**Project Documentation**

**  
Attock Campus**

**Submitted By:**

Hammas Rashid (SP21-BCS-011)

**Submitted To:**

Sir Bilal Haider Bukhari

**Subject:**

Compiler Construction

**Date:**

31st May 2024

**Question 1:**

Write an introduction of your compiler construction project.

**Ans:**

**Table of Contents**

1. Introduction
2. Architecture Overview
3. Components

* Lexer
* Parser
* Code Generator
* Virtual Machine
* Main

1. Usage
2. Example
3. Enhancements
4. Troubleshooting

**Introduction**

The Mini Python Compiler is a simple interpreter for a subset of Python-like syntax. It reads, tokenizes, parses, generates bytecode, and executes code using a virtual machine. The purpose of this compiler is educational, helping you understand the basic concepts of lexing, parsing, code generation, and virtual machine execution.

**Architecture Overview**

The compiler is divided into several stages:

**Lexing:** Convert input code into tokens.

**Parsing:** Convert tokens into an Abstract Syntax Tree (AST).

**Code Generation:** Convert AST into bytecode instructions.

**Execution:** Execute bytecode using a virtual machine.

**Components**

**Lexer:**

# lexer.py

import re

class Lexer:

def \_\_init\_\_(self, code):

self.code = code

self.tokens = []

self.token\_specification = [

('NUMBER', r'\d+(\.\d\*)?'), # Integer or decimal number

('ASSIGN', r'='), # Assignment operator

('END', r';'), # Statement terminator

('ID', r'[A-Za-z]+'), # Identifiers

('OP', r'[+\-\*/]'), # Arithmetic operators

('LPAREN', r'\('), # Left parenthesis

('RPAREN', r'\)'), # Right parenthesis

('NEWLINE', r'\n'), # Line endings

('SKIP', r'[ \t]+'), # Skip over spaces and tabs

('MISMATCH', r'.'), # Any other character

]

self.token\_regex = '|'.join(f'(?P<{pair[0]}>{pair[1]})' for pair in self.token\_specification)

def tokenize(self):

for match in re.finditer(self.token\_regex, self.code):

kind = match.lastgroup

value = match.group(kind)

if kind == 'NUMBER':

value = float(value) if '.' in value else int(value)

elif kind == 'ID' and value in {'if', 'else', 'while', 'for', 'def', 'return', 'class'}:

kind = value.upper()

elif kind == 'NEWLINE':

continue

elif kind == 'SKIP':

continue

elif kind == 'MISMATCH':

raise RuntimeError(f'Unexpected character: {value}')

self.tokens.append((kind, value))

return self.tokens

**Parser:**

class Parser:

def \_\_init\_\_(self, tokens):

self.tokens = tokens

self.current\_index = 0

self.current\_token = self.tokens[self.current\_index]

def consume(self, expected\_type):

if self.current\_token[0] == expected\_type:

self.current\_index += 1

if self.current\_index < len(self.tokens):

self.current\_token = self.tokens[self.current\_index]

else:

raise RuntimeError(f'Expected {expected\_type} but got {self.current\_token[0]}')

def parse(self):

statements = []

while self.current\_index < len(self.tokens):

statements.append(self.statement())

return ('statements', statements)

def statement(self):

node = self.assignment()

self.consume('END')

return node

def assignment(self):

left = self.current\_token

self.consume('ID')

self.consume('ASSIGN')

right = self.expression()

return ('assign', left, right)

def expression(self):

node = self.term()

while self.current\_token[0] in ('OP',) and self.current\_token[1] in ('+', '-'):

token = self.current\_token

self.consume('OP')

node = ('binop', token, node, self.term())

return node

def term(self):

node = self.factor()

while self.current\_token[0] in ('OP',) and self.current\_token[1] in ('\*', '/'):

token = self.current\_token

self.consume('OP')

node = ('binop', token, node, self.factor())

return node

def factor(self):

token = self.current\_token

if token[0] == 'NUMBER':

self.consume('NUMBER')

return ('num', token)

elif token[0] == 'LPAREN':

self.consume('LPAREN')

node = self.expression()

self.consume('RPAREN')

return node

elif token[0] == 'ID':

self.consume('ID')

return ('id', token)

else:

raise RuntimeError(f'Unexpected token: {token}')

**Code Generator:**class CodeGenerator:

def \_\_init\_\_(self, ast):

self.ast = ast

self.code = []

def generate(self):

self.visit(self.ast)

return '\n'.join(self.code)

def visit(self, node):

node\_type = node[0]

if node\_type == 'statements':

for statement in node[1]:

self.visit(statement)

elif node\_type == 'num':

self.code.append(f'PUSH {node[1][1]}')

elif node\_type == 'id':

self.code.append(f'LOAD {node[1][1]}')

elif node\_type == 'assign':

self.visit(node[2]) # Visit the right-hand side (expression)

self.code.append(f'STORE {node[1][1]}')

elif node\_type == 'binop':

self.visit(node[2]) # Visit left operand

self.visit(node[3]) # Visit right operand

op = node[1][1]

if op == '+':

self.code.append('ADD')

elif op == '-':

self.code.append('SUB')

elif op == '\*':

self.code.append('MUL')

elif op == '/':

self.code.append('DIV')

**Virtual Machine:**class VirtualMachine:

def \_\_init\_\_(self, instructions):

self.instructions = instructions

self.stack = []

self.variables = {}

self.instruction\_pointer = 0

def run(self):

while self.instruction\_pointer < len(self.instructions):

instr = self.instructions[self.instruction\_pointer]

parts = instr.split()

op = parts[0]

if op == 'PUSH':

self.stack.append(float(parts[1]))

elif op == 'LOAD':

var\_name = parts[1]

self.stack.append(self.variables[var\_name])

elif op == 'STORE':

var\_name = parts[1]

value = self.stack.pop()

self.variables[var\_name] = value

elif op == 'ADD':

b = self.stack.pop()

a = self.stack.pop()

self.stack.append(a + b)

elif op == 'SUB':

b = self.stack.pop()

a = self.stack.pop()

self.stack.append(a - b)

elif op == 'MUL':

b = self.stack.pop()

a = self.stack.pop()

self.stack.append(a \* b)

elif op == 'DIV':

b = self.stack.pop()

a = self.stack.pop()

self.stack.append(a / b)

elif op == 'MOD':

b = self.stack.pop()

a = self.stack.pop()

self.stack.append(a % b)

elif op == 'POW':

b = self.stack.pop()

a = self.stack.pop()

self.stack.append(a \*\* b)

elif op == 'JZ':

label = parts[1]

if self.stack.pop() == 0:

self.instruction\_pointer = self.labels[label]

elif op == 'JMP':

label = parts[1]

self.instruction\_pointer = self.labels[label]

self.instruction\_pointer += 1

# Example usage:

instructions = [

'PUSH 5',

'PUSH 3',

'PUSH 2',

'PUSH 8',

'SUB',

'MUL',

'ADD',

'STORE x'

]

vm = VirtualMachine(instructions)

vm.run()

print(vm.variables)

**Main:**

from lexer import Lexer

from parser import Parser

from code\_generator import CodeGenerator

from virtual\_machine import VirtualMachine

def read\_input\_code():

print("Enter your code (end with an empty line):")

lines = []

while True:

line = input()

if line.strip() == "":

break

lines.append(line)

return "\n".join(lines)

def main():

# Step 1: Read input code from the user

input\_code = read\_input\_code()

print(f"Input Code:\n{input\_code}")

# Step 2: Tokenize the code

lexer = Lexer(input\_code)

tokens = lexer.tokenize()

print(f"Tokens: {tokens}")

# Step 3: Parse the tokens to generate an AST

parser = Parser(tokens)

ast = parser.parse()

print(f"AST: {ast}")

# Step 4: Generate bytecode from the AST

code\_generator = CodeGenerator(ast)

bytecode = code\_generator.generate()

print(f"Generated Bytecode:\n{bytecode}")

# Step 5: Execute the bytecode using the virtual machine

vm = VirtualMachine(bytecode.split('\n'))

vm.run()

print("Execution Result:")

print(vm.variables)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Usage:  
Command:**

python main.py

**Question 2**

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

**Ans:**

**Example:**

**Input:**

x = 5 + 3 \* (2 - 8);

**The output will be:**

Tokens: [('ID', 'x'), ('ASSIGN', '='), ('NUMBER', 5), ('OP', '+'), ('NUMBER', 3), ('OP', '\*'), ('LPAREN', '('), ('NUMBER', 2), ('OP', '-'), ('NUMBER', 8), ('RPAREN', ')'), ('END', ';')]

AST: ('assign', ('ID', 'x'), ('binop', ('OP', '+'), ('num', ('NUMBER', 5)), ('binop', ('OP', '\*'), ('num', ('NUMBER', 3)), ('binop', ('OP', '-'), ('num', ('NUMBER', 2)), ('num', ('NUMBER', 8))))))

Generated Bytecode:

PUSH 5

PUSH 3

PUSH 2

PUSH 8

SUB

MUL

ADD

STORE x

Execution Result:

{'x': -13.0}

**Enhancements**

Possible enhancements to the mini compiler include:

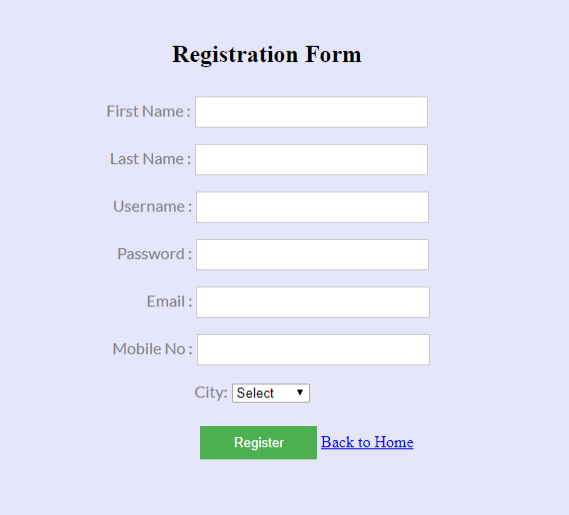
* Support for more complex data types (strings, floats, etc.).
* More comprehensive error handling and reporting.
* Additional arithmetic and logical operations.
* Control flow constructs (if statements, loops, etc.).
* Function definitions and calls.
* Optimization passes in the code generation stage.

**Troubleshooting**

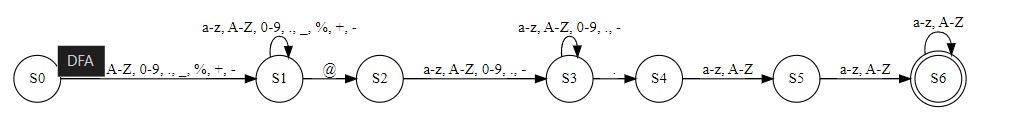
* **Unexpected character error:** Make sure the input code contains only valid characters.
* **Expected END but got (token):** Ensure proper syntax in the input code. Missing semicolons or mismatched parentheses can cause this error.
* **TypeError:** This may occur if the parser does not generate a correct AST. Ensure that the parser correctly handles all expected constructs.
* **Stack underflow in the VM:** This indicates that an operation expected more values on the stack than were present. Ensure the bytecode is correctly generated and balanced.

**Question 3**

Create and implement RE and DFAs for the form below.



You must use Regex to validate data.

**Ans:**

**Question 4:**

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

**Ans:**

**index.html**

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Registration Form</title>

    <style>

      body {

        font-family: Arial, sans-serif;

        background-color: #f0f0f0;

        display: flex;

        justify-content: center;

        align-items: center;

        height: 100vh;

      }

      .form-container {

        background-color: #d6e0f5;

        padding: 20px;

        border-radius: 8px;

        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

        width: 100%;

      }

      .form-container h2 {

        text-align: center;

      }

      .form-group {

        margin-bottom: 15px;

      }

      .form-group label {

        display: block;

        margin-bottom: 5px;

      }

      .form-group input,

      .form-group select {

        width: 100%;

        padding: 8px;

        box-sizing: border-box;

      }

      .form-group input[type="submit"] {

        background-color: #4caf50;

        color: white;

        border: none;

        cursor: pointer;

      }

      .form-group input[type="submit"]:hover {

        background-color: #45a049;

      }

      .error {

        color: red;

        font-size: 12px;

        margin-top: 5px;

      }

    </style>

    <script>

      function validateForm() {

        let isValid = true;

        // Regular Expressions for validation

        const nameRegex = /^[A-Z][a-zA-Z]{1,}$/;

        const usernameRegex = /^[a-zA-Z0-9]{5,15}$/;

        const passwordRegex =

          /^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/;

        const emailRegex = /^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

        const mobileRegex = /^\d{10}$/;

        // Get form values

        const firstName = document.getElementById("firstName").value;

        const lastName = document.getElementById("lastName").value;

        const username = document.getElementById("username").value;

        const password = document.getElementById("password").value;

        const email = document.getElementById("email").value;

        const mobile = document.getElementById("mobile").value;

        // Validate each field and show error messages

        if (!nameRegex.test(firstName)) {

          document.getElementById("firstNameError").textContent =

            "Invalid First Name. Should start with a capital letter and contain only alphabets.";

          isValid = false;

        } else {

          document.getElementById("firstNameError").textContent = "";

        }

        if (!nameRegex.test(lastName)) {

          document.getElementById("lastNameError").textContent =

            "Invalid Last Name. Should start with a capital letter and contain only alphabets.";

          isValid = false;

        } else {

          document.getElementById("lastNameError").textContent = "";

        }

        if (!usernameRegex.test(username)) {

          document.getElementById("usernameError").textContent =

            "Invalid Username. Should be alphanumeric and 5-15 characters long.";

          isValid = false;

        } else {

          document.getElementById("usernameError").textContent = "";

        }

        if (!passwordRegex.test(password)) {

          document.getElementById("passwordError").textContent =

            "Invalid Password. Should contain at least one uppercase letter, one lowercase letter, one digit, and one special character. Minimum length: 8 characters.";

          isValid = false;

        } else {

          document.getElementById("passwordError").textContent = "";

        }

        if (!emailRegex.test(email)) {

          document.getElementById("emailError").textContent =

            "Invalid Email. Should follow the standard email format (e.g., user@example.com).";

          isValid = false;

        } else {

          document.getElementById("emailError").textContent = "";

        }

        if (!mobileRegex.test(mobile)) {

          document.getElementById("mobileError").textContent =

            "Invalid Mobile Number. Should be a 10-digit number.";

          isValid = false;

        } else {

          document.getElementById("mobileError").textContent = "";

        }

        // If all fields are valid, navigate to the success page

        if (isValid) {

          window.location.href = "success.html";

        }

        return false;

      }

    </script>

  </head>

  <body>

    <div class="form-container">

      <h2>Registration Form</h2>

      <form onsubmit="return validateForm()">

        <div class="form-group">

          <label for="firstName">First Name:</label>

          <input type="text" id="firstName" name="firstName" required />

          <div id="firstNameError" class="error"></div>

        </div>

        <div class="form-group">

          <label for="lastName">Last Name:</label>

          <input type="text" id="lastName" name="lastName" required />

          <div id="lastNameError" class="error"></div>

        </div>

        <div class="form-group">

          <label for="username">Username:</label>

          <input type="text" id="username" name="username" required />

          <div id="usernameError" class="error"></div>

        </div>

        <div class="form-group">

          <label for="password">Password:</label>

          <input type="password" id="password" name="password" required />

          <div id="passwordError" class="error"></div>

        </div>

        <div class="form-group">

          <label for="email">Email:</label>

          <input type="email" id="email" name="email" required />

          <div id="emailError" class="error"></div>

        </div>

        <div class="form-group">

          <label for="mobile">Mobile No:</label>

          <input type="text" id="mobile" name="mobile" required />

          <div id="mobileError" class="error"></div>

        </div>

        <div class="form-group">

          <label for="city">City:</label>

          <select id="city" name="city" required>

            <option value="" disabled selected>Select</option>

            <option value="City1">Attock</option>

            <option value="City2">Lahore</option>

            <option value="City3">Islamabad</option>

          </select>

        </div>

        <div class="form-group">

          <input type="submit" value="Register" />

        </div>

      </form>

    </div>

  </body>

</html>

**Success.html:**

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Success</title>

    <style>

      body {

        font-family: Arial, sans-serif;

        background-color: #f0f0f0;

        display: flex;

        justify-content: center;

        align-items: center;

        height: 100vh;

      }

      .message-container {

        background-color: #d6e0f5;

        padding: 20px;

        border-radius: 8px;

        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

        text-align: center;

      }

    </style>

  </head>

  <body>

    <div class="message-container">

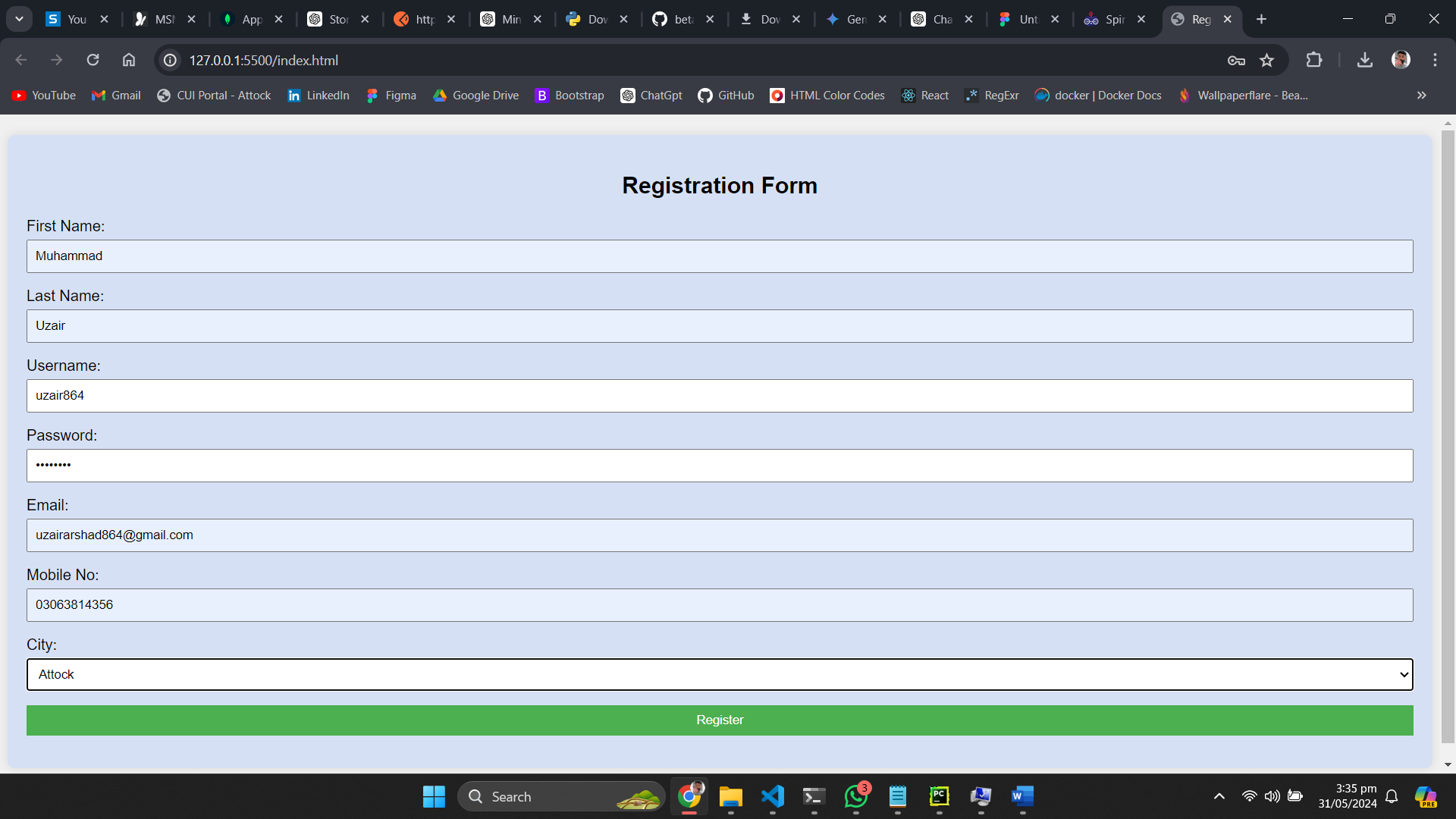
      <h2>Form Submitted Successfully!</h2>

    </div>

  </body>

</html>

**Output:**



**After Pressing the Register button:**