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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Department of: | **Computer Science** | | |  |
|  |  |  | | |  |
| Class/Program: | **BS(CS)-7th** | Date: | **31/05/ 2024** | | |
| Student Name: | Jahanzeb Razzaq | Registration #: | | SP21-BCS-013 | |
|  |  |  | |  | |

**Question 1:**

Write an introduction of your compiler construction project

**Answer:**

The mini-cpp-compiler project is a simple compiler for a subset of the C++ programming language, designed as a course project for a Compiler Design class. The main goal is to generate optimized intermediate code for a C++-like language, focusing on constructs such as conditional statements, loops, and the ternary operator.

The project uses Lex and Yacc compiler generation packages, along with a simple GUI built using C++. It follows these key steps:

Generate a symbol table after performing expression evaluation.

Generate an Abstract Syntax Tree (AST) for the code.

Generate 3-address code followed by corresponding quadruples.

Perform code optimization.

The project demonstrates the implementation of a compiler's frontend (scanner and parser) and backend (static semantics, code generation, and optimization). It serves as a learning resource for understanding the fundamentals of compiler design and implementation**.**

**Question 2**

Give a sample input and output for your compiler construction project

**Answer:**

INPUT ----------------------

// Sample input code

int main() {

int x = 5;

if (x > 3) {

int y = 7;

cout << "x is greater than 3" << endl;

} else {

cout << "x is less than or equal to 3" << endl;

}

    return 0;

}

OUTPUT-------------------

cpp

// Optimized intermediate code

main:

// Initialize x

x = 5

// Conditional statement

if (x > 3) {

// Initialize y

y = 7

// Print statement

cout << "x is greater than 3" << endl

} else {

// Print statement

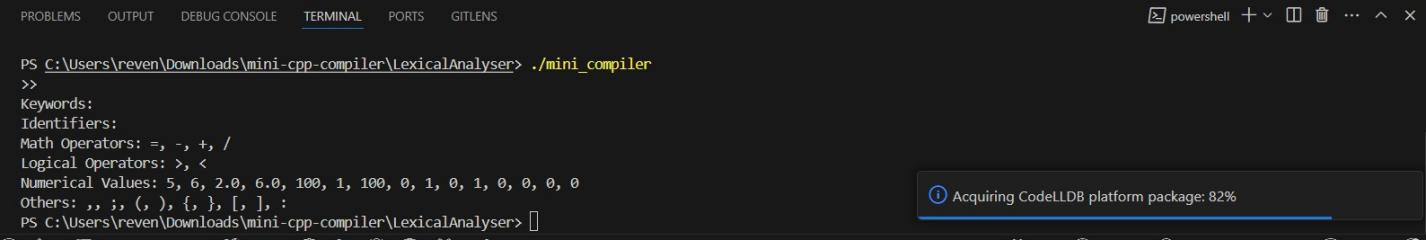
cout << "x is less than or equal to 3" << endl

}

// Return statement

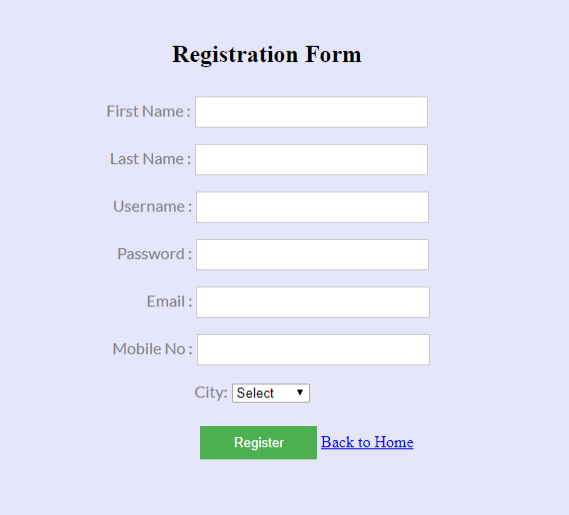
return 0

**Output:**



**Question 3**

Create and implement RE and DFAs for the form below



You must use Regex to validate data.

**Answer:**

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

**Answer:**

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)