|  |  |
| --- | --- |
|  | **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: | | Ahsan Ali | | Registration #: | | | SP21-BCS-001 | | |
|  | |  | |  | | |  | | |

**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

The C and C++ programming languages are closely related. C++ grew out of C, as it was designed to be source-and-link compatible with C. C++ was based on C and retains a great deal of the functionality. The C++ language provides mechanisms for mixing code that is compiled by compatible C and C++ compilers in the same program. As a matter of fact, C++ can run most of C code while C cannot run most C++ code.

The **purpose of compatibility** with C is so that C++ programs can have **convenient access** to the billions (trillions?) of lines of existing C code in the world.

Although, C and C++ code are almost compatible but there are still many incompatibilities or conflicts between them. The conflicts can be of two types:

1. **Incompatible C feature** - valid as C code but not as C++ code.
2. **Incompatible C++ feature** - valid as C++ code but not as C code.In this project we focus on a different domain. **Compatible** C/C++ features i.e. features of C code that are valid in C++.

We aim at detecting such snippets of code in our input program and will give an error if a C code is detected, whilst if no C code could be detected then we will compile it for minor errors, i.e. **a mini compiler strictly for C++.**

**Question 2**

Give a sample input and output for your compiler construction project

#include <iostream>

#include <cstdio>

#include <cmath>

using namespace std;

int main()

{

int x;

// C++ style IO

cout << “Hello World, Enter a number: ” << endl;

cin >> x;

/\* C Style IO \*/

printf(“Hello World, Enter a number: ”);

scanf(“%d”, &x);

/\* A fuction call invalid in C++ \*

\* if not including C header files. \*/

x = sqrt(4);

return 0;

}

In addition to detect C code inside C++ code, our Mini C++ compiler will also be able to report following errors to the user:

* Invalid variable name.
* Invalid basic arithmetic expression.
* Syntax error in While loop.
* Syntax errors in For loop.
* Syntax errors in If-Then-Else.

3. Tools used

We have used following two tools to implement our project:

1. YACC - produces a parser
2. LEX - generates lexical analyzers
3. GCC – gnu C Compiler
4. G++ - gnu C++ Compiler

4. Methodology or Algorithm

The project is implemented in the following steps: -

1. Read the given Input.
2. Tokenize the input using Lex rules.
3. Parse using Yacc rules.
4. Run the Algorithm described below.

Algorithm:-

Goal: detect a c code that is generally successfully compiled by a C++ compiler and accept small C++ codes.

Steps:

1. Detect for header files (generally all c codes have **.h** header files)
2. Detect C language functions and keywords that are compatible with C++ compilers.
3. If(c code detected )

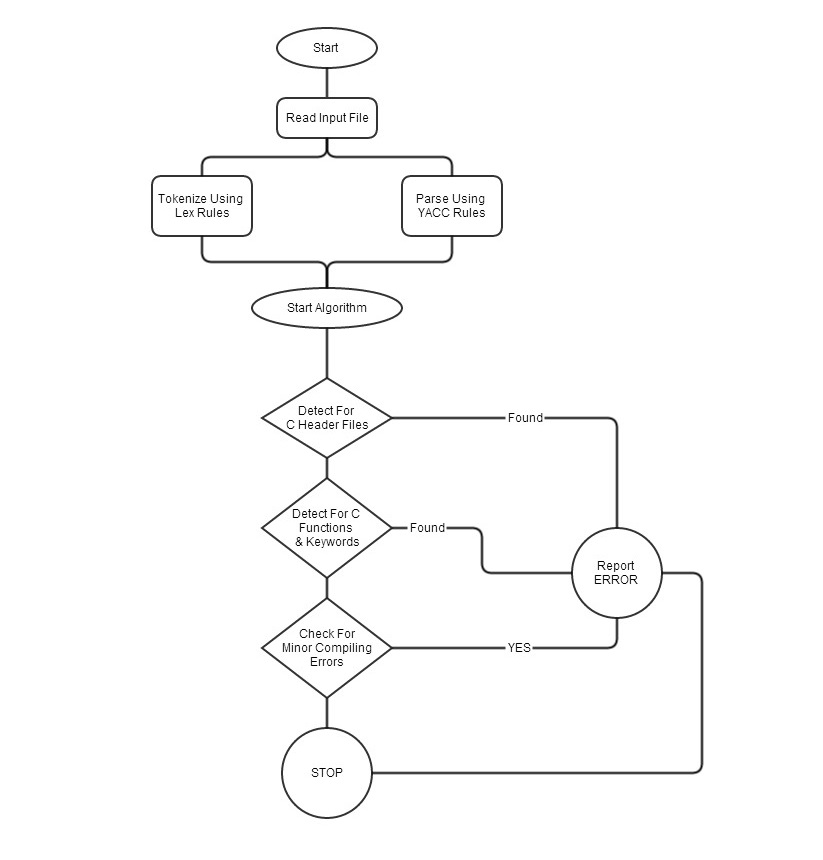
Then return ERROR and STOP.

Else goto step 4.

4. Check for error like

* + - * + Invalid variable name.
        + Invalid basic arithmetic expression.
        + Syntax error in While loop.
        + Syntax errors in For loop.
        + Syntax errors in If-Then-Else.

5. Flowchart



6. Work done so Far

1. Did the required literature survey.

2. Design our Algorithm

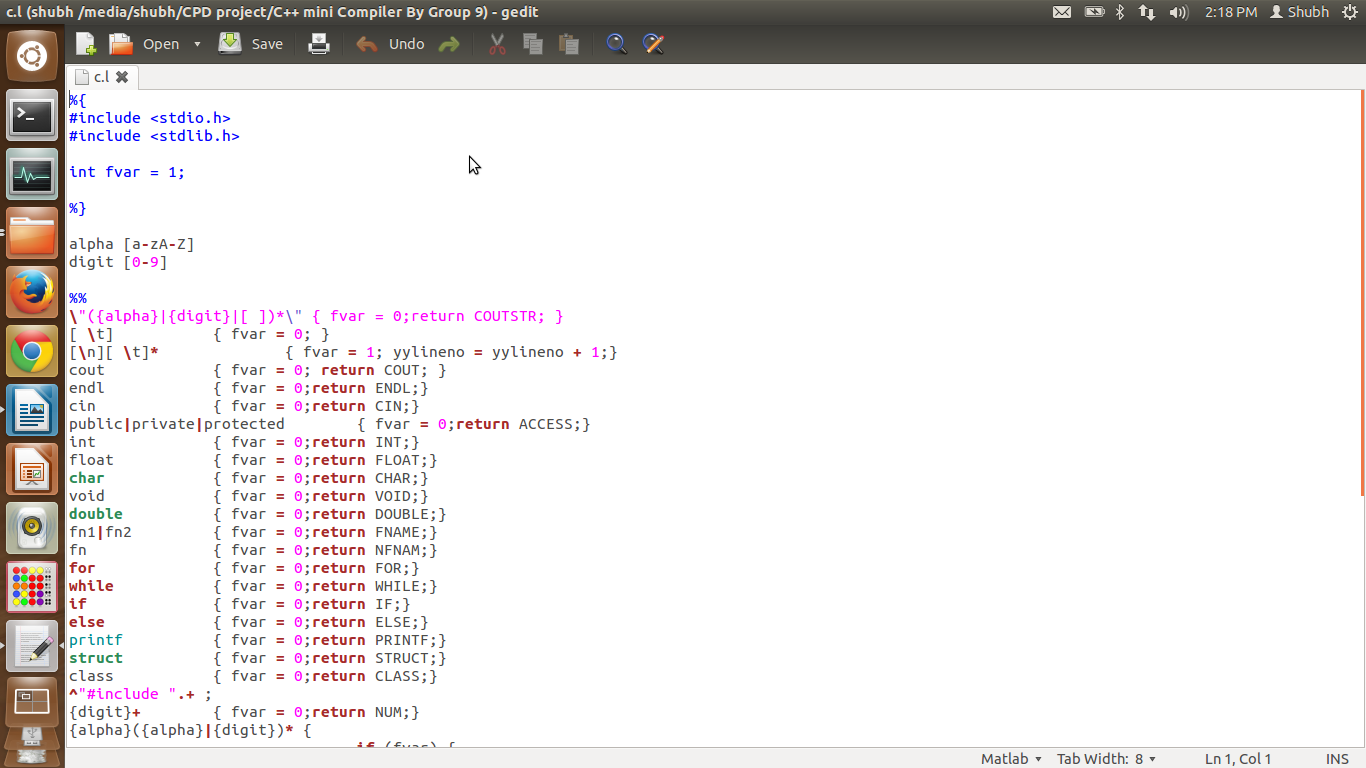
3. Flow chart prepared.

4. Project report 1 created.

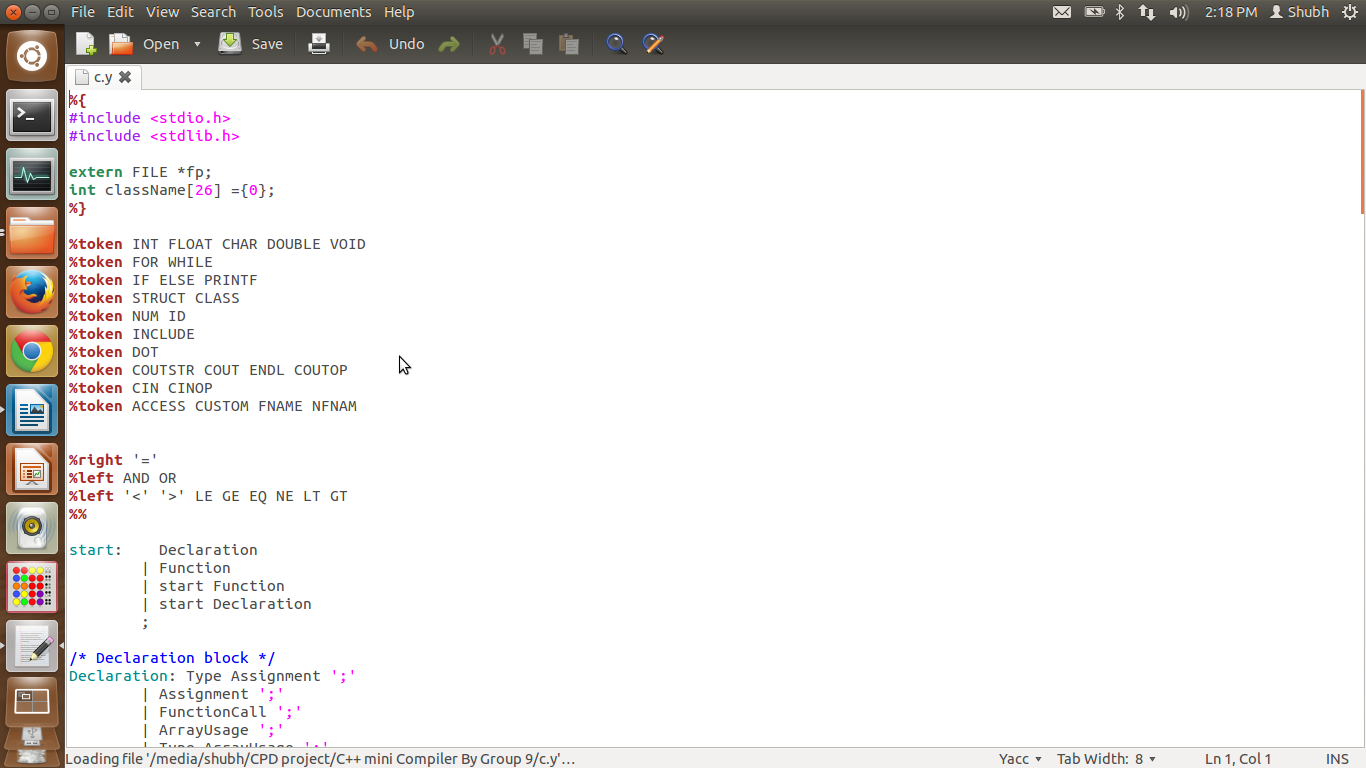
5. Implemented the mini compiler for C++.

6. Completed the project with results verified with test files.

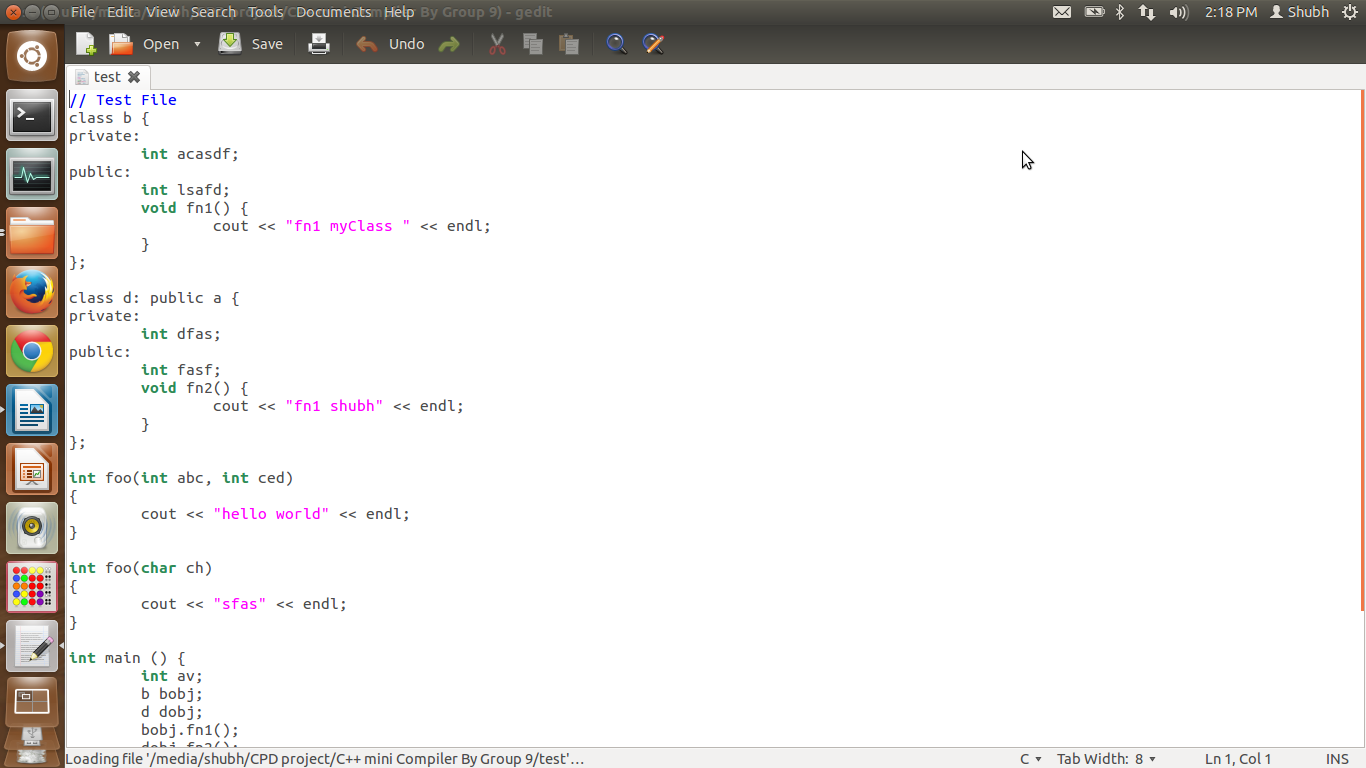
**Lex File Snap**

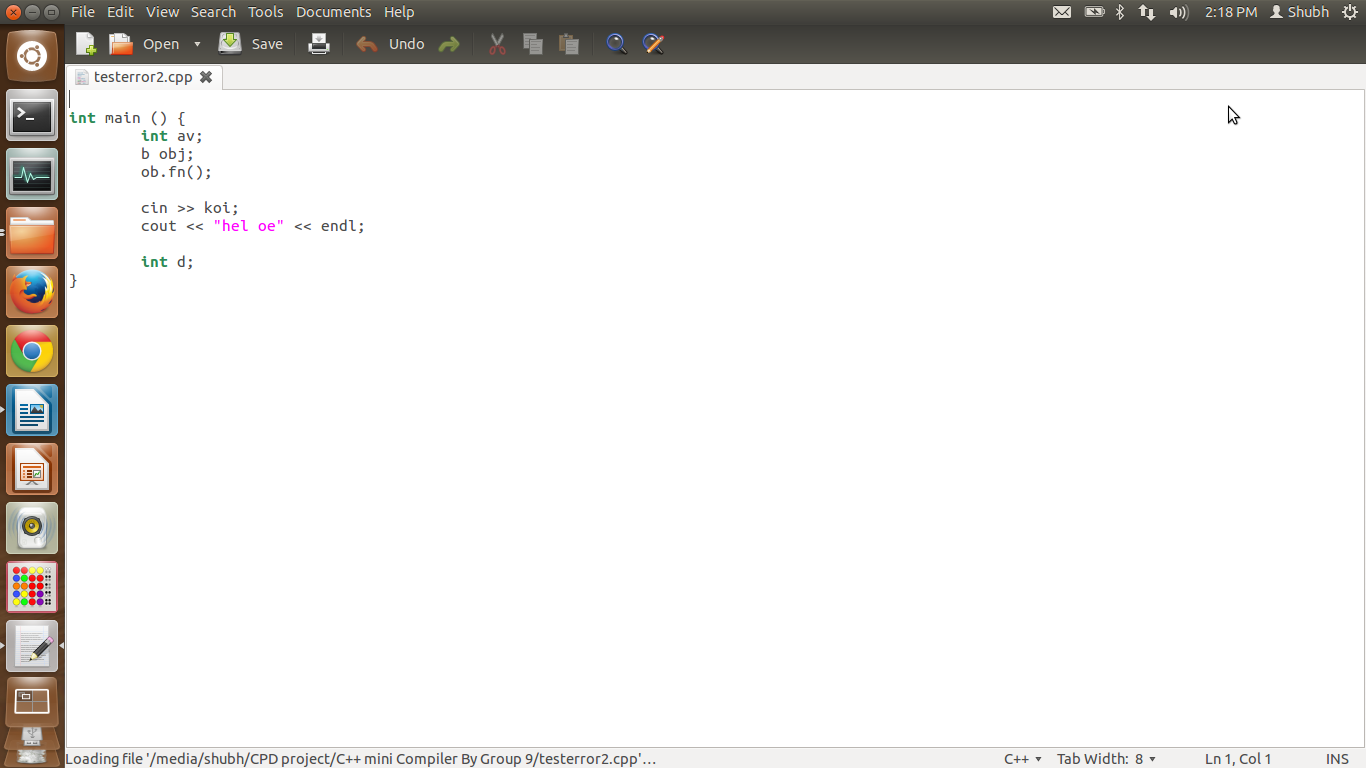


**Yacc File Snap**

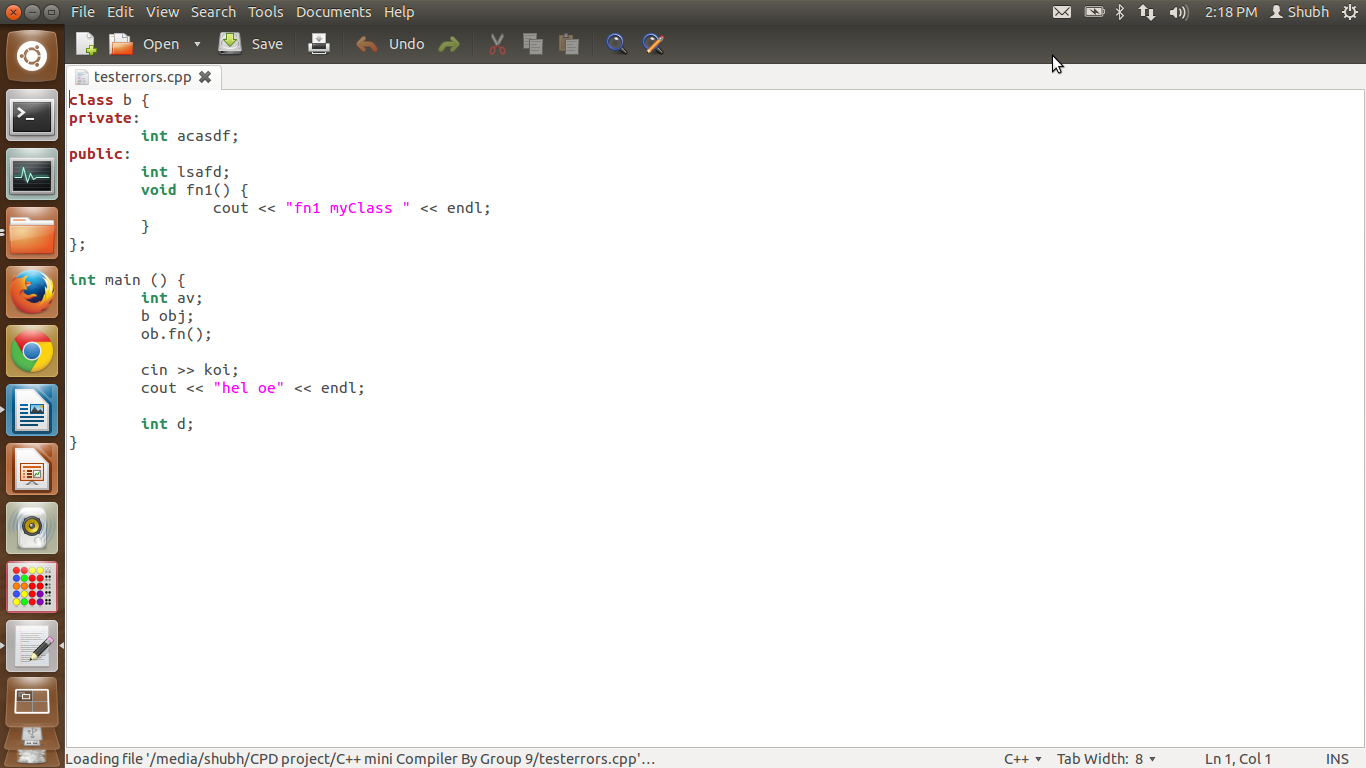


Test File For Polymorphism:-



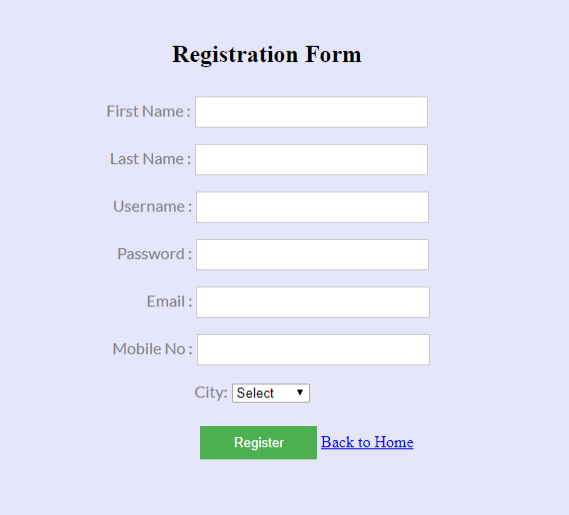


Test file for error



**Question 3**

Create and implement RE and DFAs for the form below



You must use Regex to validate data.

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)

**Question 4:**

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

import re

class SymbolTable:

    def \_\_init\_\_(self):  # Corrected to use double underscores

        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):  # Corrected to use double underscores

        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)