Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

LABORATORY WORK NO.4

# Topic: Regular expressions

Elaborated:

st. gr.

FAF-223 Ciornii Alexandr

Verified:

asist. univ.

Dumitru Cretu

Chişinău - 2024

# Objectives:

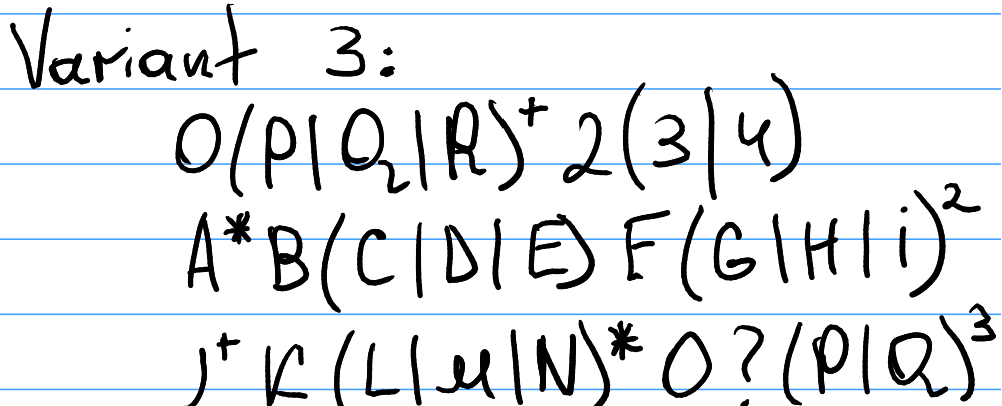
1. Write and cover what regular expressions are, what they are used for;
2. Below you will find 3 complex regular expressions per each variant. Take a variant depending on your number in the list of students and do the following:

a. Write a code that will generate valid combinations of symbols conform given regular expressions (examples will be shown).

b. In case you have an example, where symbol may be written undefined number of times, take a limit of 5 times (to evade generation of extremely long combinations);

c. Bonus point: write a function that will show sequence of processing regular expression (like, what you do first, second and so on)

Write a good report covering all performed actions and faced difficulties.



Implementation:

Firstly I used Lexer from previous Lab. (I changed symbol types according to regex, elsewhere it’s the same. Here’s enum with all accesible tokens:

public enum TokenType  
{  
 Symbol,  
 LBracket,  
 Rbracket,  
 Or,  
 Power,  
 QuestionMark,  
 Plus,  
 Star,  
 EOF // End of File  
}

Also I have RegExGenerator class with 1 method GenerateStrings that basically does what it should

In constructor I create everything needed, but most important is to make List of all tokens from RegEx

public RegExGenerator(List<Token> tokens)  
{  
 \_isInBrackets = false;  
 \_isPower = false;  
 \_inBracketsTokens = new List<string>();  
 \_tokens = tokens;  
 \_rand = new Random();  
}

In GenerateString method I initialize StringBuilder (it’s convinient way to make string that is frequently changed)and start foreach loop in tokens

StringBuilder sb = new StringBuilder();  
 var curStr = "";  
 foreach (Token token in \_tokens)

Than for each tokentype I do necessary actions, for example for symbol token I have:

case (TokenType.Symbol, \_):  
 if(\_isInBrackets)  
 \_inBracketsTokens.Add(token.Value);  
 else if (\_isPower)  
 {  
 var count = token.Value.ToCharArray()[0] - '0' - 1 ;  
 for (int i = 0; i < count; i++)  
 {  
 sb.Append(curStr);  
 Console.WriteLine("current string = " + sb.ToString());  
  
 }  
  
 \_isPower = false;  
 }  
 else  
 {  
 curStr = token.Value;  
 sb.Append(curStr);  
 Console.WriteLine("current string = " + sb.ToString());  
 }  
  
 break;

base case is if it’s just simple symbol that i just append to stringbuilder, if it’s symbol to the power it’s added exactly that number of symbols, and last case if the symbol is in the brackets it’s added to another list from which later I choose random symbol

Plus and Star symbols are almost the same, I take currentSymbol and apply it random number of times from 0 (if star) or from 1 (if plus) and to 5 toavoid infinite strings, here’s code for plus:

case (TokenType.Plus, \_):  
 rand = new Random().Next(0, 5);  
 for (int i = 0; i < rand; i++)  
 {  
 sb.Append(curStr);  
 Console.WriteLine("current string = " + sb.ToString());  
  
 }  
 break;

Implentation of brackets is quite stright forward, just take all symbols and choose randomly

at the end GenerateString method return resuilting string so it’s easy to write in Console  
note that all

Console.WriteLine("current string = " + sb.ToString());

line of code are needed for showing steb-by-step showing how strings are built

Conclusion

In this lab, we explored regular expressions, powerful tools for pattern matching and text manipulation. Through a C# program, we generated valid symbol combinations based on complex regular expressions. Here's a concise overview of our process:

1. Understanding Regular Expressions: We grasped the importance of regular expressions in text processing, serving various purposes like pattern matching and data validation.
2. Parser Implementation: We built a lexer in C# to tokenize the provided regular expressions, identifying symbols, brackets, and operators for further processing.
3. Generating Valid Combinations: Utilizing the tokenized expressions, we produced valid symbol combinations adhering to specified patterns. We carefully managed repetition operators to avoid excessively long outputs.
4. Challenges Faced: We encountered challenges in handling repetition and selecting symbols randomly from brackets while ensuring equal chances for each symbol. Proper tokenization was crucial for accurate parsing.

Through this lab, we gained practical experience in working with regular expressions and implementing parsers for pattern matching tasks. Our successful generation of valid symbol combinations demonstrates proficiency in utilizing regular expressions to solve real-world software engineering challenges.