]:

Символ, метасимвол, набор символов	**Интерпретация**
^	Соответствие должно начинаться в начале строки
[…]	Указывается допустимый диапазон символов
+	Квантификатор. Означает, что должен быть введён один или более символов из диапазона, указанного в […]
\.	\ - экранизирующий символ, указывает на то, что «.» является метасимволом, а не диапазоном любых символов
*	Квантификатор. Означает, что всё, что находится в (…) может повторяться 0 или более раз
{n, m}	Квантификатор. Указывается допустимое значение повторений символов из диапазона (как минимум n раз, но не более чем m раз)
$	Соответствие должно обнаруживаться в конце строки

Код программы представлен на рисунке 3.

```
string correct_email =
@"^(([a-z0-9_-]+\.)*[a-z0-9_-]+@[a-z0-9-]+(\.[a-z0-9-]+)*\.[a-z]{2,6})$";
            do
            {
                Console.Write("Введите адрес электронной почты: ");
                string email = Console.ReadLine();

            if (Regex.IsMatch(email, correct_email, RegexOptions.IgnoreCase))
            {
                 Console.WriteLine("Email подтверждён!");
            }
               else
                {
                    Console.WriteLine("Некорректный email! " +
                        "Проверьте правильность введённого адреса.");
                }
                Console.WriteLine("Для выхода нажмите клавишу ESC");
            }
            while (Console.ReadKey(true).Key != ConsoleKey.Escape);
```

Рис. 3. Код программы для проверки на соответствие формату адреса электронной почты

Метод IsMatch() указывает, обнаружено ли в указанной строке соответствие регулярному выражению, заданному в конструкторе Regex, а параметр IgnoreCase игнорирует регистр символов во входной строке.

Для проверки работоспособности программы были придуманы пять различных адресов электронной почты:

1. Ivanov.ivan2005@yandex.ru (верный формат)
2. petrovpetr@mail (неверный формат)
3. lunin..roman@gmail.com (неверный формат)
4. Pe4enkin-Mark@rambler.ru (верный формат)
5. Utkina Tanya@yahoo.com (неверный формат)

На рисунке 4 представлены результаты выполнения программы при вводе вышеперечисленных e-mail адресов.

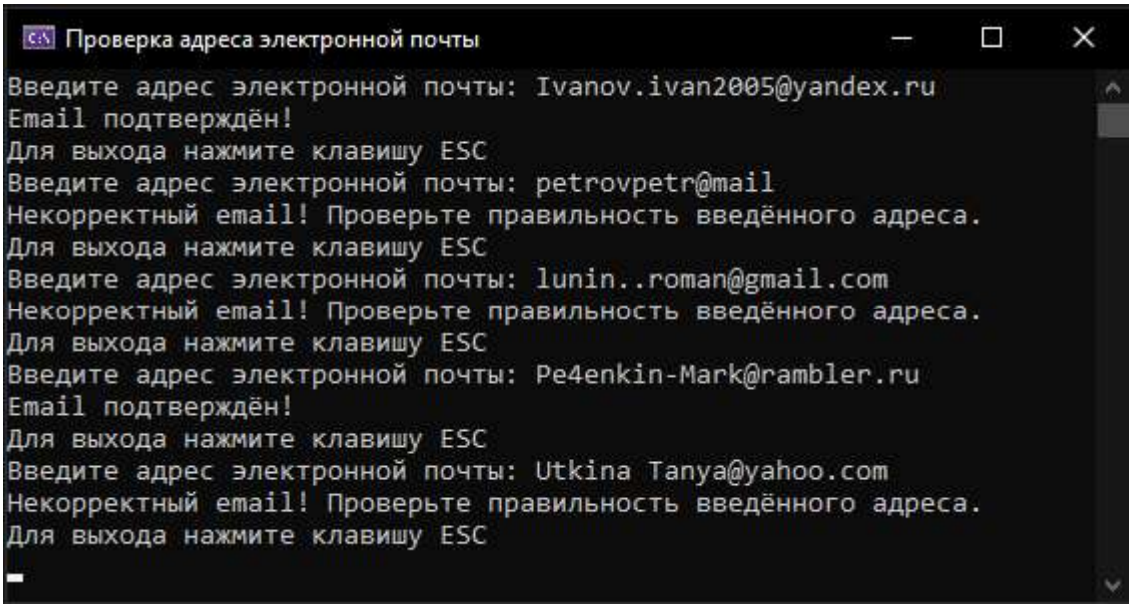

Рис. 4. Консоль приложения

Результатом проделанной работы является:

1) Знакомство с синтаксисом и реализацией регулярных выражений в C#.
2) Работающая программа для проверки входных данных на соответствие формату адреса электронной почты, написанная с использованием регулярных выражений.

Таким образом, регулярные выражения предоставляют эффективный и гибкий способ обработки строк. Коллекция методов и параметров, входящих в класс регулярных выражений Regex, позволяют быстро анализировать большие объёмы текста, экономя время программиста и существенно уменьшая программный код.

Библиографический список

1. Регулярные выражения в .NET - https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/regular-expressions
2. Регулярные выражения в C# - https://professorweb.ru/my/csharp/charp_theory/level4/4_10.php
3. Элементы языка регулярных выражений – краткий справочник - https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/regular-expression-language-quick-reference
4. Объектная модель регулярных выражений - https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/the-regular-expression-object-model
5. Регулярные выражения - https://metanit.com/sharp/tutorial/7.4.php