

COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS

LAB MID



Submitted By

Muhammad Haseeb (SP21-BCS-020)

Submitted To

Mr. Bilal Bukhari

Course Title

Compiler Construction

Date

5-4-2024

Question No 1:

Briefly describe the regex library of C#

Overview of Regular Expressions Library in C#

Introduction:

Regular expressions (regex) are a powerful tool for pattern matching and text manipulation. In C#, the **System.Text.RegularExpressions** namespace provides a comprehensive library for working with regular expressions. This document presents an overview of its features and functionalities.

1. Pattern Matching:

Regular expressions allow defining patterns to match text within strings.

2. Regex Class:

The **Regex** class is central to the library, offering methods for compiling regex patterns into **Regex** objects and performing various operations like matching, replacing, and splitting strings.

3. Match and MatchCollection:

When a regex is applied to a string, it returns a **Match** object representing the first match found. Multiple matches can be retrieved using the **Matches** method, which returns a **MatchCollection**.

4. Groups and Capturing Groups:

Regex matches are organized into groups, and capturing groups allow extraction of specific parts of the matched text.

5. Options:

The library supports options that affect regex behavior, such as case sensitivity, single-line mode, and explicit capture.

6. Replacement:

Methods are available for replacing matched substrings with new strings based on regex patterns.

7. Anchors and Quantifiers:

Use anchors (^, \$) to specify the start and end of a line or string, and quantifiers (*, +, ?, {}) to specify repetition of characters or groups.

8. Character Classes and Escapes:

Support for character classes ([...]) for specifying sets of characters, as well as escape sequences (\d, \w, \s) for common character types.

9. Lookahead and Lookbehind:

Advanced features include lookahead ((?=...), (!?...)) and lookbehind ((?<=...), (?<!...)) for more complex matching conditions.

Conclusion:

The **System.Text.RegularExpressions** namespace in C# provides comprehensive support for working with regular expressions, making it invaluable for tasks like text parsing, validation, and transformation.

Question No 2:

```
#include <iostream>
#include <stdlib.h>
using namespace std;

// Global variables
int count = 0; // Counter to keep track of the position in the expression string
string expr;   // Input expression

// Function prototypes
void S();      // Start symbol
void X();      // Non-terminal X
void Xp();     // X prime
void Y();      // Non-terminal Y
void Yp();     // Y prime
void Z();      // Non-terminal Z

int main() {
    // Read the expression from the user
    cin >> expr;
    // Append "$" to the end of the expression to indicate the end
    expr += "$";

    // Call the start symbol parsing function
    S();
}
```

```

    // Check if the entire expression has been parsed
    if (expr.length() == count) {
        cout << "Accepted" << endl;
    } else {
        cout << "Rejected" << endl;
    }

    cin.get(); // Wait for a character input
}

// Parse S
void S() {
    cout << "S->X$" << endl;
    X();
    if (expr[count] == '$') {
        count++;
    } else {
        cout << "Rejected" << endl;
        exit(0);
    }
}

// Parse X
void X() {
    cout << "X->YX'" << endl;
    Y();
    Xp();
}

// Parse X prime
void Xp() {
    if (expr[count] == '%') {
        count++;
        cout << "X'->%YX'" << endl;
        Y();
        Xp();
    } else {
        cout << "X'-> $\epsilon$ " << endl;
    }
}

// Parse Y
void Y() {
    cout << "Y->ZY'" << endl;

```

```

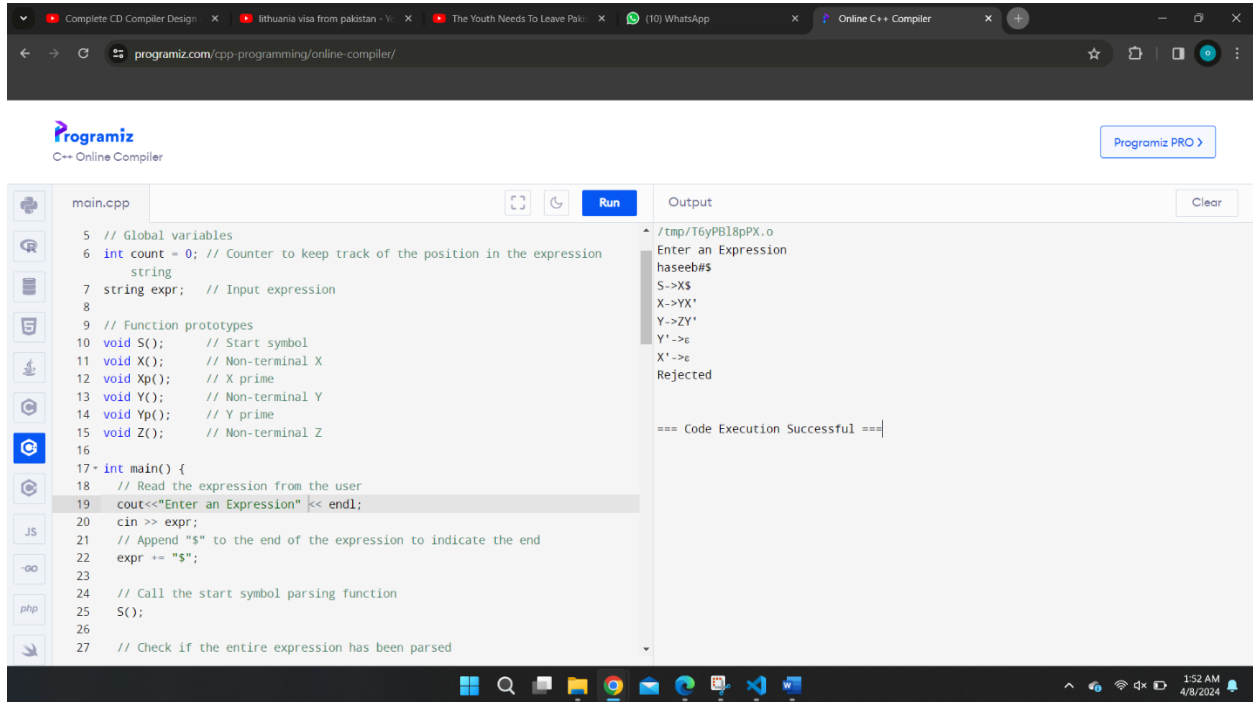
    Z();
    Yp();
}

// Parse Y prime
void Yp() {
    if (expr[count] == '&') {
        count++;
        cout << "Y'->&ZY'" << endl;
        Z();
        Yp();
    } else {
        cout << "Y'-> $\epsilon$ " << endl;
    }
}

// Parse Z
void Z() {
    if (expr[count] == 'k') {
        count++;
        cout << "Z->kXk" << endl;
        X();
        if (expr[count] == 'k') {
            count++;
        } else {
            cout << "Rejected" << endl;
            exit(0);
        }
    } else if (expr[count] == 'g') {
        count++;
        cout << "Z->g" << endl;
        return;
    }
}
}

```

OUTPUT:



The screenshot shows the Programiz C++ Online Compiler interface. The code editor on the left contains a C++ program for parsing expressions. The code defines global variables, function prototypes for S(), X(), Xp(), Y(), Yp(), and Z(), and a main function that reads an expression from the user and calls the parsing function S(). The output window on the right shows the execution results, including the prompt "Enter an Expression", the input "haseeb\$", and the final output "Rejected". The status bar at the bottom indicates "Code Execution Successful".

```
5 // Global variables
6 int count = 0; // Counter to keep track of the position in the expression
7 string expr; // Input expression
8
9 // Function prototypes
10 void S(); // Start symbol
11 void X(); // Non-terminal X
12 void Xp(); // X prime
13 void Y(); // Non-terminal Y
14 void Yp(); // Y prime
15 void Z(); // Non-terminal Z
16
17 int main() {
18 // Read the expression from the user
19 cout<<"Enter an Expression" << endl;
20 cin >> expr;
21 // Append "$" to the end of the expression to indicate the end
22 expr += "$";
23
24 // Call the start symbol parsing function
25 S();
26
27 // Check if the entire expression has been parsed
```

Output:

```
/tmp/T6yPB18pPX.o
Enter an Expression
haseeb$
S->XS
X->YX'
Y->ZY'
Y'->ε
X'->ε
Rejected

=== Code Execution Successful ===
```

Question No 3:

```
using System;
using System.Linq;
using System.Text;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Welcome to the Password Generator!");
        Console.WriteLine("Please enter your first name:");
        string firstName = Console.ReadLine();

        Console.WriteLine("Please enter your last name:");
        string lastName = Console.ReadLine();

        Console.WriteLine("Please enter your registration numbers:");
        string registrationNumbers = Console.ReadLine();
```

```

        string password = GeneratePassword(firstName, lastName,
registrationNumbers);
        Console.WriteLine("Generated Password: " + password);
    }

    static string GeneratePassword(string firstName, string lastName, string
registrationNumbers)
    {
        StringBuilder password = new StringBuilder();

        password.Append(char.ToUpper(firstName[0]));
        password.Append(char.ToUpper(lastName[0]));

        for (int i = 0; i < firstName.Length; i++)
        {
            if (i % 2 == 0)
                password.Append(firstName[i]);
        }

        for (int i = 0; i < lastName.Length; i++)
        {
            if (i % 2 != 0)
                password.Append(lastName[i]);
        }

        password.Append((char)('A' + new Random().Next(0, 26)));

        var selectedNumbers = registrationNumbers.Where(char.IsDigit).OrderBy(n
=> Guid.NewGuid()).Take(2);
        foreach (var number in selectedNumbers)
        {
            password.Append(number);
        }

        for (int i = 0; i < 2; i++)
        {
            password.Append(new Random().Next(0, 10));
        }

        string specialChars = "!@#$%^&*()_+=[{}|;:,<>?";
        for (int i = 0; i < 2; i++)
        {
            password.Append(specialChars[new Random().Next(0,
specialChars.Length)]);
        }
    }

```

```

        string shuffledPassword = new string(password.ToString().OrderBy(x =>
Guid.NewGuid()).ToArray());

        if (shuffledPassword.Length > 16)
        {
            shuffledPassword = shuffledPassword.Substring(0, 16);
        }

        return shuffledPassword;
    }
}

```

OUTPUT:

The screenshot displays the Programiz C++ Online Compiler interface. The left pane shows the source code for `main.cpp`, which includes global variables, function prototypes, and a `main` function that reads an expression and calls a parsing function `S()`. The right pane shows the output of the program, which includes the prompt "Enter an Expression", the input "haseeb#\$", and the resulting parse tree "S->XS", "X->YX'", "Y->ZY'", "Y'->ε", "X'->ε", and "Rejected". The output also confirms "Code Execution Successful".

```

main.cpp
5 // Global variables
6 int count = 0; // Counter to keep track of the position in the expression
  string
7 string expr; // Input expression
8
9 // Function prototypes
10 void S(); // Start symbol
11 void X(); // Non-terminal X
12 void Xp(); // X prime
13 void Y(); // Non-terminal Y
14 void Yp(); // Y prime
15 void Z(); // Non-terminal Z
16
17 int main() {
18     // Read the expression from the user
19     cout<<"Enter an Expression" << endl;
20     cin >> expr;
21     // Append "$" to the end of the expression to indicate the end
22     expr += "$";
23
24     // Call the start symbol parsing function
25     S();
26
27     // Check if the entire expression has been parsed

```

Output

```

/tmp/T6yPB18pPX.o
Enter an Expression
haseeb#$
S->XS
X->YX'
Y->ZY'
Y'->ε
X'->ε
Rejected

=== Code Execution Successful ===

```