**Lab Terminal**

**COMSATS UNIVERSITY ISLAMABAD**

**Logo, company name

Description automatically generated**

**ATTOCK CAMPUS**

**Submitted By**

Salar Muhammad Umer(SP21-BCS-029)

Raja Muhammad Zeeshan(SP21-BS-025)

**Submitted To**

Mr. Syed Bilal Haider Bukhari

**Course Title**

Compiler Construction

**Date**

31-5-2024

Q1

**Project Purpose**

The purpose of this project is to provide a practical introduction to the fundamental concepts of compiler construction, specifically focusing on the initial stages of lexical and syntax analysis. The project aims to demonstrate how a compiler processes and translates source code into a structured format that can be further analyzed and executed.

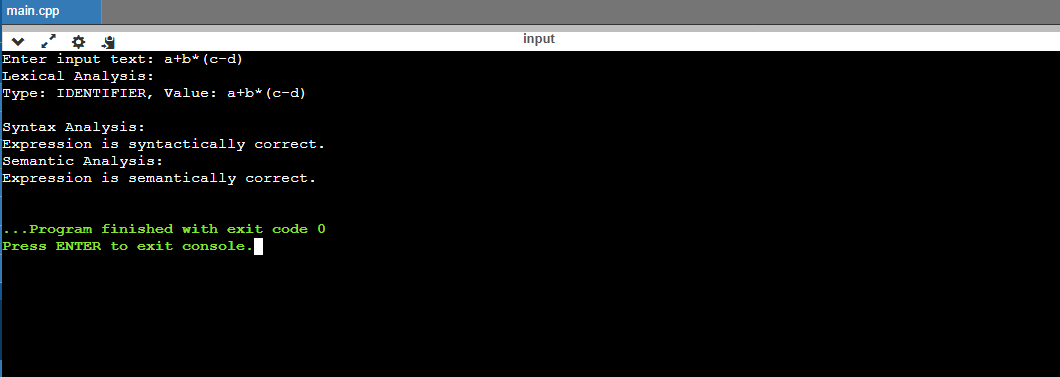
**Key Objectives:**

1. **Educational Insight**:
   * Offer students and developers a practical understanding of compiler operations by breaking down the complexities into manageable parts.
2. **Lexical Analysis**:
   * Illustrate the process of dividing source code into tokens, the smallest units of meaning, such as keywords, identifiers, numbers, and symbols.
3. **Syntax Analysis**:
   * Show how a syntax analyzer takes these tokens and ensures they follow the grammatical rules of the programming language, detecting syntax errors and constructing a hierarchical structure of the code.
4. **Foundation for Advanced Studies**:
   * Lay the groundwork for more advanced topics in compiler construction, such as semantic analysis, optimization, and code generation.

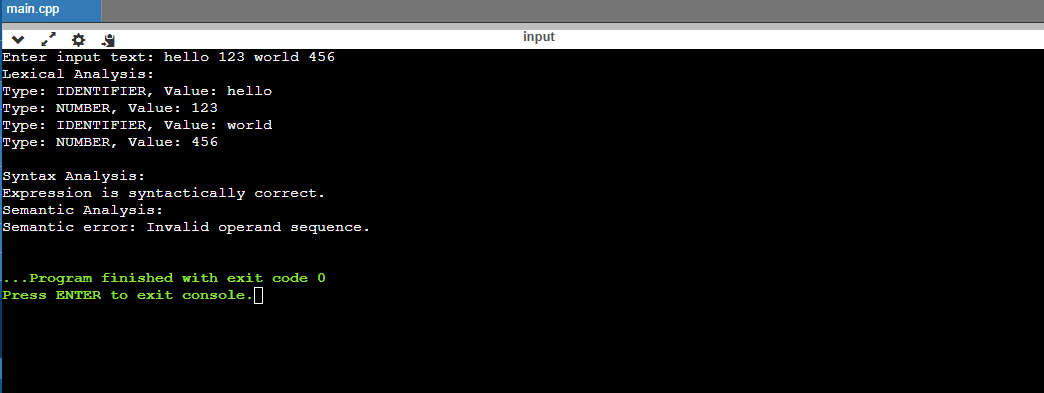
This project serves as an educational tool to enhance understanding of the initial stages of compilation, providing a stepping stone for more advanced studies and practical insights into compiler design and programming language development.

Q2

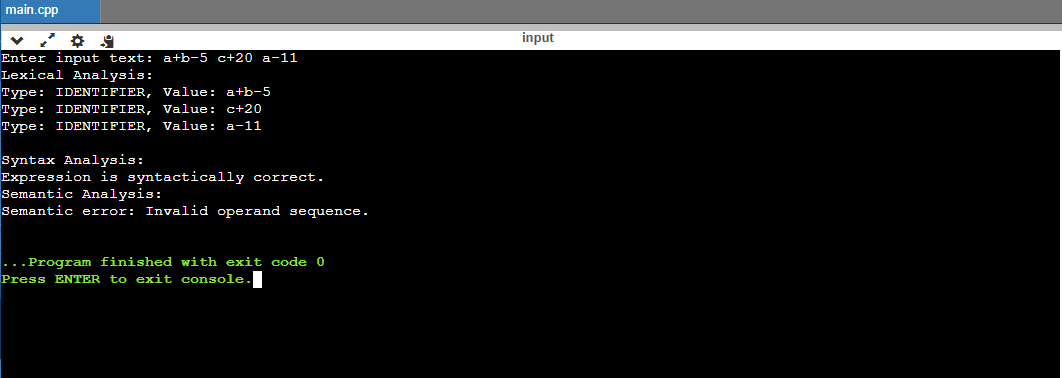
Screenshot1:



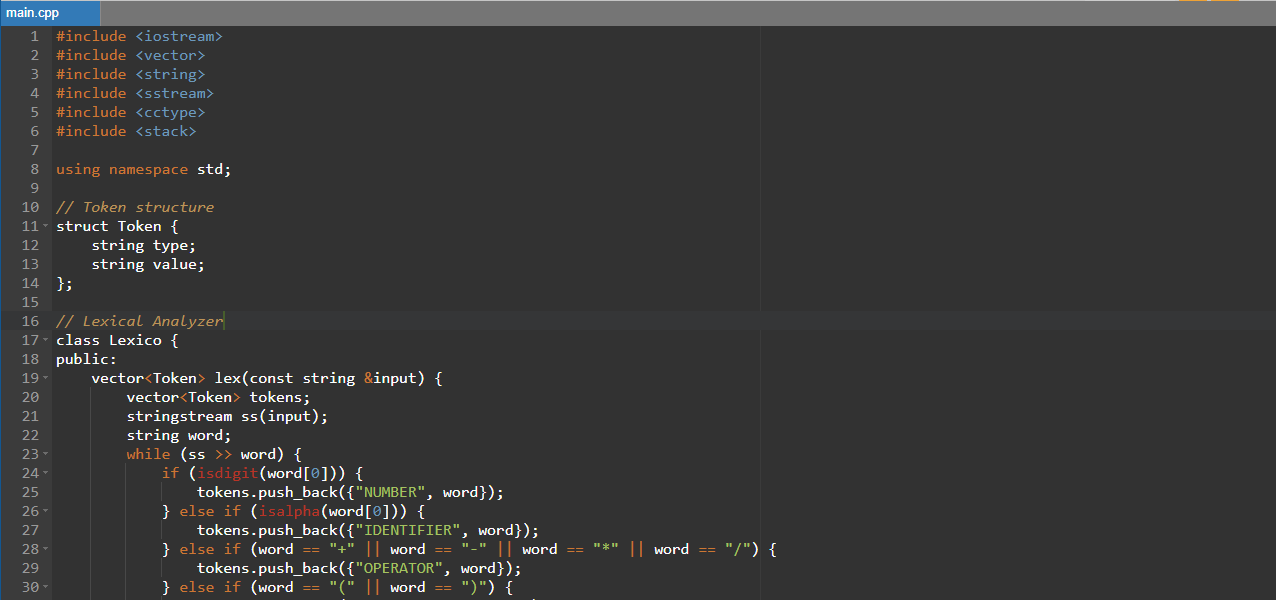
Screenshot2:



Screenshot3:



Screenshot of Code:



A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Q3,Q4

from bs4 import BeautifulSoup

# The HTML content from question 3

html\_content = '''

<!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>

'''

# Parse the HTML content using BeautifulSoup

soup = BeautifulSoup(html\_content, 'html.parser')

# Symbol table to store information about form elements

symbol\_table = []

# Extract form elements

form\_elements = soup.find\_all(['input', 'select'])

for element in form\_elements:

# Initialize a dictionary to store element details

element\_details = {

'tag': element.name,

'type': element.get('type', 'N/A'),

'id': element.get('id', 'N/A'),

'name': element.get('name', 'N/A'),

'attributes': element.attrs

}

# Add the element details to the symbol table

symbol\_table.append(element\_details)

# Print the symbol table

for entry in symbol\_table:

print(entry)

Screenshot:

