|  |  |
| --- | --- |
|  | **COMSATS University Islamabad, Attock Campus**  **Lab Terminal Examinations (Spring 2024)** |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Department of: | | **Computer Science** | | | | |  |
|  | |  | |  | | | | |  |
| Class/Program: | | **BS(CS)-7th** | | Date: | **31/05/ 2024 (1:30 - 4:30)** | | | | |
| Subject: | **Compiler construction Lab EXAM** | | | Instructor: | | **Bilal Haider** | | | |
| Total Time Allowed: | | | **3Hrs** | Maximum Marks: | | | | **50** | |
| Student Name: | | Hamid Ali Khan | | Registration #: | | | SP21-BCS-009 | | |
|  | |  | |  | | |  | | |

**To submit create a world file titled csc441-sp23-lab terminal-your complete registration number.docx.**

**Each question has 10 marks, please make a word file for answer of question 1 and 2, add screen shots of your input and output of question 3 and 4 in the same world document. Upload your code for project, question3 and question on google drive and paste it at the end of the word document**

**Viva will have 10 marks you can give viva until 27th of june 2023.**

**Question 1**

Write an introduction of your compiler construction project:

**INTRODUCTION:**

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or code that a computer's processors use. The file used for writing a C-language contains what are called the source statements. The programmer then runs the appropriate language compiler, specifying the name of the file that contains the source statements. When executing, the compiler first parses all of the language statements syntactically one after the other and then, in one or more successive stages, builds the output code, making sure that statements that refer to other statements are referred to correctly in the final code. The output of the compilation is called object code or sometimes an object module.

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, etc.

Symbol table is used by both the analysis and the synthesis parts of a compiler. We have designed a lexical analyzer for the C language using lex. It takes as input a C code and outputs a stream of tokens. The tokens displayed as part of the output include keywords, identifiers, signed/unsigned integer/floating point constants, operators, special characters, headers, data-type specifiers, array, single-line comment, multi-line comment, preprocessor directive, pre-defined functions (printf and scanf), user-defined functions and the main function. The token, the type of token and the line number of the token in the C code are being displayed. The line number is displayed so that it is easier to debug the code for errors. Errors in single-line comments, multi-line comments are displayed along with line numbers. The output also contains the symbol table which contains tokens and their type. The symbol table is generated using the hash organisation.

**Question 2**

Give a sample input and output for your compiler construction project:

**INPUT:**

#include<stdio.h>

int main()

{

    int a,i,j,flag=0;

    printf("Input no"); //Input

    scanf("%d",&a);

    i=3.1415E+3;

    j=127;

    float 3b = 9.5;

    while(i <= a/2)

    {

        if(a%i == 0)

        {

            flag=1;

            break;

        }

        i++;

    }

    if(flag==0)

        printf("Prime"); // It's a prime number.

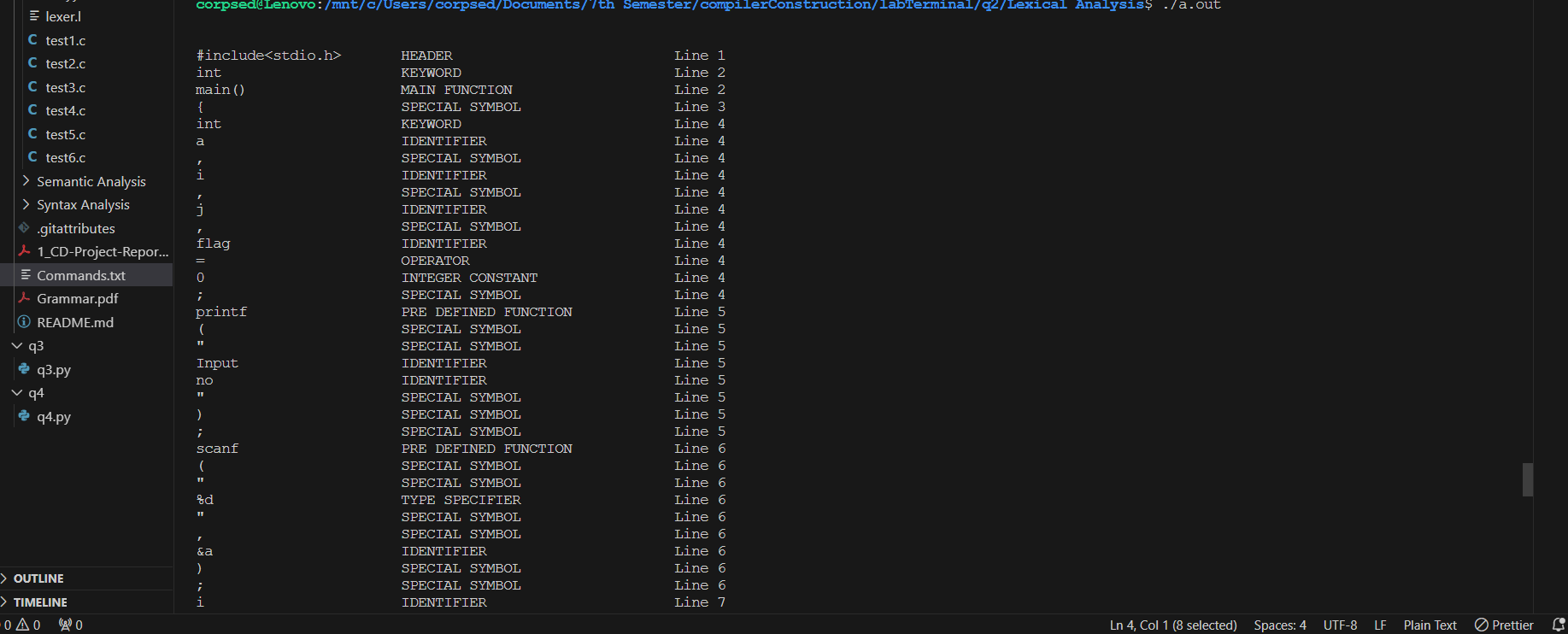
    else

        printf("Not Prime");

    return 0;

}

**OUTPUT:**

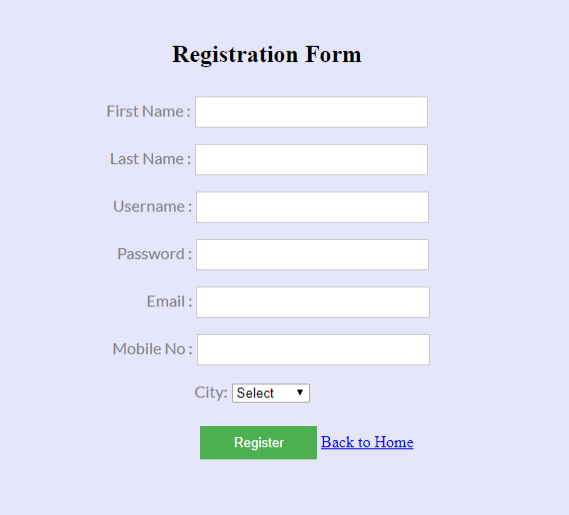


A screen shot of a computer

Description automatically generated

**Question 3**

Create and implement RE and DFAs for the form below



You must use Regex to validate data.

**CODE:**import re

# Regular Expressions

regexes = {

    "first\_name": re.compile(r"^[A-Za-z]+$"),

    "last\_name": re.compile(r"^[A-Za-z]+$"),

    "username": re.compile(r"^[A-Za-z0-9\_]{3,16}$"),

    "password": re.compile(r"^(?=.\*[A-Za-z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$"),

    "email": re.compile(r"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"),

    "mobile\_no": re.compile(r"^\d{10}$")

}

# DFA Simulation Function

def validate\_input(field, value):

    if field in regexes:

        return bool(regexes[field].match(value))

    return False

# Test Cases

test\_data = {

    "first\_name": "John",

    "last\_name": "Doe",

    "username": "john\_doe\_123",

    "password": "Passw0rd!",

    "email": "john.doe@example.com",

    "mobile\_no": "1234567890"

}

# Validation

validation\_results = {field: validate\_input(field, value) for field, value in test\_data.items()}

print(validation\_results)

**INPUT:**

test\_data = {

    "first\_name": "John",

    "last\_name": "Doe",

    "username": "john\_doe\_123",

    "password": "Passw0rd!",

    "email": "john.doe@example.com",

    "mobile\_no": "1234567890"

}

**OUTPUT:**



**Question 4:**

Write a program which generates symbol table for the code you submitted in question 3

**CODE:**

import re

class SymbolTable:

    def \_\_init\_\_(self):

        self.table = {}

    def add(self, name, type, value=None):

        self.table[name] = {"type": type, "value": value}

    def get(self, name):

        return self.table.get(name, None)

    def \_\_str\_\_(self):

        result = "Symbol Table:\n"

        result += "Name\t\tType\t\tValue\n"

        result += "-"\*40 + "\n"

        for name, info in self.table.items():

            result += f"{name}\t\t{info['type']}\t\t{info['value']}\n"

        return result

# Creating the symbol table

symbol\_table = SymbolTable()

# Adding variables to the symbol table

symbol\_table.add("regexes", "dictionary", {

    "first\_name": r"^[A-Za-z]+$",

    "last\_name": r"^[A-Za-z]+$",

    "username": r"^[A-Za-z0-9\_]{3,16}$",

    "password": r"^(?=.\*[A-Za-z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$",

    "email": r"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$",

    "mobile\_no": r"^\d{10}$"

})

symbol\_table.add("validate\_input", "function", "validates input based on regexes")

symbol\_table.add("test\_data", "dictionary", {

    "first\_name": "John",

    "last\_name": "Doe",

    "username": "john\_doe\_123",

    "password": "Passw0rd!",

    "email": "john.doe@example.com",

    "mobile\_no": "1234567890"

})

symbol\_table.add("validation\_results", "dictionary", None)

regexes = {

    "first\_name": re.compile(r"^[A-Za-z]+$"),

    "last\_name": re.compile(r"^[A-Za-z]+$"),

    "username": re.compile(r"^[A-Za-z0-9\_]{3,16}$"),

    "password": re.compile(r"^(?=.\*[A-Za-z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$"),

    "email": re.compile(r"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"),

    "mobile\_no": re.compile(r"^\d{10}$")

}

# Adding regexes to the symbol table

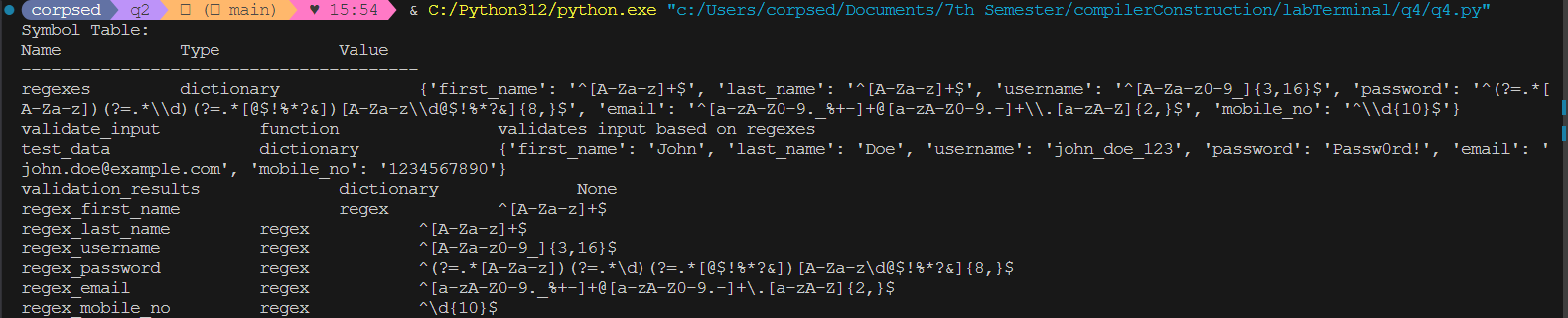
for key in regexes:

    symbol\_table.add(f"regex\_{key}", "regex", regexes[key].pattern)

# Printing the symbol table

print(symbol\_table)

**OUTPUT:**

****