**Assignment #03**

**COMSATS UNIVERSITY ISLAMABAD**

**Logo, company name

Description automatically generated**

**ATTOCK CAMPUS**

**Submitted By**

Faraidon

**Registration No**

SP21-BCS-005

**Course Title**

Compiler Construction

**Date**

05-04-2024

**Regex**

In C#, the regex library is provided by the **System.Text.RegularExpressions** namespace. It enables developers to work with regular expressions for pattern matching and manipulation of strings. Some key features of the regex library in C# include:

1. **Regex Class**: The **Regex** class is the primary class used for working with regular expressions. It provides methods for matching patterns within strings, replacing patterns, and splitting strings based on patterns.
2. **Pattern Syntax**: C# supports a wide range of regular expression syntax, allowing for complex pattern matching. This includes support for character classes, quantifiers, anchors, groups, and more.
3. **Match Object**: When a regex pattern is matched against a string, the **Match** object is returned, providing information about the match such as the matched value, index, length, and any captured groups.
4. **Match Collections**: Multiple matches within a string can be obtained as a collection of **Match** objects, allowing for iterative processing of matches.
5. **Replacement Patterns**: Regex provides support for replacing matched patterns within strings using replacement patterns. This allows for dynamic string manipulation based on regex matches.
6. **Options**: Various options can be specified when working with regular expressions, such as case sensitivity, single-line mode, and ignore whitespace mode.

Here's another way to look at it:

1. **Pattern Matcher:** Just like how you might use Ctrl + F to find text in a document, regex allows you to do this but with much more complex patterns. You can search for things like email addresses, phone numbers, or even specific words with certain criteria.
2. **Pattern Creator**: Imagine you're trying to find all the email addresses in a document. With regex, you can create a pattern that matches the structure of an email address. This pattern can be as simple or as complex as you need it to be.
3. **Pattern Modifier**: Sometimes you might want to replace certain parts of text with something else. Regex lets you not only find patterns but also modify them. For example, you could change all instances of "color" to "colour" in a document.
4. **Pattern Analyzer**: It's not just about finding and replacing text; regex can also help you understand patterns in your data. You can extract specific parts of a text that match certain patterns, like pulling out all the dates or numbers from a document.
5. **Pattern Validator**: Beyond just finding and modifying text, regex can also help ensure that text matches certain criteria. For instance, you can use it to validate whether a user's input follows a specific format, like ensuring an entered phone number is in the correct format.

**Question number 2 code:**

#include <iostream>

#include <string>

using namespace std;

class Parser {

private:

    string input\_string;

    size\_t index;

    char current\_token;

public:

    Parser(const string& input) : input\_string(input), index(0), current\_token(input[0]) {}

    bool match(char expected\_token) {

        if (current\_token == expected\_token) {

            index++;

            if (index < input\_string.length()) {

                current\_token = input\_string[index];

            }

            return true;

        }

        return false;

    }

    bool parse\_S() {

        if (parse\_X() && match('$')) {

            return true;

        }

        return false;

    }

    bool parse\_X() {

        if (parse\_Y() && parse\_X\_prime()) {

            return true;

        }

        return false;

    }

    bool parse\_X\_prime() {

        if (match('%')) {

            if (parse\_Y() && parse\_X\_prime()) {

                return true;

            }  }

        return true;

    }

    bool parse\_Y() {

        if (parse\_Z() && parse\_Y\_prime()) {

            return true;

        }

        return false;

    }

    bool parse\_Y\_prime() {

        if (match('&')) {

            if (parse\_Z() && parse\_Y\_prime()) {

                return true;

            }

        }

        return true;

    }

    bool parse\_Z() {

        if (match('k') && parse\_X() && match('k')) {

            return true;

        }

        else if (match('g')) {

            return true;

        }

        return false;

    }

    bool parse() {

        return parse\_S();

    }

};

int main() {

    string input;

    cout << "Enter a string to parse: ";

    cin >> input;

    Parser parser(input);

    if (parser.parse()) {

        cout << "String is in the language" << endl;

    }

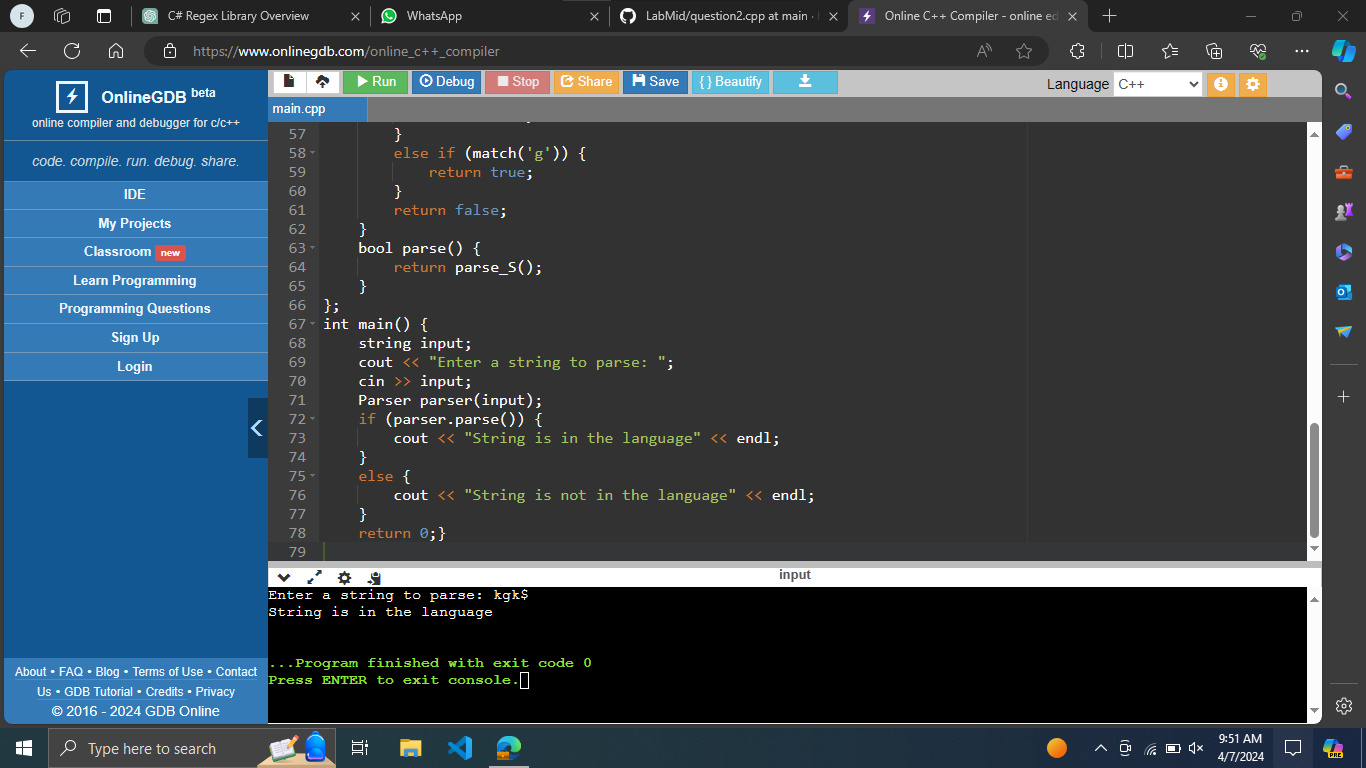
    else {

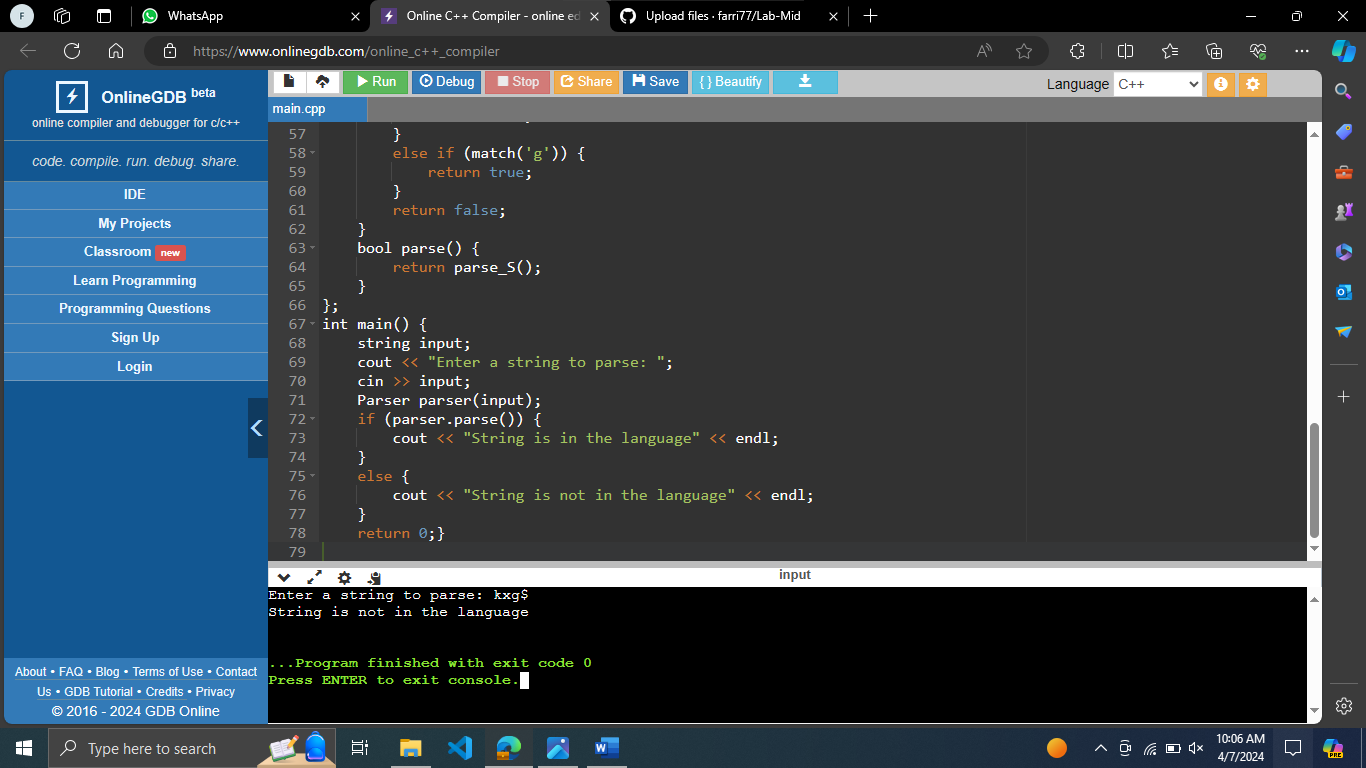
        cout << "String is not in the language" << endl;

    }

    return 0;}

**Question number 2 output:**





**Question number 3 code:**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace labMid**

**{**

**internal class PasswordGenerator**

**{**

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

**{**

**if (registrationNumber.Length < 2)**

**{**

**return null;**

**}**

**Random random = new Random();**

**int positionOfNum1 = random.Next(0, 4);**

**int positionOfNum2 = random.Next(0, 4);**

**while (positionOfNum1 == positionOfNum2)**

**{**

**positionOfNum2 = random.Next(0, 5);**

**}**

**string password = "";**

**string lastTwoDigits = registrationNumber.Substring(registrationNumber.Length - 2);**

**int nums = random.Next(4, 8);**

**for (int i = 1; i <= nums; i++)**

**{**

**if (i == positionOfNum1)**

**{**

**password += lastTwoDigits[0];**

**}**

**else if (i == positionOfNum2)**

**{**

**password += lastTwoDigits[1];**

**}**

**else if (i == 3)**

**{**

**password += "1";**

**}**

**else if (i == 4)**

**{**

**password += "2";**

**}**

**else**

**{**

**password += random.Next(0, 10).ToString();**

**}**

**}**

**string specialCharacters = "!@#$%^&\*()\_-+=<>?";**

**int numberOfSpecialChars = random.Next(2, 5);**

**for (int i = 1; i <= numberOfSpecialChars; i++)**

**{**

**password += specialCharacters[random.Next(0, specialCharacters.Length)];**

**}**

**int chars = random.Next(1, 4);**

**password += firstName[0];**

**for (int i = 1; i <= chars; i++)**

**{**

**password += Char.ToUpper((char)('a' + random.Next(26)));**

**}**

**password += lastName[0];**

**if (password.Length > 16)**

**{**

**password = password.Substring(0, 16);**

**}**

**return ShufflePassword(password);**

**}**

**static string ShufflePassword(string input)**

**{**

**char[] characters = input.ToCharArray();**

**Random random = new Random();**

**for (int i = characters.Length - 1; i > 0; i--)**

**{**

**int j = random.Next(0, i + 1);**

**char temp = characters[i];**

**characters[i] = characters[j];**

**characters[j] = temp;**

**}**

**return new string(characters);**

**}**

**public static void Main(string[] args)**

**{**

**string pass = GeneratePassword("sp21-bcs-005", "nope", "fari");**

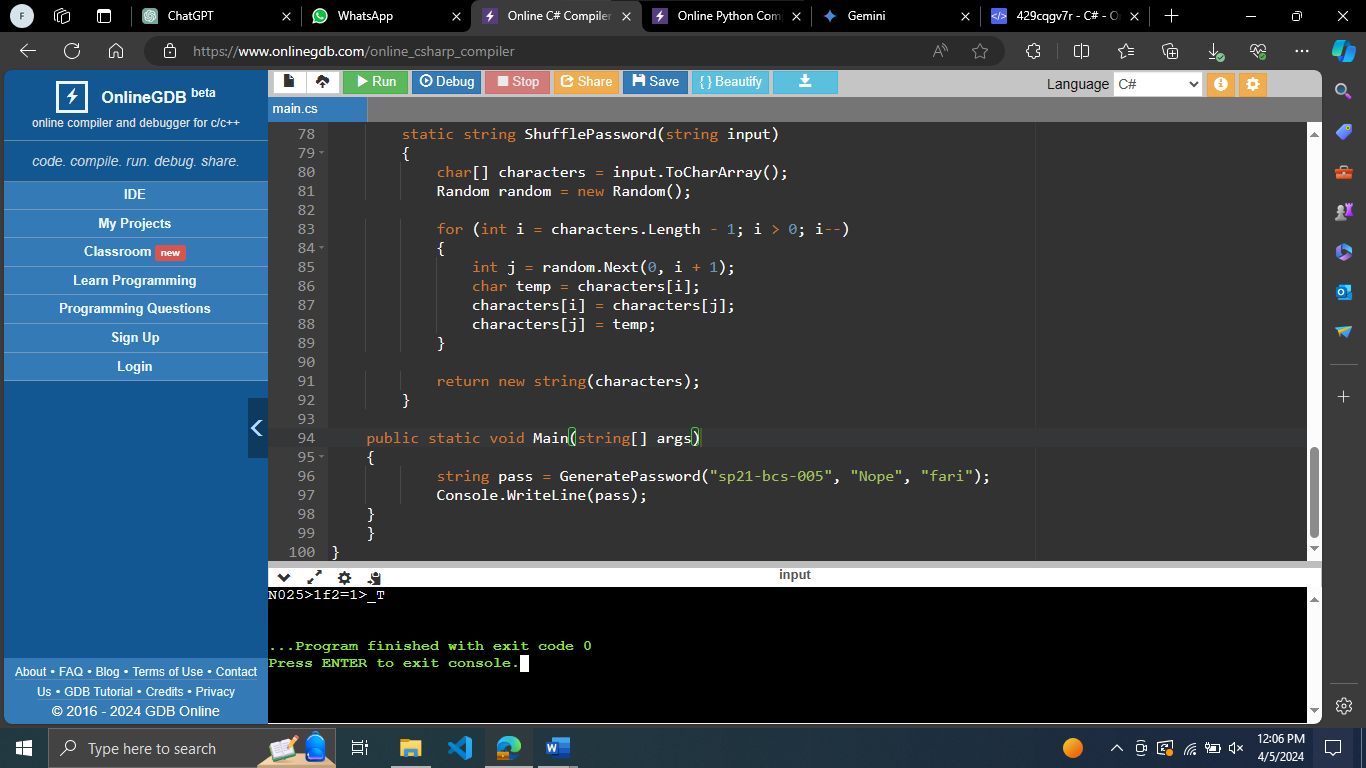
**Console.WriteLine(pass);**

**}**

**}**

**}**

**Question number 3 Output**

****