NAME: MUHAMMAD SHARJEEL IQBAL

CLASS: BCS-7B

REG NO: FA20-BCS-022

SUBJECT: COMPILER CONSTRUCTION

TEACHER: BILAL SAAB

**QUESTION 1**

**Briefly describe the regex library of C#**

In C#, the .NET Framework provides a powerful regular expression library through the System.Text.RegularExpressions namespace. This library allows you to work with regular expressions for pattern matching and manipulation.

Here's a brief overview of how to use it:

**Regex Class:** The Regex class is at the core of this library. It allows you to create and work with regular expressions. Here's an example of how to create a regex pattern and use it to match a string:

using System;

using System.Text.RegularExpressions;

class Program

{

static void Main()

{

string pattern = @"\d+"; // Matches one or more digits

string input = "12345 abc 6789";

MatchCollection matches = Regex.Matches(input, pattern);

foreach (Match match in matches)

{

Console.WriteLine(match.Value);

}

}

}

**Regex Options**: You can use options like IgnoreCase to make your regular expressions case-insensitive, and Multiline to match across multiple lines.

**Match and MatchCollection:** The Match and MatchCollection classes are used to capture matched portions of a string. In the example above, MatchCollection contains all the matched substrings.

**Replacement:** You can use regular expressions for text replacement as well. Here's an example of replacing all occurrences of a word:

using System;

using System.Text.RegularExpressions;

class Program

{

static void Main()

{

string pattern = @"\bapple\b"; // Matches the word "apple" as a whole word

string input = "I like apple and applesauce. Apple is delicious.";

string replacement = "banana";

string result = Regex.Replace(input, pattern, replacement);

Console.WriteLine(result);

}

}

The C# regex library is extensive and provides many features for pattern matching and manipulation. These examples should give you a basic idea of how to get started with it.

**QUESTION 2**

**Make recursive descent or LL1 parser for the following grammer:**

**S -> E$**

**E -> T E'**

**E' -> + T E' | ε**

**T -> F T'**

**T' -> \* F T' | ε**

**F -> ( E ) | id**

ANSWER

To make this parser LL(1), let's start by creating a parsing table:

| | id | + | \* | ( | ) | $ |

|----|-------|-------|-------|-------|-------|-------|

| S | S->E$ | | | S->E$ | | |

| E | E->TE' | | | E->TE' | | |

| E' | | E'->+TE' | | | E'->ε | E'->ε |

| T | T->FT' | | | T->FT' | | |

| T' | | T'->ε | T'->\*FT' | | T'->ε | T'->ε |

| F | F->(E) | | | F->id | | |

The table indicates the production rules to apply for each non-terminal symbol (rows) when reading a specific terminal symbol (columns). Now, let's create the recursive descent parser:

**CODE:**

using System;

using System.Collections.Generic;

class RecursiveDescentParser

{

static List<string> tokens;

static int currentToken;

static void Main(string[] args)

{

tokens = new List<string> { "id", "+", "id", "\*", "id", "$" };

currentToken = 0;

ParseS();

if (currentToken == tokens.Count - 1 && tokens[currentToken] == "$")

{

Console.WriteLine("Parsing successful.");

}

else

{

Console.WriteLine("Parsing failed.");

}

}

static void ParseS()

{

ParseE();

if (tokens[currentToken] == "$")

{

// Parsing successful

}

else

{

Console.WriteLine("Parsing failed.");

}

}

static void ParseE()

{

ParseT();

ParseEPrime();

}

static void ParseEPrime()

{

if (tokens[currentToken] == "+")

{

currentToken++;

ParseT();

ParseEPrime();

}

// Epsilon production

}

static void ParseT()

{

ParseF();

ParseTPrime();

}

static void ParseTPrime()

{

if (tokens[currentToken] == "\*")

{

currentToken++;

ParseF();

ParseTPrime();

}

// Epsilon production

}

static void ParseF()

{

if (tokens[currentToken] == "id")

{

currentToken++;

}

else if (tokens[currentToken] == "(")

{

currentToken++;

ParseE();

if (tokens[currentToken] == ")")

{

currentToken++;

}

else

{

Console.WriteLine("Parsing failed.");

}

}

else

{

Console.WriteLine("Parsing failed.");

}

}

}

It parses a list of tokens and reports whether parsing was successful or failed.

**QUESTION 3**

Make a Password generator according the following rules:

(a) At least one uppercase alphabet

(b) At least 4 numbers (2 numbers must be my registration number that is 22)

(c) At least 2 special characters

(d) Must contain initials of first and last name

(e) maximum length of 16

**CODE:**

using System;

using System.Text;

using System.Security.Cryptography;

class PasswordGenerator

{

public static string GeneratePassword(string firstName, string lastName, string registrationNumber)

{

// Ensure the maximum length is 16 characters

int maxLength = 16;

// Generate a random uppercase alphabet

Random random = new Random();

char uppercaseLetter = (char)(random.Next(26) + 'A');

// Generate two random numbers

string randomNumber1 = "22";

string randomNumber2 = random.Next(10).ToString();

// Generate two random special characters

string specialCharacters = "!@#$%^&\*()-\_=+[]{}|;:'\",.<>?/";

StringBuilder password = new StringBuilder();

password.Append(firstName[0]); // Initial of the first name

password.Append(lastName[0]); // Initial of the last name

password.Append(uppercaseLetter);

password.Append(randomNumber1);

password.Append(randomNumber2);

// Calculate how many more characters are needed

int remainingLength = maxLength - password.Length;

// Fill the remaining space with random characters

for (int i = 0; i < remainingLength; i++)

{

int randomIndex = random.Next(specialCharacters.Length);

password.Append(specialCharacters[randomIndex]);

}

// Shuffle the password characters randomly

char[] passwordArray = password.ToString().ToCharArray();

for (int i = 0; i < passwordArray.Length; i++)

{

int randomIndex = random.Next(i, passwordArray.Length);

char temp = passwordArray[i];

passwordArray[i] = passwordArray[randomIndex];

passwordArray[randomIndex] = temp;

}

return new string(passwordArray);

}

static void Main(string[] args)

{

string firstName = "Sharjeel";

string lastName = "Iqbal";

string registrationNumber = "22";

string password = GeneratePassword(firstName, lastName, registrationNumber);

Console.WriteLine("Generated Password: " + password);

}

}