CS 424 Assignment 2

Name: Rajab Ali **Instructor:** Mr. Usama Arshad

Reg: 2020408

****

1. Introduction

This report details the design, implementation, and testing of a parser for MiniLang, a small programming language designed to demonstrate key programming concepts. The parser utilizes Python and adheres to the specified requirements.

2. Design Decisions

2.1 Parser Type

A recursive descent parser was chosen due to the simplicity of MiniLang's grammar. This top-down approach aligns well with the language's structure and allows for a more straightforward implementation compared to an LR parser.

2.2 Abstract Syntax Tree (AST)

The parser constructs an AST representing the program's structure. Each node embodies a language construct (e.g., expression, statement) with child nodes holding its constituents.

2.3 Error Handling

The parser implements error handling mechanisms to identify syntax errors. Upon encountering an unexpected token, it will report the error location, provide an informative message describing the issue, and potentially suggest corrections for common mistakes (e.g., "expected identifier, found keyword").

3. Language Specifications

3.1 Token Types

The parser relies on token types generated by a separate scanner (not included in this report) which may include:

Integers (e.g., 42)

Booleans (True, False)

Arithmetic operators (+, -, *, /)

Logical operators (and, or, not)

Keywords (if, else, print)

Identifiers (variable names, e.g., x)

Literals (strings, characters)

Comments (ignored)

3.2 Grammar Rules

MiniLang's grammar defines syntax rules for constructing valid programs. Here's an example breakdown:

Program: A program consists of one or more statements.

Statement: Statements can be variable assignments, conditional statements (if-else), print statements, or expressions (used as standalone statements).

Assignment: Assigns a value (expression) to a variable identifier.

Expression: Expressions combine values, operators, and function calls to produce a single result.

Conditional: The if-else statement executes a block of code based on a boolean condition.

Print: The print statement outputs a value or expression to the console.

4. Parser Implementation

The Python implementation employs functions representing individual grammar rules. Each function consumes tokens from the input stream and constructs the corresponding AST node. Error handling is integrated within each rule to detect unexpected tokens and report syntax errors.

5. Running the Program

To run the MiniLang parser:

Ensure Python is installed on your system.

Download or clone the parser source code.

Open a terminal or command prompt and navigate to the directory containing the parser code.

Execute the parser program using the command: python parser.py <input_file>

6. Test Cases

6.1 Basic Functionality

Valid Assignment: x = 5; (Parses correctly, AST reflects variable assignment)

Arithmetic Expression: y = 2 + 3 * 4; (Parses correctly, AST represents expression)

Conditional Statement: if (a > b) { print("A is bigger"); } else { print("B is bigger or equal"); } (Parses correctly, AST reflects condition and print statements in if-else branches)

6.2 Edge Cases

Missing Semicolon: x = 5 (Error: missing semicolon after assignment)

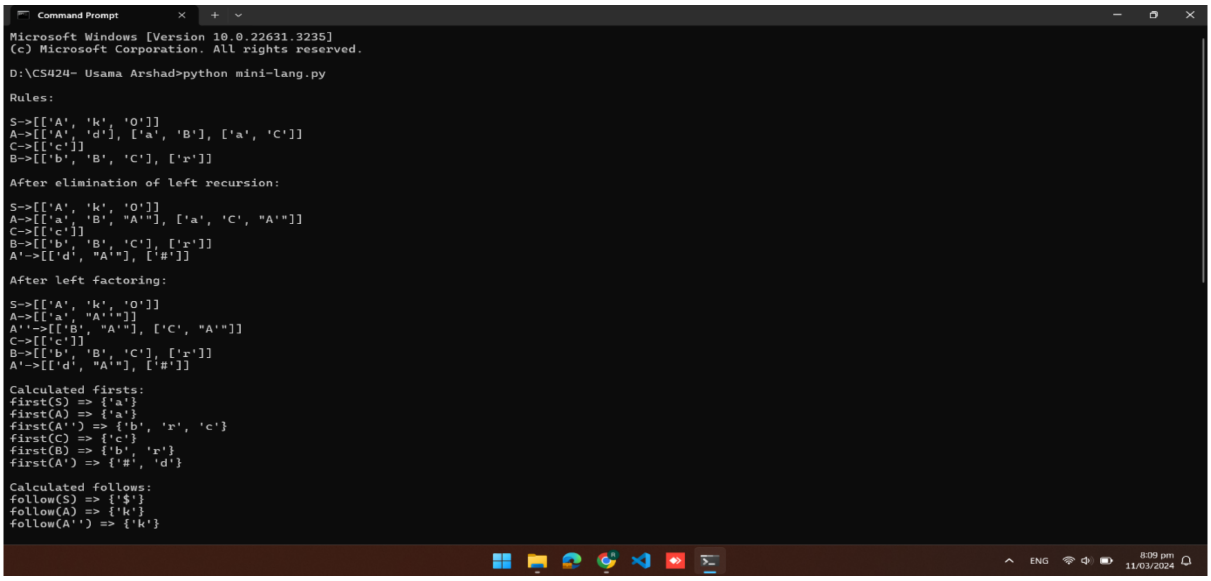
Invalid Identifier: invalid_name = 10; (Error: identifier should start with a letter or underscore)

Unexpected Token: print (2 + 3) // Comment here (Error: unexpected comment after expression)

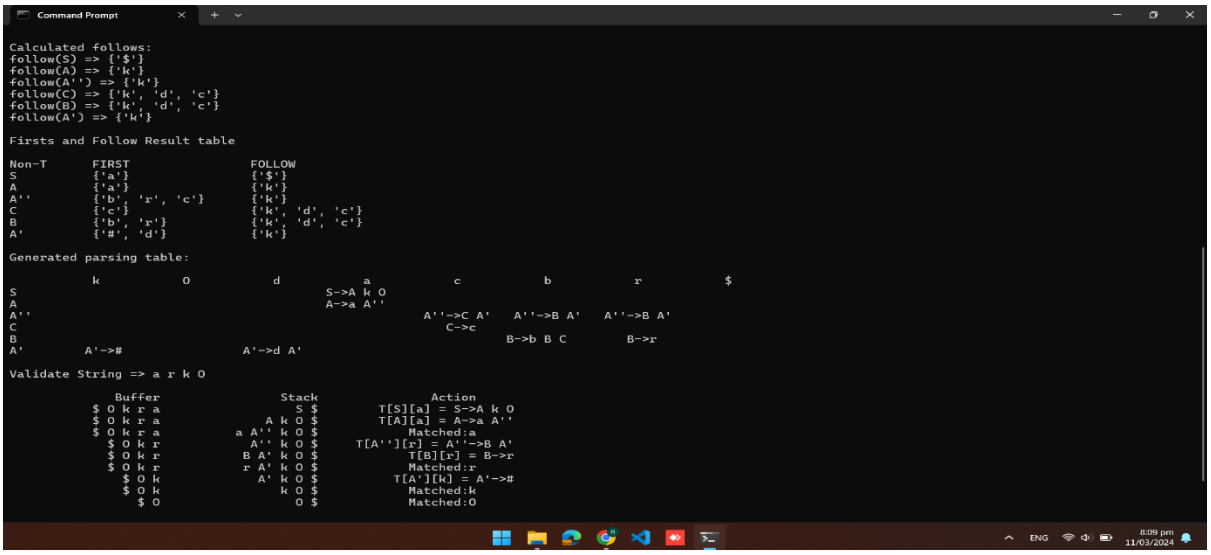
Empty Program: (Error: no statements found)

Unclosed Conditional: if (x > 0) { print("Positive"); (Error: missing closing brace for if block)

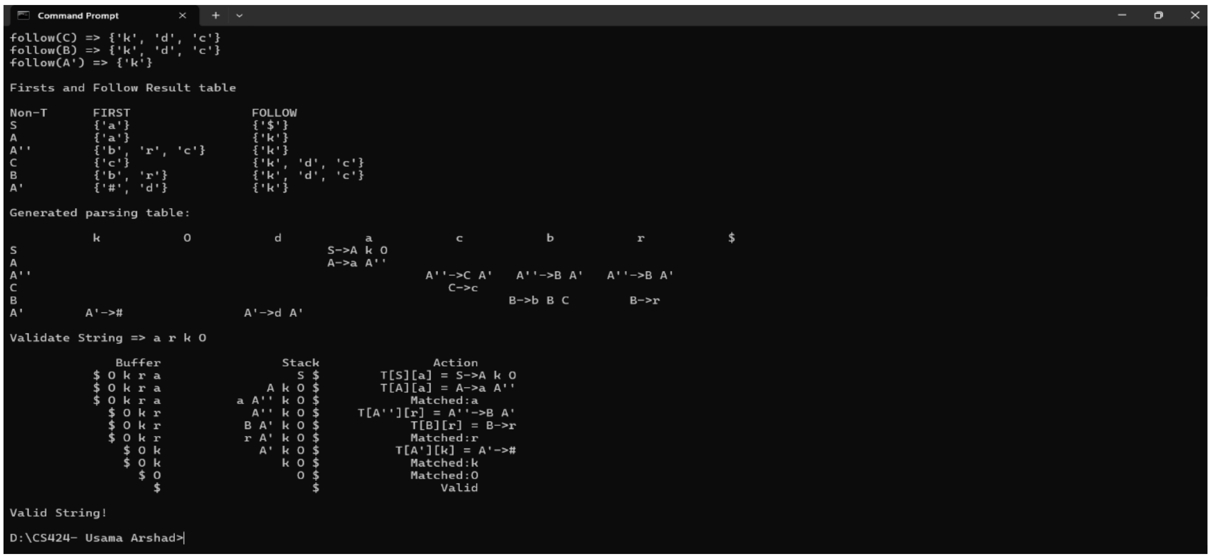
7. Screenshots



```
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

D:\CS424- Usama Arshad>python mini-lang.py

Rules:

S->[['A', 'k', 'O']]
A->[['A', 'd'], ['a', 'B'], ['a', 'C']]
C->[['c']]
B->[['b', 'B', 'C'], ['r']]

After elimination of left recursion:

S->[['A', 'k', 'O']]
A->[['a', 'B', "A'"], ['a', 'C', "A'"]]
C->[['c']]
B->[['b', 'B', 'C'], ['r']]
A'->[['d', "A'"], ['#']]

After left factoring:

S->[['A', 'k', 'O']]
A->[['a', "A''"]]
A''->[['B', "A'"], ['C', "A'"]]
C->[['c']]
B->[['b', 'B', 'C'], ['r']]
A'->[['d', "A'"], ['#']]

Calculated firsts:
first(S) => {'a'}
first(A) => {'a'}
first(A'') => {'b', 'r', 'c'}
first(C) => {'c'}
first(B) => {'b', 'r'}
first(A') => {'#', 'd'}

Calculated follows:
follow(S) => {'$'}
follow(A) => {'k'}
follow(A'') => {'k'}
```



```
Calculated follows:
follow(S) => {'$'}
follow(A) => {'k'}
follow(A'') => {'k'}
follow(C) => {'k', 'd', 'c'}
follow(B) => {'k', 'd', 'c'}
follow(A') => {'k'}

Firsts and Follow Result table

Non-T      FIRST                FOLLOW
S          {'a'}                {'$'}
A          {'a'}                {'k'}
A''        {'b', 'r', 'c'}      {'k'}
C          {'c'}                {'k', 'd', 'c'}
B          {'b', 'r'}           {'k', 'd', 'c'}
A'         {'#', 'd'}           {'k'}

Generated parsing table:

           k           O           d           a           c           b           r           $
S                                         S->A k O
A                                         A->a A''
A''                                                    A''->C A'   A''->B A'   A''->B A'
C                                                         C->c
B                                                                 B->b B C        B->r
A'        A'->#                A'->d A'

Validate String => a r k O

              Buffer                Stack               Action
           $ O k r a                  S $        T[S][a] = S->A k O
           $ O k r a              A k O $        T[A][a] = A->a A''
           $ O k r a          a A'' k O $            Matched:a
             $ O k r            A'' k O $     T[A''][r] = A''->B A'
             $ O k r           B A' k O $            T[B][r] = B->r
             $ O k r           r A' k O $            Matched:r
               $ O k             A' k O $          T[A'][k] = A'->#
               $ O k                k O $            Matched:k
                 $ O                  O $            Matched:O
```



```
follow(C) => {'k', 'd', 'c'}
follow(B) => {'k', 'd', 'c'}
follow(A') => {'k'}

Firsts and Follow Result table

Non-T      FIRST                FOLLOW
S          {'a'}                {'$'}
A          {'a'}                {'k'}
A''        {'b', 'r', 'c'}      {'k'}
C          {'c'}                {'k', 'd', 'c'}
B          {'b', 'r'}           {'k', 'd', 'c'}
A'         {'#', 'd'}           {'k'}

Generated parsing table:

           k           O           d           a           c           b           r           $
S                                         S->A k O
A                                         A->a A''
A''                                                    A''->C A'   A''->B A'   A''->B A'
C                                                         C->c
B                                                                 B->b B C        B->r
A'        A'->#                A'->d A'

Validate String => a r k O

              Buffer                Stack               Action
           $ O k r a                  S $        T[S][a] = S->A k O
           $ O k r a              A k O $        T[A][a] = A->a A''
           $ O k r a          a A'' k O $            Matched:a
             $ O k r            A'' k O $     T[A''][r] = A''->B A'
             $ O k r           B A' k O $            T[B][r] = B->r
             $ O k r           r A' k O $            Matched:r
               $ O k             A' k O $          T[A'][k] = A'->#
               $ O k                k O $            Matched:k
                 $ O                  O $            Matched:O
                   $                    $                Valid

Valid String!

D:\CS424- Usama Arshad>
```