COMSATS Institute of Information Technology

Group Members:

M. Abdullah Arshad	(SP20-BCS-033)
Hasnain Ahmed	(FA20-BCS-005)

Class/Section: BCS-7 (A)

Subject: CC-Lab (Compiler Construction)

Submission To: Sir Bilal Haider

Date: 28-Dec-2023

CC-Lab Terminal:

Question 2: 2 functionalities along with Screen Shorts (Function Code + Output)

Answer:

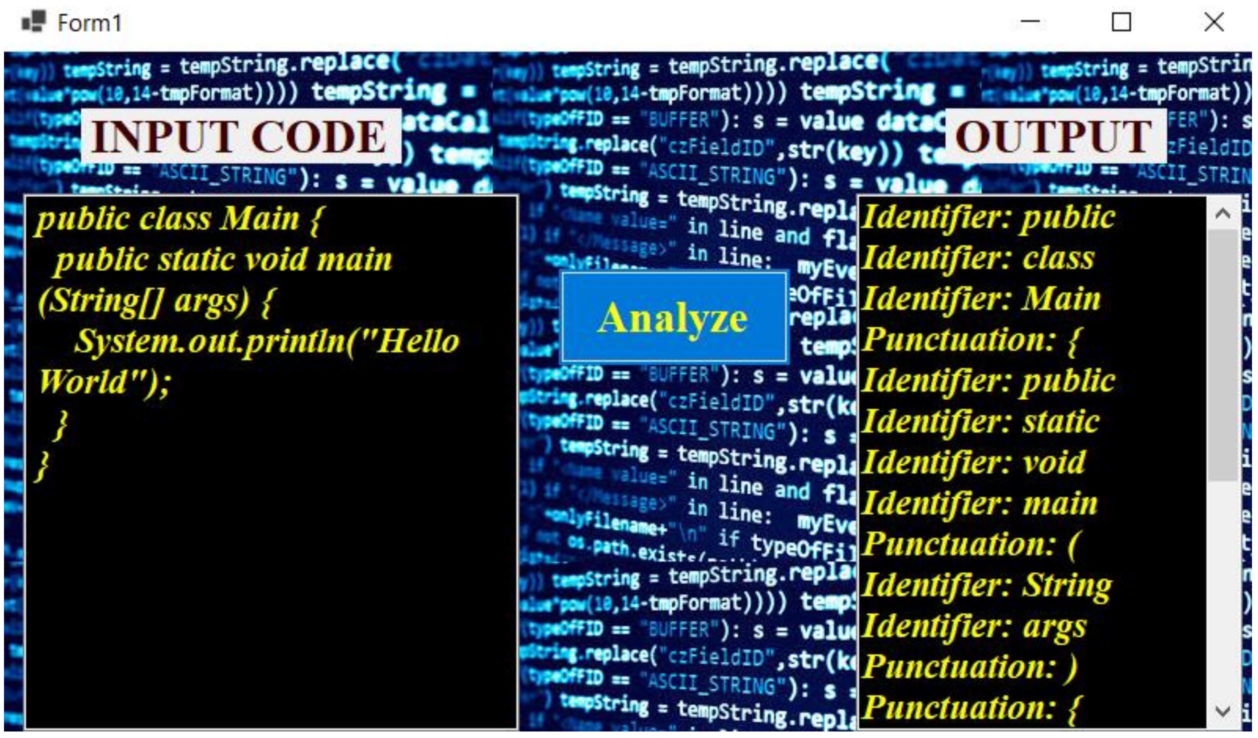
❖ Lexical Analyzer (Implemented in C# Visual Studio):

- **CODE:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Text.RegularExpressions;
namespace CC_TERMINAL_005_LEXICAL_ANALYZER_
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnAnalyze_Click(object sender, EventArgs e)
        {
            string input = txtInput.Text;
            List<Token> tokens = Analyze(input);

            // Display the tokens in the list box
            lstTokens.Items.Clear();
            foreach (Token token in tokens)
            {
                lstTokens.Items.Add(token);
            }

        }
        private List<Token> Analyze(string input)
        {
            List<Token> tokens = new List<Token>();

            // Define regular expressions for various token types
            var keywordRegex = new Regex(@"\b(if|else|while)\b");
            var identifierRegex = new Regex(@"\b[a-zA-Z_]\w*\b");
            var numberRegex = new Regex(@"\b\d+\b");
            var stringLiteralRegex = new Regex(@"""([^""\\]|\\.)*""");
            var operatorRegex = new Regex(@"[\+\-\*/=]");
            var punctuationRegex = new Regex(@"[{};,()]");
            var commentRegex = new Regex(@"\/\/.*|\/\*[\s\S]*?\*\/");
            var whitespaceRegex = new Regex(@"\s+");

            // Remove comments and whitespaces
            input = commentRegex.Replace(input, string.Empty);
            input = whitespaceRegex.Replace(input, " ");

            // Tokenize the input and keep track of positions
            var matches = keywordRegex.Matches(input);
            foreach (Match match in matches)
            {
                tokens.Add(new Token(match.Value, TokenType.Keyword, match.Index));
            }

            matches = identifierRegex.Matches(input);
            foreach (Match match in matches)
            {
                tokens.Add(new Token(match.Value, TokenType.Identifier, match.Index));
            }

            matches = numberRegex.Matches(input);
            foreach (Match match in matches)
            {
                tokens.Add(new Token(match.Value, TokenType.Number, match.Index));
            }

            matches = stringLiteralRegex.Matches(input);
            foreach (Match match in matches)
            {
                tokens.Add(new Token(match.Value, TokenType.StringLiteral, match.Index));
            }

            matches = operatorRegex.Matches(input);
            foreach (Match match in matches)
            {
                tokens.Add(new Token(match.Value, TokenType.Operator, match.Index));
            }

            matches = punctuationRegex.Matches(input);
            foreach (Match match in matches)
            {
                tokens.Add(new Token(match.Value, TokenType.Punctuation, match.Index));
            }

            // Sort tokens based on their positions
            tokens = tokens.OrderBy(t => t.Position).ToList();

            return tokens;
        }


    }
    public class Token
    {
        public string Lexeme { get; }
        public TokenType Type { get; }
        public int Position { get; }  // Added position
```

```
        public Token(string lexeme, TokenType type, int position)
        {
            Lexeme = lexeme;
            Type = type;
            Position = position;
        }

        public override string ToString()
        {
            return $"{Type}: {Lexeme}";
        }
    }

    public enum TokenType
    {
        Keyword,
        Identifier,
        Number,
        StringLiteral,
        Operator,
        Punctuation,
        Comment
    }
}
```

- **OUPUT:**



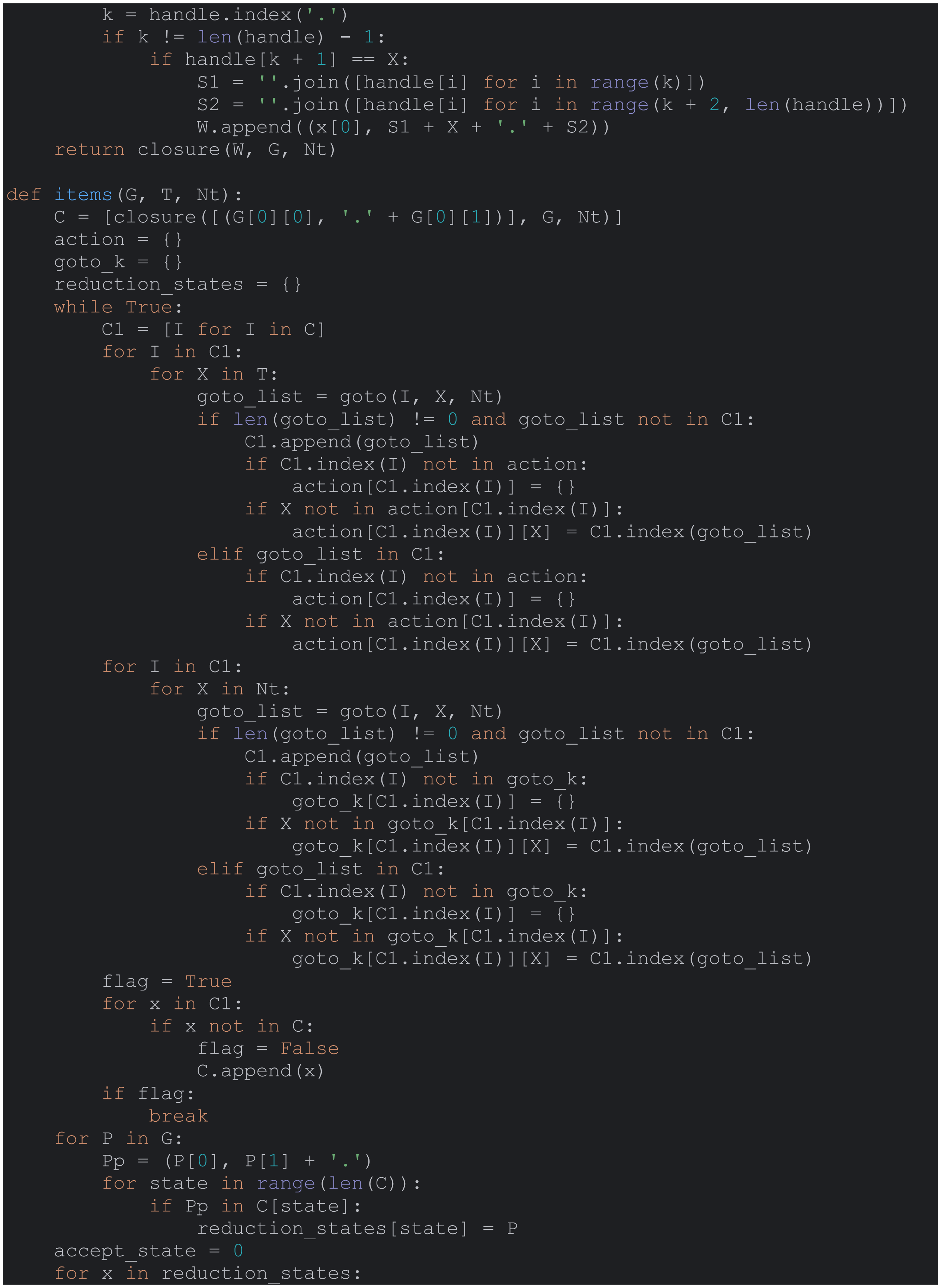
```
public class Main {
  public static void main
(String[] args) {
    System.out.println("Hello
World");
  }
}
```

```
Identifier: public
Identifier: class
Identifier: Main
Punctuation: {
Identifier: public
Identifier: static
Identifier: void
Identifier: main
Punctuation: (
Identifier: String
Identifier: args
Punctuation: )
Punctuation: {
```

❖ LR Parsing(Implemented in Python):

- **Code:**

```python
from pprint import pprint

def import_grammar(fileHandle):
    G, T, Nt = [], [], []
    for lines in fileHandle:
        production = lines.split(' -> ')
        if production[0] not in Nt:
            Nt.append(production[0])
        listStr = list(production[1])
        del listStr[-1]
        production[1] = ''.join(i for i in listStr)
        for char in production[1]:
            if 65 <= ord(char) <= 90:
                if char not in Nt:
                    Nt.append(char)
            else:
                if char not in T:
                    T.append(char)
        if production not in G:
            G.append((production[0], production[1]))
    T.append('#')
    return G, T, Nt

def closure(I, G, Nt):
    J = [p for p in I]
    while True:
        J1 = [x for x in J]
        for x in J1:
            handle = list(x[1])
            k = handle.index('.')
            if k + 1 != len(handle):
                if handle[k + 1] in Nt:
                    for p in G:
                        if p[0] == handle[k + 1]:
                            new_p = (p[0], '.' + p[1])
                            if new_p not in J1:
                                J1.append(new_p)
        flag = True
        for x in J1:
            if x not in J:
                flag = False
                J.append(x)
        if flag:
            break
    return J

def goto(I, X, Nt):
    W = []
    for x in I:
        handle = list(x[1])
```



```python
        k = handle.index('.')
        if k != len(handle) - 1:
            if handle[k + 1] == X:
                S1 = ''.join([handle[i] for i in range(k)])
                S2 = ''.join([handle[i] for i in range(k + 2, len(handle))])
                W.append((x[0], S1 + X + '.' + S2))
    return closure(W, G, Nt)

def items(G, T, Nt):
    C = [closure([(G[0][0], '.' + G[0][1])], G, Nt)]
    action = {}
    goto_k = {}
    reduction_states = {}
    while True:
        C1 = [I for I in C]
        for I in C1:
            for X in T:
                goto_list = goto(I, X, Nt)
                if len(goto_list) != 0 and goto_list not in C1:
                    C1.append(goto_list)
                    if C1.index(I) not in action:
                        action[C1.index(I)] = {}
                    if X not in action[C1.index(I)]:
                        action[C1.index(I)][X] = C1.index(goto_list)
                elif goto_list in C1:
                    if C1.index(I) not in action:
                        action[C1.index(I)] = {}
                    if X not in action[C1.index(I)]:
                        action[C1.index(I)][X] = C1.index(goto_list)
        for I in C1:
            for X in Nt:
                goto_list = goto(I, X, Nt)
                if len(goto_list) != 0 and goto_list not in C1:
                    C1.append(goto_list)
                    if C1.index(I) not in goto_k:
                        goto_k[C1.index(I)] = {}
                    if X not in goto_k[C1.index(I)]:
                        goto_k[C1.index(I)][X] = C1.index(goto_list)
                elif goto_list in C1:
                    if C1.index(I) not in goto_k:
                        goto_k[C1.index(I)] = {}
                    if X not in goto_k[C1.index(I)]:
                        goto_k[C1.index(I)][X] = C1.index(goto_list)
        flag = True
        for x in C1:
            if x not in C:
                flag = False
                C.append(x)
        if flag:
            break
    for P in G:
        Pp = (P[0], P[1] + '.')
        for state in range(len(C)):
            if Pp in C[state]:
                reduction_states[state] = P
    accept_state = 0
    for x in reduction_states:
        if reduction_states[x] == G[0]:
            accept_state = x
    return C, action, goto_k, reduction_states, accept_state

def parse_input_string(G, T, Nt, action_list, goto_list, reduction_states,
accept_state, input_str):
    stack = [0]
    i, top = 0, 0

    while True:
        s = stack[top]
        try:
            print(s, stack, input_str[i] if i != len(input_str) else
'Finish', end=' ')
            if s == accept_state:
                print('accept')
                break
            elif s in reduction_states:
                A, beta = reduction_states[s]
                print('reduce', A, '->', beta)
                for j in range(len(beta)):
                    del stack[top]
                t = stack[top]
                stack.insert(top, goto_list[t][A])
            else:
                a = input_str[i]
                stack.insert(top, action_list[s][a])
                print('shift', action_list[s][a])
                i = i + 1
        except Exception as e:
            print('Syntax error')
            break

if __name__ == "__main__":
    txt = input('Enter the name of the file: ') + '.txt'
    fileHandle = open(txt)
    G, T, Nt = import_grammar(fileHandle)
    print('Terminals:', T)
    print('Non-terminals:', Nt)
    C, action_list, goto_list, reduction_states, accept_state = items(G, T,
Nt)
    print('Action list')
    pprint(action_list)
    print('Goto list')
    pprint(goto_list)
    print('Reduction states')
    pprint(reduction_states)
    print('Accept state', accept_state)

    input_str = input('Enter some string: ') + '#'
    parse_input_string(G, T, Nt, action_list, goto_list, reduction_states,
accept_state, input_str)
```

- **Output**

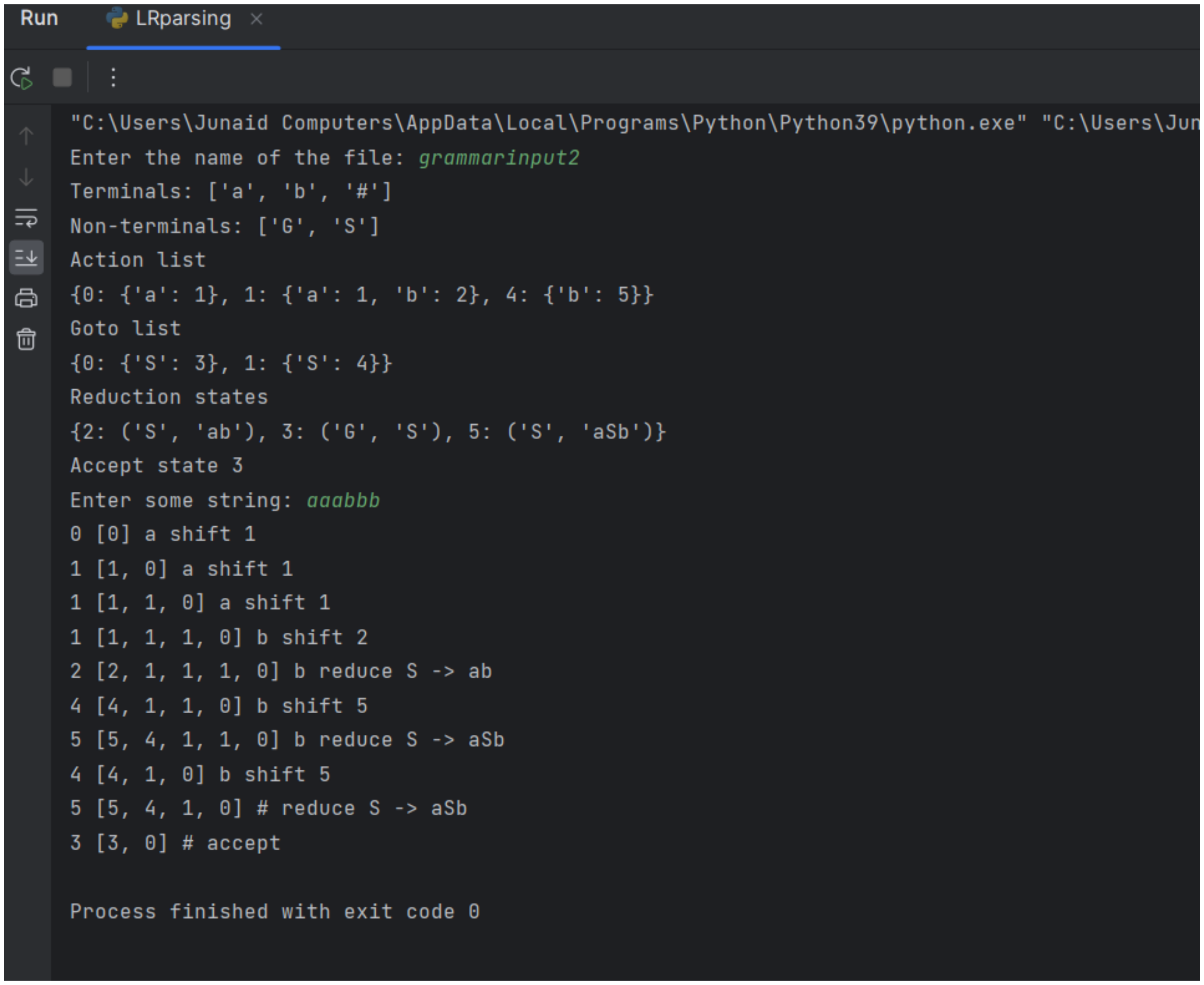
- **Given Grammar:**

G -> S

S -> aSb

S -> ab

- **Testing String : aaabbb**



```
"C:\Users\Junaid Computers\AppData\Local\Programs\Python\Python39\python.exe" "C:\Users\Jun
Enter the name of the file: grammarinput2
Terminals: ['a', 'b', '#']
Non-terminals: ['G', 'S']
Action list
{0: {'a': 1}, 1: {'a': 1, 'b': 2}, 4: {'b': 5}}
Goto list
{0: {'S': 3}, 1: {'S': 4}}
Reduction states
{2: ('S', 'ab'), 3: ('G', 'S'), 5: ('S', 'aSb')}
Accept state 3
Enter some string: aaabbb
0 [0] a shift 1
1 [1, 0] a shift 1
1 [1, 1, 0] a shift 1
1 [1, 1, 1, 0] b shift 2
2 [2, 1, 1, 1, 0] b reduce S -> ab
4 [4, 1, 1, 0] b shift 5
5 [5, 4, 1, 1, 0] b reduce S -> aSb
4 [4, 1, 0] b shift 5
5 [5, 4, 1, 0] # reduce S -> aSb
3 [3, 0] # accept

Process finished with exit code 0
```

-End