/*
==
========
PITCH++ COMPILER - DOCUMENTATION
Compiler Construction - Phase 1
Lexical Analyzer Implementation
==
========

Author: Abdullah Butt
Registration ID: L1F22BSCS0497
Course: Compiler Construction
Assignment: Phase 1 - Lexical Analysis using Flex

==
========
TABLE OF CONTENTS
==
========

==
========
1. INTRODUCTION AND THEME
==
========

Language Name: Pitch++
Theme: Cricket-Inspired C++ Programming Language

EXPLANATION:

Pitch++ is designed for cricket lovers to make programming more engaging.
Each keyword represents a cricket term that relates conceptually to
programming actions. For example:

- OVER is used as "for" loop (overs in cricket)
- PLAY is used as main function (playing the match)
- RETIRE is used as "return" (batsman retiring)

The purpose is to teach programming logic through cricket analogies,
making coding more relatable and fun for people familiar with the game.

==
========

2. REGULAR EXPRESSION DEFINITIONS

TABLE OF REGEX DEFINITIONS:

Token Type	Regular Expression	Description
IDENTIFIER	[A-Z]([A-Z]|[0-9])*	Uppercase letters start, then letters/digits
INTEGER	[+-]?[0-9]+	Optional sign, then digits
FLOAT	[+-]?[0-9]*\.[0-9]+	Optional sign, optional digits, decimal, digits
NUMBER	INTEGER | FLOAT	Either integer or float
STRING	"([^"\n]|\\")*"	Double-quoted with escapes
CHAR	'([^'\n]|\\')	Single-quoted with escapes
WHITESPACE	[\t]+	Spaces and tabs (ignored)
NEWLINE	\n	Line break (line tracking)
SINGLE_COMMENT	//.*	Single-line comments
MULTI_COMMENT	/*([^*]|*+[^*/])**+/	Multi-line comments

+------------------+--------------------------------+--------------------
--------+

==========
========
3. KEYWORDS EXPLANATION
==========
========

Total Keywords: 15 (All Cricket-Themed)

No.	Keyword	Equivalent	Explanation & Reason
1	OPENER	int	An opener is a batsman who starts the innings. Similarly, int is a fundamental data type. Openers are solid and dependable - like integers.
2	ALLROUNDER	float	An all-rounder is versatile (bat & bowl). Floats are versatile numbers with decimal precision.
3	PLAY	main	The main function is where the program "plays" or executes. Just like players PLAY the match.
4	LEGALBALL	true	A legal ball in cricket is a valid delivery. Represents boolean true.
5	NOBALL	false	A no-ball is an illegal delivery. Represents boolean false.

6	RETIRE	return	When a batsman retires, they leave the field. Return exits a function.
7	OUT	break	When a batsman is OUT, the innings stops for them. Break exits a loop.
8	NOTOUT	continue	When a batsman is NOT OUT, they continue batting. Continue statement continues to next iteration.
9	TOSS	if	Before a match, captains TOSS a coin to make decisions. 'if' makes conditional decisions.
10	UMPIRE	else	The umpire makes the alternative decision when the initial doesn't apply. Like 'else'.
11	INNING	while	An inning is a period of play that continues while conditions are met. While loop continues based on cond.
12	OVER	for	An over consists of 6 balls bowled in sequence. 'for' loop iterates a specific number of times.

13	RUN	cout/print	Scoring runs is the output of batting. RUN outputs program data.
14	TEAM	class	A team is a group/class of players. Classes group related data/methods.
15	TEXTBOOK	string	"Textbook shot" means perfect shot. Strings are text - hence TEXTBOOK.

WHY THESE KEYWORDS?

These keywords were carefully chosen to create meaningful associations between
cricket and programming:

1. ACTION-BASED: Keywords like PLAY, RUN, TOSS represent actions in both cricket and programming.

2. PLAYER-BASED: OPENER, ALLROUNDER represent different types/roles, just like data types have different roles.

3. EVENT-BASED: OUT, NOTOUT, RETIRE represent events that control flow, matching control flow statements.

4. STRUCTURE-BASED: INNING, OVER, TEAM represent structural elements.

This makes the language intuitive and memorable for cricket enthusiasts!

4. OPERATORS AND PUNCTUATIONS

A. OPERATORS (13 Total):

Operator	Description	Example Usage
+	Addition	RUNS = RUNS + 4~
-	Subtraction	RUNS = TARGET - RUNS~
*	Multiplication	TOTAL = OVERS * 6~
/	Division	AVG = RUNS / OVERS~

===	Equality	TOSS(RUNS === 100)
!==	Not Equal	TOSS(RUNS !== 0)
=	Assignment	RUNS = 10~
+++	Increment	WICKETS+++~
---	Decrement	BALLS---~
<	Less Than	TOSS(RUNS < TARGET)
>	Greater Than	TOSS(RUNS > 50)
<=	Less or Equal	TOSS(RUNS <= 100)
>=	Greater or Equal	TOSS(WICKETS >= 10)

REASON FOR CHOOSING:
- Standard arithmetic operators (+, -, *, /) are universal
- Used "===" instead of "==" to make it distinct and memorable
- Used "+++" and "---" (triple) to fit cricket theme (boundaries!)
- Comparison operators are essential for conditional logic

B. PUNCTUATIONS (7 Total):

Symbol	Description	Example Usage
~	Statement terminator	OPENER runs = 0~
{	Code block start	PLAY() {
}	Code block end	}
(	Parameters start	PLAY()
)	Parameters end	TOSS(runs > 0)
,	Separator	RUN(a, b, c)~
;	Alt terminator	OPENER x = 5;

REASON FOR CHOOSING:
- "~" as statement terminator is unique and cricket-themed (like a wave!)
- Curly braces { } are standard for code blocks
- Parentheses ( ) are standard for functions and expressions
- Comma for separation is universal
- Semicolon as alternative for flexibility

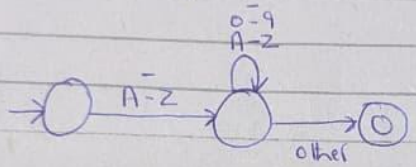
5. FINITE AUTOMATA

Capital Allowed)

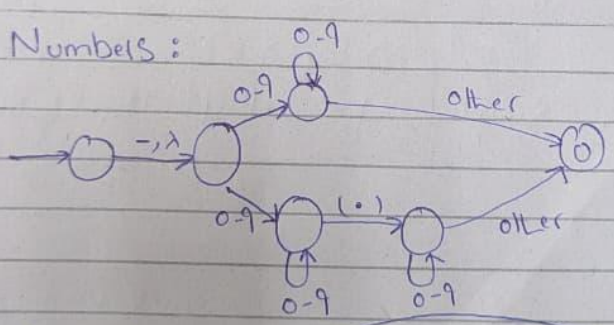
TEAM 1 TEXTBOOK

5. FINITE AUTOMATA

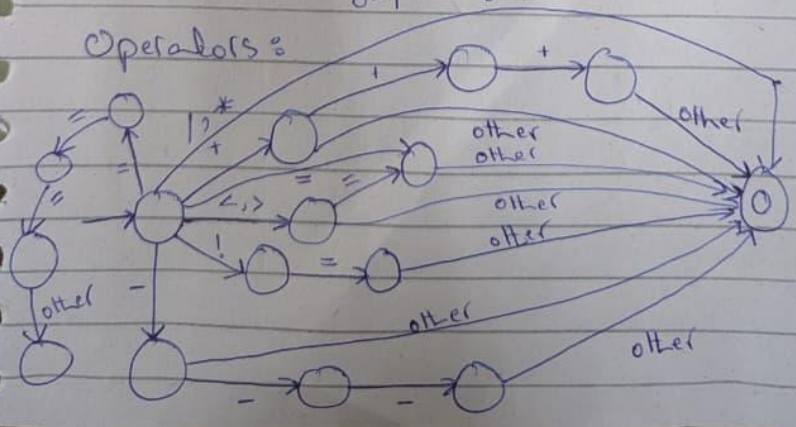
Identifiers :



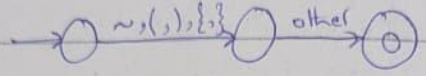
Numbers :



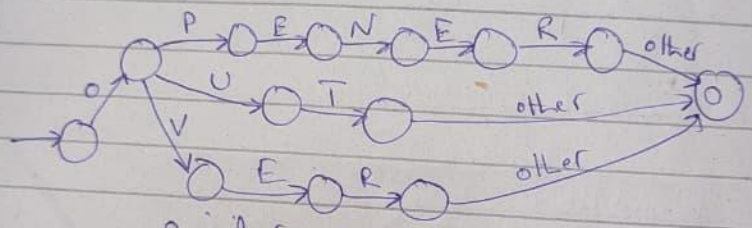
Operators :



Punctuators :



Keywords :



similar for other id keywords

b. Explanation :

Click it

==
========

6. IMPLEMENTATION DETAILS

==
========

LEXICAL ANALYZER STRUCTURE:

1. TOKEN RECOGNITION:
   - Uses Flex pattern matching
   - Longest match principle
   - Keywords matched before identifiers (priority)

2. LINE NUMBER TRACKING:
   - Global variable 'line_num' initialized to 1
   - Incremented on every newline character
   - Reported with every token and error

3. ERROR HANDLING:
   - Invalid tokens caught by catch-all pattern
   - Lowercase identifiers rejected
   - Invalid characters reported
   - Identifiers starting with numbers rejected

4. OUTPUT FORMAT:
   - Format: "Line <n>: <TOKEN_TYPE> → <lexeme>"
   - Tokens written to: tokens.txt
   - Errors written to: errors.log

5. COMMENTS:
   - Single-line comments: // ... (ignored)
   - Multi-line comments: /* ... */ (ignored, but newlines counted)

==
========

7. SAMPLE TEST PROGRAM

==
========

File: test_program.pitch

```
// Pitch++ Sample Program
// Cricket Match Simulator

PLAY() {
    // Initialize match variables
    OPENER TOTALRUNS = 0~
    OPENER TARGET = 150~
    OPENER WICKETS = 0~
    ALLROUNDER RUNRATE = 0.0~
    TEXTBOOK TEAMNAME = "PAKISTAN"~
    OPENER BALLSPLAYED = 0~
```

```
    // Start the match
    RUN("Cricket Match Started!")~
    RUN(TEAMNAME)~

    // First innings - batting loop
    OVER(OPENER I = 0~ I < 20~ I+++) {
        BALLSPLAYED = BALLSPLAYED + 6~

        TOSS(WICKETS < 10) {
            OPENER RUNSSCORED = 8~
            TOTALRUNS = TOTALRUNS + RUNSSCORED~

            TOSS(RUNSSCORED === 0) {
                WICKETS+++~
                RUN("Wicket fallen!")~
            }

            UMPIRE {
                RUN("Runs scored")~
            }
        }

        TOSS(WICKETS === 10) {
            OUT~
        }
    }

    // Calculate run rate
    RUNRATE = TOTALRUNS / 20.0~

    // Check if target achieved
    TOSS(TOTALRUNS >= TARGET) {
        RUN("Target achieved!")~
    }
    UMPIRE {
        TOSS(TOTALRUNS < TARGET) {
            RUN("Target not achieved!")~
        }
    }

    // Display final score
    RUN("Match Complete!")~
    RUN(TOTALRUNS)~

    RETIRE 0~
}

// Team class definition
TEAM CRICKET {
    OPENER PLAYERS = 11~
    TEXTBOOK CAPTAIN = "BABAR"~
    ALLROUNDER AVERAGE = 45.5~
}
```

```
================================================================================
========
                          8. EXPECTED OUTPUT (tokens.txt)
================================================================================
========

=== Pitch++ Lexical Analyzer Output ===
Input File: test_program.pitch

Line 5: KEYWORD → PLAY
Line 5: PUNCTUATION → (
Line 5: PUNCTUATION → )
Line 5: PUNCTUATION → {
Line 7: KEYWORD → OPENER
Line 7: IDENTIFIER → TOTALRUNS
Line 7: OPERATOR → =
Line 7: NUMBER → 0
Line 7: PUNCTUATION → ~
Line 8: KEYWORD → OPENER
Line 8: IDENTIFIER → TARGET
Line 8: OPERATOR → =
Line 8: NUMBER → 150
Line 8: PUNCTUATION → ~
...
(continues for all tokens)

=== Analysis Complete ===
Total lines: 75
Errors: 0


================================================================================
========
                          9. ERROR HANDLING EXAMPLES
================================================================================
========

TEST CASE 1: Lowercase Identifier

Input:
    opener score = 10~

Output (errors.log):
    Line 1: ERROR → opener (invalid token)

Reason: Identifiers must start with uppercase letters only.


TEST CASE 2: Identifier Starting with Digit

Input:
    OPENER 123ABC = 5~
```

```
Output (errors.log):
    Line 1: ERROR → 123 (invalid token)

Reason: Identifiers cannot start with digits.


TEST CASE 3: Invalid Special Characters

Input:
    OPENER SCORE@ = 10~

Output (errors.log):
    Line 1: ERROR → @ (invalid token)

Reason: @ is not a valid operator or punctuation in Pitch++.


TEST CASE 4: Unclosed String

Input:
    TEXTBOOK NAME = "Pakistan~

Output (errors.log):
    Line 1: ERROR → " (invalid token)

Reason: String must be closed with matching quote.


================================================================================
                              END OF DOCUMENTATION
================================================================================

For more information, refer to:
- README.md (detailed guide)
- scanner.l (implementation)
- test_program.pitch (example code)

Contact: Abdullah Butt (L1F22BSCS0497)
Course: Compiler Construction - Phase 1
Date: November 2025

================================================================================
*/
```