**Lab Terminal**

**  
Attock Campus**

**Submitted By:**

Faraidon (SP21-BCS-005)

Fakhrul Hassan (SP21-BCS-004)

**Submitted To:**

Sir Bilal Haider Bukhari

**Subject:**

Compiler Construction

**Date:**

31st May 2024

**Question 1:**

Write an introduction of your compiler construction project.

**Ans:**

**Introduction:**

In the realm of programming, compilers play a pivotal role in translating human-readable code into machine-executable instructions. This project endeavors to explore the fundamental concepts of lexing, parsing, semantic analysis, and code generation through the implementation of a simple arithmetic expression compiler.

The primary goal of this compiler is to take arithmetic expressions as input, analyze their structure, perform semantic checks, and generate bytecode that represents the equivalent operation. The process unfolds in several stages, each crucial for ensuring the correctness and efficiency of the generated code.

At its core, the compiler comprises four key components:

* **Lexer (Tokenization):** The lexer breaks down the input expression into a sequence of tokens, where each token represents a meaningful unit such as numbers, operators, or parentheses. Regular expressions are employed to match and identify these tokens within the input stream.
* **Parser (Syntax Analysis):** Once the input has been tokenized, the parser examines the sequence of tokens to discern the underlying syntactic structure of the expression. It constructs an abstract syntax tree (AST) that captures the hierarchical relationships between different components of the expression, facilitating further analysis and manipulation.
* **Semantic Analyzer**: With the AST in hand, the semantic analyzer traverses the tree to enforce semantic rules and constraints. It ensures that the expression adheres to the expected semantics, such as type compatibility and operator applicability. Any violations of these rules result in semantic errors being flagged.
* **Code Generator:** Finally, the code generator translates the validated AST into executable bytecode. It emits a sequence of low-level instructions that, when executed by a virtual machine or interpreter, perform the desired arithmetic operation. Each bytecode instruction corresponds to a specific action, such as pushing values onto a stack or performing arithmetic computations.Components

**Lexer:**

import re  
from collections import namedtuple  
  
# Token definition  
Token = namedtuple('Token', ['type', 'value'])  
  
# Lexer  
def lexer(text):  
 token\_specification = [  
 ('NUMBER', r'\d+'), # Integer  
 ('ADD', r'\+'), # Addition  
 ('MUL', r'\\*'), # Multiplication  
 ('LPAREN', r'\('), # Left Parenthesis  
 ('RPAREN', r'\)'), # Right Parenthesis  
 ('SKIP', r'[ \t]+'), # Skip over spaces and tabs  
 ('MISMATCH', r'.'), # Any other character  
 ]  
 tokens = []  
 for mo in re.finditer('|'.join(f'(?P<{pair[0]}>{pair[1]})' for pair in token\_specification), text):  
 kind = mo.lastgroup  
 value = mo.group()  
 if kind == 'NUMBER':  
 value = int(value)  
 elif kind == 'SKIP':  
 continue  
 elif kind == 'MISMATCH':  
 raise RuntimeError(f'{value} unexpected')  
 tokens.append(Token(kind, value))  
 return tokens

**Parser:**

# Parser  
class Parser:  
 def \_\_init\_\_(self, tokens):  
 self.tokens = tokens  
 self.pos = 0  
  
 def parse(self):  
 result = self.expr()  
 if self.pos != len(self.tokens):  
 raise SyntaxError('Unexpected token at the end')  
 return result  
  
 def expr(self):  
 node = self.term()  
 while self.pos < len(self.tokens) and self.tokens[self.pos].type == 'ADD':  
 self.pos += 1  
 node = ('ADD', node, self.term())  
 return node  
  
 def term(self):  
 node = self.factor()  
 while self.pos < len(self.tokens) and self.tokens[self.pos].type == 'MUL':  
 self.pos += 1  
 node = ('MUL', node, self.factor())  
 return node  
  
 def factor(self):  
 token = self.tokens[self.pos]  
 if token.type == 'NUMBER':  
 self.pos += 1  
 return ('NUMBER', token.value)  
 elif token.type == 'LPAREN':  
 self.pos += 1  
 node = self.expr()  
 if self.tokens[self.pos].type != 'RPAREN':  
 raise SyntaxError('Expected closing parenthesis')  
 self.pos += 1  
 return node  
 else:  
 raise SyntaxError(f'Unexpected token: {token.type}')  
  
# Semantic Analyzer  
def analyze(node):  
 if node[0] == 'NUMBER':  
 return 'int'  
 elif node[0] in ('ADD', 'MUL'):  
 left\_type = analyze(node[1])  
 right\_type = analyze(node[2])  
 if left\_type != 'int' or right\_type != 'int':  
 raise TypeError('Type error: Only integer operations are supported')  
 return 'int'  
 else:  
 raise TypeError(f'Unknown node type: {node[0]}')  
  
# Code Generator  
def generate\_code(node):  
 if node[0] == 'NUMBER':  
 return [f'PUSH {node[1]}']  
 elif node[0] in ('ADD', 'MUL'):  
 left\_code = generate\_code(node[1])  
 right\_code = generate\_code(node[2])  
 op\_code = 'ADD' if node[0] == 'ADD' else 'MUL'  
 return left\_code + right\_code + [op\_code]  
  
# Main function to compile and generate bytecode from input  
def compile\_expression(expression):  
 tokens = lexer(expression)  
 parser = Parser(tokens)  
 ast = parser.parse()  
 analyze(ast) # Perform semantic analysis  
 bytecode = generate\_code(ast)  
 return tokens, ast, bytecode  
  
# Get user input  
expression = input("Enter an arithmetic expression: ")  
tokens, ast, bytecode = compile\_expression(expression)  
  
print("Tokens:", tokens)  
print("AST:", ast)  
print("Bytecode:", bytecode)

**Usage:  
Command:**

python main.py

**Question 2**

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

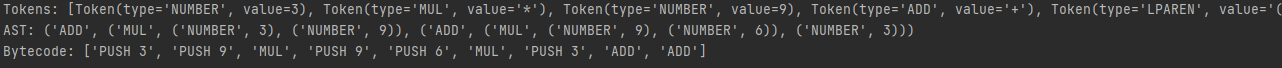
**Ans:**

**Example:**

**Input:**



**The output will be:**



Tokens: [Token(type='NUMBER', value=3), Token(type='MUL', value='\*'), Token(type='NUMBER', value=9), Token(type='ADD', value='+'), Token(type='LPAREN', value='('), Token(type='NUMBER', value=9), Token(type='MUL', value='\*'), Token(type='NUMBER', value=6), Token(type='ADD', value='+'), Token(type='NUMBER', value=3), Token(type='RPAREN', value=')')]

AST: ('ADD', ('MUL', ('NUMBER', 3), ('NUMBER', 9)), ('ADD', ('MUL', ('NUMBER', 9), ('NUMBER', 6)), ('NUMBER', 3)))

Bytecode: ['PUSH 3', 'PUSH 9', 'MUL', 'PUSH 9', 'PUSH 6', 'MUL', 'PUSH 3', 'ADD', 'ADD']

**Enhancements**

Possible enhancements to the mini compiler include:

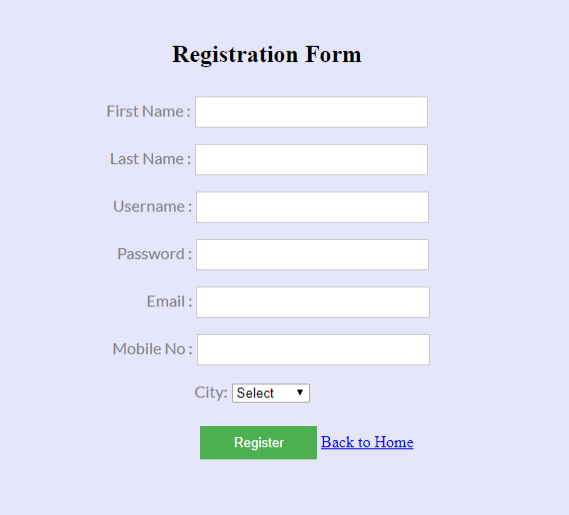
* Support for more complex data types (strings, floats, etc.).
* More comprehensive error handling and reporting.
* Additional arithmetic and logical operations.
* Control flow constructs (if statements, loops, etc.).
* Function definitions and calls.
* Optimization passes in the code generation stage.

**Troubleshooting**

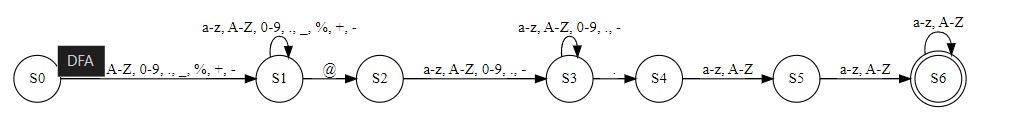
* **Unexpected character error:** Make sure the input code contains only valid characters.
* **Expected END but got (token):** Ensure proper syntax in the input code. Missing semicolons or mismatched parentheses can cause this error.
* **TypeError:** This may occur if the parser does not generate a correct 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">

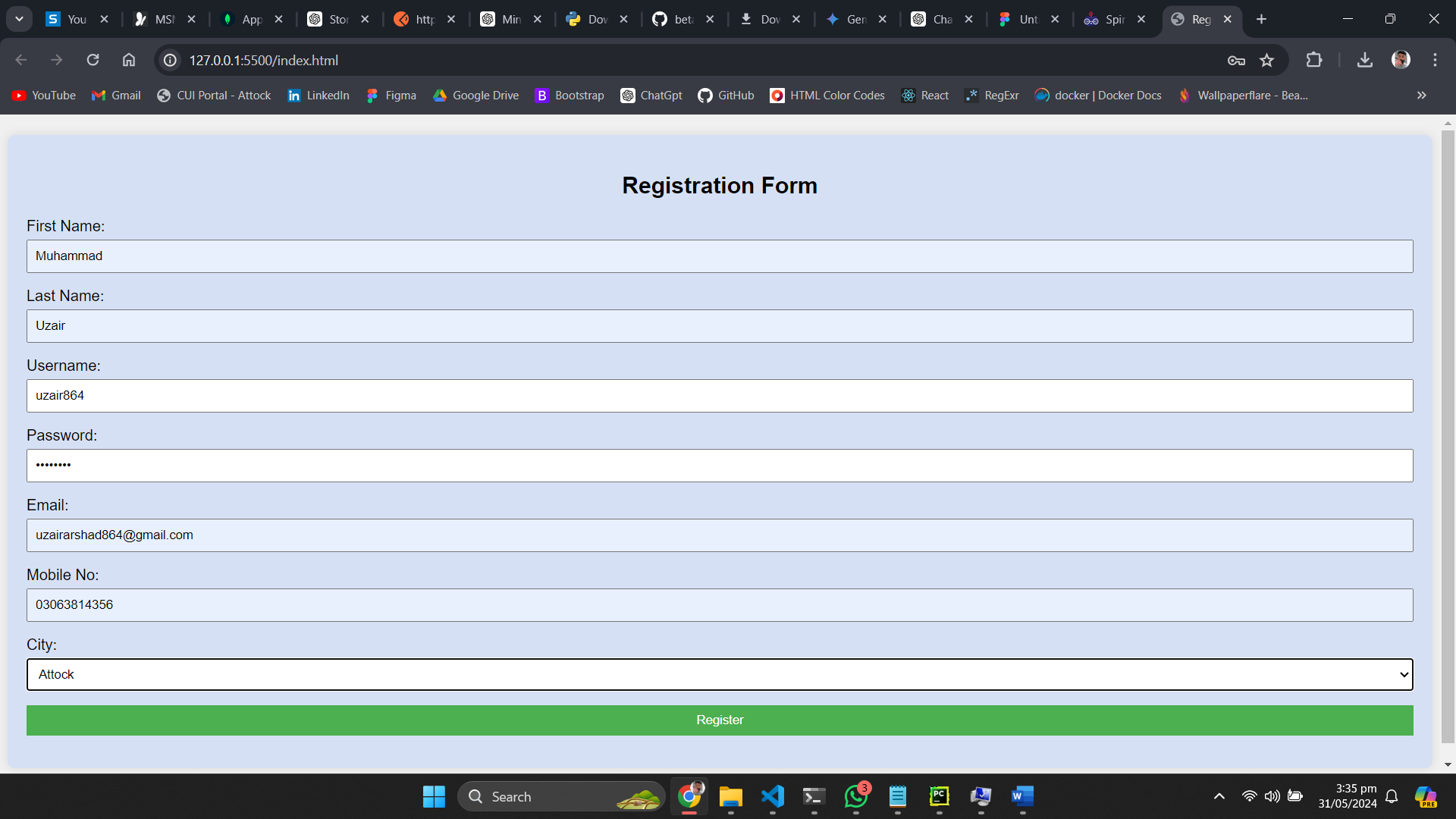
      <h2>Form Submitted Successfully!</h2>

    </div>

  </body>

</html>

**Output:**



**After Pressing the Register button:**