**EXPERIMENT NO: 07**

**Title**: Develop program to use Regex. Matches method and Regular Expression pattern matching.

**Aim:** Study of Regular expression pattern matching.

**Theory:**

**Regular Expression:**

In C#, Regular Expression is a pattern which is used to parse and check whether the given input text is matching with the given pattern or not. In C#, Regular Expressions are generally termed as C# Regex. The .Net Framework provides a regular expression engine that allows the pattern matching. Patterns may consist of any character literals, operators or constructors.

C# provides a class termed as Regex which can be found in System.Text.RegularExpression namespace. This class will perform two things:

Parsing the inputting text for the regular expression pattern.

Identify the regular expression pattern in the given text.

**Example 1:** Below example demonstrate the use of regex in Mobile Number Verification. Suppose you are making a form where you need to verify the user-entered mobile number then you can use regex.

// C# program to validate the Mobile

// Number using Regular Expressions

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main(string[] args)

{

// Input strings to Match

// valid mobile number

string[] str = {"9925612824",

"8238783138", "02812451830"};

foreach(string s in str)

{

Console.WriteLine("{0} {1} a valid mobile number.", s,

isValidMobileNumber(s) ? "is" : "is not");

}

Console.ReadKey();

}

// method containing the regex

public static bool isValidMobileNumber(string inputMobileNumber)

{

string strRegex = @"(^[0-9]{10}$)|(^\+[0-9]{2}\s+[0-9]

{2}[0-9]{8}$)|(^[0-9]{3}-[0-9]{4}-[0-9]{4}$)";

// Class Regex Represents an

// immutable regular expression.

// Format Pattern

// xxxxxxxxxx ^[0 - 9]{ 10}$

// +xx xx xxxxxxxx ^\+[0 - 9]{ 2}\s +[0 - 9]{ 2}\s +[0 - 9]{ 8}$

// xxx - xxxx - xxxx ^[0 - 9]{ 3} -[0 - 9]{ 4}-[0 - 9]{ 4}$

Regex re = new Regex(strRegex);

// The IsMatch method is used to validate

// a string or to ensure that a string

// conforms to a particular pattern.

if (re.IsMatch(inputMobileNumber))

return (true);

else

return (false);

}

}

**Output:**

9925612824 is a valid mobile number.

8238783138 is a valid mobile number.

02812451830 is not a valid mobile number.

**Example 2:** Below example demonstrate the use of regex in Email ID Verification. Suppose you are making a form where you need to verify the user-entered email id then you can use regex.

// C# program to validate the Email

// ID using Regular Expressions

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main(string[] args)

{

// Input strings for Match

// valid E-mail address.

string[] str = {"parth@gmail.com",

"parthmaniyargmail.com",

"@gmail.com"};

foreach(string s in str)

{

Console.WriteLine("{0} {1} a valid E-mail address.", s,

isValidEmail(s) ? "is" : "is not");

}

}

// Method to check the Email ID

public static bool isValidEmail(string inputEmail)

{

// This Pattern is use to verify the email

string strRegex = @"\A(?:[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?)\Z";

Regex re = new Regex(strRegex, RegexOptions.IgnoreCase);

if (re.IsMatch(inputEmail))

return (true);

else

return (false);

}

}

**Output:**

parth@gmail.com is a valid E-mail address.

parthmaniyargmail.com is not a valid E-mail address.

@gmail.com is not a valid E-mail address.

**Regex Syntax**

There are many basic syntaxes like Quantifiers, Special Characters, Character Classes, Grouping & Alternatives are used for regular expressions.

**Quantifiers:**

| Sub-expression(Greedy) | Sub-expression(Lazy) | Matches |
| --- | --- | --- |
| \* | \*? | Used to match the preceding character zero or more times. |
| + | +? | Used to match the preceding character one or more times. |
| ? | ?? | Used to match the preceding character zero or one time. |
| {n} | {n}? | Used to match the preceding character exactly n times. |
| {n, } | {n, }? | Used to match the preceding character at least n times. |
| {n, m} | {n, m}? | Used to match the preceding character from n to m times. |

**Example 1:**

// C# program to demonstrate

// the \* Quantifier

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

public static void Main(string[] args)

{

// This will return any

// pattern b, ab, aab, ...

Regex regex = new Regex(@"a\*b");

Match match = regex.Match("aaaabcd");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: aaaab

**Example 2:**

// C# program to demonstrate

// the + Quantifier

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

public static void Main()

{

// this will return any pattern

// like ab, aab, aaab, ....

Regex regex = new Regex(@"a+b");

Match match = regex.Match("aaabcd");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: aaab

**Example 3:**

// C# program to demonstrate

// the ? Quantifier

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This return any pattern like b, ab

Regex regex = new Regex(@"a?b");

Match match = regex.Match("aaaabcd");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: ab

**Special Characters**

| **Sub-expression** | **Matches** |
| --- | --- |
| ^ | Word after this element matches at the beginning of the string or line. |
| $ | Word before this element matches at the end of the line or string. |
| .(Dot) | Matches any character only once expect \n(new line). |
| \d | It is use to match the digit character. |
| \D | It is use to match the non-digit character. |
| \w | It is use to match any alphanumeric and underscore character. |
| \W | It is use to match the any non-word character. |
| \s | It is use to match the white-space characters. |
| \S | It is use to match the non white-space characters. |
| \n | It is use to match a newline character. |

**Example 1:**

// C# program to demonstrate

// the ^ Special Character

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This will return if shyam exist

// at the beginning of the line

Regex regex = new Regex(@"^Shyam");

Match match = regex.Match("Shyam is my pet name");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: Shyam

**Example 2:**

// C# program to demonstrate

// the $ Special Character

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

public static void Main()

{

// This return parth if it

// exist at the end of the line

Regex regex = new Regex(@"Parth$");

Match match = regex.Match("My name is Parth");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: Parth

**Example 3:**

// C# program to demonstrate

// the .(Dot) Special Character

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This will return any word which

// contains only one letter between

// s and t

Regex regex = new Regex(@"s..t");

Match match = regex.Match("This is my seat");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: seat

**Example 4:**

// C# program to demonstrate

// the \d Special Character

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This will the return

// the one digit character

Regex regex = new Regex(@"\d");

Match match = regex.Match("I am 19 years old");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: 1

**Character Classes**

| **Sub-expression** | **Matches** |
| --- | --- |
| [] | It is used to match the range of character |
| [a-z] | It is used to match any character in the range of a-z. |
| [^a-z] | It is used to match any character not in the range of a-z. |
| \ | It is used to match Escaped special character. |

**Example 1:**

// C# program to demonstrate

// the [] character class

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This will return one character either

// a or b or c which will come first

Regex regex = new Regex(@"[abc]");

Match match = regex.Match("abcdef");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: a

**Example 2:**

// C# program to demonstrate

// the [a-z] character class

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This will return any character

// between x and z inclusive

Regex regex = new Regex(@"[x-z]");

Match match = regex.Match("xmax");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: x

**Example 3:**

// C# program to demonstrate

// the [^a-z] character class

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This will return other x,

// y and z character

Regex regex = new Regex(@"[^x-z]");

Match match = regex.Match("xmax");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: m

**Grouping and Alternatives**

| **Sub-expression** | **Matches** |
| --- | --- |
| () | It is used for group expression |
| (a|b) | | Operator is used for alternative either a or b. |
| (?(exp) yes|no) | If expression is matched it gives yes otherwise it gives no. |

**Example 1:**

// C# program to demonstrate

// the grouping in regex

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This will return pattern

// will cd, cdcd, cdcdcd, ...

Regex regex = new Regex(@"(cd)+");

Match match = regex.Match("cdcdde");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: cdcd

**Example 2:**

// C# program to demonstrate

// the grouping in regex

using System;

using System.Text.RegularExpressions;

class GFG {

// Main Method

static void Main()

{

// This will either d or e

// which ever comes first

Regex regex = new Regex(@"d|e");

Match match = regex.Match("edge");

if (match.Success)

{

Console.WriteLine("Match Value: " + match.Value);

}

}

}

**Output:**

Match Value: e

**Problem Statement:**

Write Program to validate following data:

EmailID, Mobile No. and Name

**Conclusion:**

Here we have studied regular expression pattern matching.

**Sample Questions:**

1) What is Regular Expression?

using System.Text.RegularExpressions;

static void Main(string[] args)

{

// Regular expressions for mobile number, email address, and name

string mobileRegex = @"^\d{10}$";

string emailRegex = @"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$";

string nameRegex = @"^[a-zA-Z\s]+$";

bool exitRequested = false;

while (!exitRequested)

{

Console.WriteLine("Choose what you want to validate:");

Console.WriteLine("1. Name");

Console.WriteLine("2. Email");

Console.WriteLine("3. Mobile Number");

Console.WriteLine("4. Exit");

int choice;

if (!int.TryParse(Console.ReadLine(), out choice) || choice < 1 || choice > 4)

{

Console.WriteLine("Invalid choice. Please choose a number between 1 and 4.");

continue;

}

string userInput;

switch (choice)

{

case 1:

Console.WriteLine("Enter name:");

userInput = Console.ReadLine();

Console.WriteLine("Name - Valid: " + Regex.IsMatch(userInput, nameRegex));

break;

case 2:

Console.WriteLine("Enter email address:");

userInput = Console.ReadLine();

Console.WriteLine("Email - Valid: " + Regex.IsMatch(userInput, emailRegex));

break;

case 3:

Console.WriteLine("Enter mobile number:");

userInput = Console.ReadLine();

Console.WriteLine("Mobile Number - Valid: " + Regex.IsMatch(userInput, mobileRegex));

break;

case 4:

exitRequested = true;

break;

}

}

Console.WriteLine("Exiting program...");

}