

Compiler Construction Project – Phase 1: Lexical Analyzer

By Nauman Irshad ali Shah

Refer No. : L1f22bscs0122

youtube video link ::: <https://youtu.be/2bxd5YWMU80?si=Ffhw70ql2J-Owe26>

github link ::: <https://github.com/Nauman-Irshad/cc-project-Nauman-irshad-0122/blob/main/video%20cc%20project%20working%20nauman%200122.mp4>

Objective:

In Phase 1, I designed and implemented a lexical analyzer for a Mini C++-like language using Flex, which I called the Urdu Script Lexical Analyzer. I used my own Urdu-style keywords such as:

hum, tum, Wapas, Agar, Adadi, Mantiqi, shuru, khatam, jabtak, warna, magar, wo, socho, dalo, rahko, pakka, aur, keliye, lo, sahi, galat, ya, nahi, kaam, wapis, hatao

Features of My Lexical Analyzer:

1. TokenRecognition:

My analyzer identifies and classifies **all types of tokens**:

- **Identifiers:** Variables and function names (regex: `[a-zA-Z_][a-zA-Z0-9_]*`)
- **Numbers:** Integers (`[+-]?[0-9]+`), floats (`[+-]?[0-9]*\.[0-9]+`), and exponential forms (`[0-9]+([Ee][+-]?[0-9]+)`)
- **Keywords:** My 25 unique Urdu-style keywords (regex: exact string match for each keyword)
- **Operators:** Includes `+`, `-`, `*`, `/`, `%`, `=`, `+=`, `==`, etc. (regex: exact match for each operator)
- **Punctuations:** `{`, `}`, `(`, `)`, `;`, `::` (regex: exact match for each punctuation)
- **Strings:** Enclosed in double quotes (regex: `"[^\"\\n]*"`)

2. Error Handling:

Invalid tokens are detected and reported with descriptive error messages. For example:

- Line 7: ERROR → `@salary` (invalid identifier)
- Line 15: ERROR → `12ab` (invalid number format)
- Line 20: ERROR → `'abc'` (character literal too long)
- Line 35: INVALID_IDENTIFIER → `#third` (ERROR: identifier cannot start with special character)
- Line 35: INVALID_IDENTIFIER → `$price` (ERROR: identifier cannot start with special character)
- Line 35: INVALID_IDENTIFIER → `%rate` (ERROR: identifier cannot start with special character)
- Line 35: INVALID_IDENTIFIER → `^exp` (ERROR: identifier cannot start with special character)
- Line 35: INVALID_IDENTIFIER → `&mem` (ERROR: identifier cannot start with

- special character)
- Line 35: INVALID_IDENTIFIER -> *data (ERROR: identifier cannot start with special character)

3. Uniqueness:

- I designed **25 unique keywords**, inspired by Urdu terms that map to programming constructs (shuru for start, khatam for end, Mantiqi for logical operations).
- I selected **operators** (+, =, ++) and **punctuations** ({, ::) that are essential for expression parsing.
- I wrote a **unique test program** of over **220** lines using these keywords, ensuring originality.

4. Regex Implementation:

- Keywords: Exact string match for each keyword in the keywords[] array.
- Identifiers: [a-zA-Z_][a-zA-Z0-9_]*
- Integers: [+]?[0-9]+
- Floats: [+]?[0-9]*\.[0-9]+
- Exponentials: [0-9]+([Ee][+]?[0-9]+)
- Operators and punctuations: Exact string matches for each symbol.
- Strings: "[^\"\\n]*"

urdu_lexer.l - Notepad

File Edit Format View Help

```
%{
#include <stdio.h>
#include <string.h>
#include <ctype.h>

FILE *token_file;
FILE *error_file;
int line_no = 1;
int total_tokens = 0;
int total_errors = 0;

int in_comment = 0;

int is_keyword(const char *s) {
    const char *keywords[] = {
        "hum", "tum", "Wapas", "Agar", "Adadi", "Mantiqi", "shuru", "khatam",
        "jabtak", "warna", "magar", "wo", "socho", "dalo", "rahko", "pakka",
        "aur", "keliye", "lo", "sahi", "galat", "ya", "nahi", "kaam", "wapis", "hatao",
        "output<-", NULL
    };
    for (int i = 0; keywords[i] != NULL; i++)
        if (strcmp(s, keywords[i]) == 0)
            return 1;
    return 0;
}

const char* get_operator_name(const char *s) {
    if (strcmp(s, "+") == 0) return "PLUS";
    if (strcmp(s, "-") == 0) return "MINUS";
    if (strcmp(s, "*") == 0) return "MULTIPLY";
    if (strcmp(s, "/") == 0) return "DIVIDE";
    if (strcmp(s, "%") == 0) return "MODULO";
    if (strcmp(s, "++") == 0) return "INCREMENT";
}
```

Project Description

1. Tokenization

Identifiers:

#	Identifier	Valid/Invalid	Notes / Reason	Regex Check (C++ Identifier)
1	valid123	Valid	Letters + digits	[a-zA-Z_][a-zA-Z0-9_]*
2	abc123def	Valid	Letters + digits	[a-zA-Z_][a-zA-Z0-9_]*
3	123start	Invalid	Starts with digit	[a-zA-Z_][a-zA-Z0-9_]*
4	@begin	Invalid	Starts with special character @	[a-zA-Z_][a-zA-Z0-9_]*
5	variable123	Valid	Letters + digits	[a-zA-Z_][a-zA-Z0-9_]*
6	file_path	Valid	Underscores allowed	[a-zA-Z_][a-zA-Z0-9_]*
7	hello-world	Invalid	Contains hyphen -	[a-zA-Z_][a-zA-Z0-9_]*
8	param1	Valid	Simple parameter	[a-zA-Z_][a-zA-Z0-9_]*
9	\$variable	Invalid	Starts with \$	[a-zA-Z_][a-zA-Z0-9_]*
10	totalCount	Valid	CamelCase	[a-zA-Z_][a-zA-Z0-9_]*
11	!invalid	Invalid	Starts with !	[a-zA-Z_][a-zA-Z0-9_]*
12	counter	Valid	Simple word	[a-zA-Z_][a-zA-Z0-9_]*
13	space here	Invalid	Contains space	[a-zA-Z_][a-zA-Z0-9_]*
14	avgScore	Valid	CamelCase	[a-zA-Z_][a-zA-Z0-9_]*

- **Numbers (integers, floats, exponential forms):**

#	Number Type	Regex Pattern	Example	Description
1	Integer	<code>[0-9]+</code>	123, 0, 456	Whole numbers without decimal point
2	Floating-point	<code>[0-9]*\.[0-9]+</code>	3.14, 0.5, .25	Numbers with decimal point
3	Exponential form	<code>[0-9]+(\.[0-9]+)?[Ee][+-]?[0-9]+</code>	1.5E3, 2e-4	Numbers in scientific notation

- **Keywords : (24 Keywords)**

Keyword	Meaning	Example in Urdu -Script -like Urdu Script
hum	Declare a variable / start	hum x = 5;
tum	Input variable	tum y;
Wapas	Return statement	Wapas x;
Agar	If condition	Agar (x > 0) { ... }
Adadi	Numeric type (int/float)	Adadi num = 10;
Mantiqi	Logical operator / condition	Mantiqi result = (x && y);
shuru	Start block / function	shuru { ... }
khatam	End block / function	khatam
jabtak	While loop	jabtak (x < 10) { ... }
warna	Else / alternative	warna { ... }
magar	But / exception	magar (x != 0) { ... }
wo	Pointer / reference	wo ptr = &x;
socho	Read / input statement	socho x;
dalo	Assign / put value	dalo x = 20;
rahko	Keep / store	rahko y = x;
pakka	Constant / final	pakka pi = 3.14;
aur	And operator	aur (x && y)
keliye	For loop	keliye (i=0; i<5; i++) { ... }
lo	Take / fetch value	lo x = num;
sahi	True / correct	Mantiqi flag = sahi;
galat	False / incorrect	Mantiqi flag = galat;
ya	Or operator	ya (x
nahi	Not operator	nahi (x)
kaam	Function call / operation	kaam doSomething();

• Operators:(20 Operators)

Operator	Type / Notes
+	PLUS
-	MINUS
*	MULTIPLY
/	DIVIDE
%	MODULO
++	INCREMENT
--	DECREMENT
=	ASSIGN
+=	PLUS_ASSIGN
-=	MINUS_ASSIGN
*=	MULTIPLY_ASSIGN
/=	DIVIDE_ASSIGN
%=	MODULO_ASSIGN
<<=	LEFT_SHIFT_ASSIGN
>>=	RIGHT_SHIFT_ASSIGN
&=	BITWISE_AND_ASSIGN
^=	BITWISE_XOR_ASSIGN
'	'
==	EQUAL
!=	NOT_EQUAL
>	GREATER
<	LESS
>=	GREATER_EQUAL
<=	LESS_EQUAL

• Punctuations:(11 Punctuations)

S.No	Punctuation	Token Type
1	;	SEMICOLON
2	,	COMMA
3	(LEFT_PAREN
4)	RIGHT_PAREN
5	{	LEFT_BRACE
6	}	RIGHT_BRACE
7	[LEFT_BRACKET
8]	RIGHT_BRACKET
9	:	COLON
10	?	QUESTION

• Strings and Characters:

S.No	Pattern / Lexeme	Token Type	Notes / Description
1	"[^"\n]*"	STRING	Valid string literal enclosed in double quotes
2	"[^"\n]*	INVALID_STRING	Unterminated string literal
3	'[^'\n]'	CHAR	Valid character literal enclosed in single quotes
4	'\\[nrt0abfv\\'\"]'	CHAR	Valid escaped character
5	'\\[^nrt0abfv\\'\"]'	INVALID_CHAR	Invalid escape sequence in character literal
6	'[^'\n]*'	INVALID_CHAR	Character literal too long

Each token must be printed in the format:

Line <n>: <TOKEN_TYPE> → <lexeme>

My Examples:

- Line 2: KEYWORD → Adadi
- Line 2: IDENTIFIER → kya_yeh_kalima
- Line 2: LEFT_PAREN → (
- Line 2: IDENTIFIER → const
- Line 2: IDENTIFIER → char
- Line 2: RIGHT_PAREN →)
- Line 2: LEFT_BRACE → {
- Line 3: IDENTIFIER → const
- Line 3: IDENTIFIER → char
- Line 3: LEFT_BRACKET → [

• Line Tracking

Report the line number for each token

- Line 19: SEMICOLON → ;
- Line 20: RIGHT_BRACE → }
- Line 22: KEYWORD → Adadi
- Line 22: KEYWORD → shuru
- Line 22: LEFT_PAREN → (
- Line 22: RIGHT_PAREN →)
- Line 22: RIGHT_PAREN →)

1. Error Handling

- Invalid tokens must be reported as errors with their line numbers.
- ❖ Line 2: INVALID_IDENTIFIER -> ***lafz** (ERROR: identifier cannot start with special character)
- ❖ Line 3: INVALID_IDENTIFIER -> ***kalimat** (ERROR: identifier cannot start with special character)
- ❖ Line 23: INVALID_NUMBER_MULTIPLE_EXPONENTS -> "Roman Urdu Keyword Check Shuru Hua\\n" (ERROR: invalid number - multiple exponents)
- ❖ Line 33: INVALID_NUMBER_FORMAT -> 123start (ERROR: identifier cannot start with digit)
- ❖ Line 33: INVALID_IDENTIFIER -> @begin (ERROR: identifier cannot start with special character)
- ❖ Line 33: INVALID_IDENTIFIER -> #first (ERROR: identifier cannot start with special character)
- ❖ Line 33: INVALID_IDENTIFIER -> \$variable (ERROR: identifier cannot start with special character)
- ❖ Line 33: INVALID_IDENTIFIER -> %percent (ERROR: identifier cannot start with special character)
- ❖ Line 33: INVALID_IDENTIFIER -> ^caret (ERROR: identifier cannot start with special character)
- ❖ Line 33: INVALID_IDENTIFIER -> &ref (ERROR: identifier cannot start with special character)
- ❖ Line 33: INVALID_IDENTIFIER -> *ptr (ERROR: identifier cannot start with special character)
- ❖ Line 34: INVALID_NUMBER_FORMAT -> 456end (ERROR: identifier cannot start with digit)
- ❖ Line 34: INVALID_IDENTIFIER -> @test (ERROR: identifier cannot start with special character)
- ❖ Line 34: INVALID_IDENTIFIER -> #second (ERROR: identifier cannot start with special character)
- ❖ Line 34: INVALID_IDENTIFIER -> \$money (ERROR: identifier cannot start with special character)
- ❖ Line 34: INVALID_IDENTIFIER -> %value (ERROR: identifier cannot start with special character)
- ❖ Line 34: INVALID_IDENTIFIER -> ^power (ERROR: identifier cannot start with special character)
- ❖ Line 34: INVALID_IDENTIFIER -> &addr (ERROR: identifier cannot start with special character)
- ❖ Line 34: INVALID_IDENTIFIER -> *pointer (ERROR: identifier cannot start with special character)
- ❖ Line 35: INVALID_NUMBER_FORMAT -> 789middle (ERROR: identifier cannot start with digit)
- ❖ Line 35: INVALID_IDENTIFIER -> @var (ERROR: identifier cannot start with special character)
- ❖ Line 35: INVALID_IDENTIFIER -> #third (ERROR: identifier cannot start with special character)
- ❖ Line 35: INVALID_IDENTIFIER -> \$price (ERROR: identifier cannot start with special

- character)
- ❖ Line 35: INVALID_IDENTIFIER -> %rate (ERROR: identifier cannot start with special character)
 - ❖ Line 35: INVALID_IDENTIFIER -> ^exp (ERROR: identifier cannot start with special character)
 - ❖ Line 35: INVALID_IDENTIFIER -> &mem (ERROR: identifier cannot start with special character)
 - ❖ Line 35: INVALID_IDENTIFIER -> *data (ERROR: identifier cannot start with special character)
 - ❖ Line 36: INVALID_NUMBER_FORMAT -> 012begin (ERROR: identifier cannot start with digit)
 - ❖ Line 36: INVALID_NUMBER -> @123 (ERROR: invalid number - cannot start with special character)
 - ❖ Line 36: INVALID_NUMBER -> #456 (ERROR: invalid number - cannot start with special character)
 - ❖ Line 36: INVALID_NUMBER -> \$789 (ERROR: invalid number - cannot start with special character)
 - ❖ Line 36: INVALID_NUMBER -> %012 (ERROR: invalid number - cannot start with special character)
 - ❖ Line 36: INVALID_NUMBER -> ^345 (ERROR: invalid number - cannot start with special character)
 - ❖ Line 36: INVALID_NUMBER -> &678 (ERROR: invalid number - cannot start with special character)
 - ❖ Line 36: INVALID_NUMBER -> *901 (ERROR: invalid number - cannot start with special character)
 - ❖ Line 37: INVALID_IDENTIFIER -> @abc123 (ERROR: identifier cannot start with special character)
 - ❖ Line 37: INVALID_IDENTIFIER -> #def456 (ERROR: identifier cannot start with special character)
 - ❖ Line 37: INVALID_IDENTIFIER -> \$ghi789 (ERROR: identifier cannot start with special character)
 - ❖ Line 37: INVALID_IDENTIFIER -> %jkl012 (ERROR: identifier cannot start with special character)
 - ❖ Line 37: INVALID_IDENTIFIER -> ^mno345 (ERROR: identifier cannot start with special character)
 - ❖ Line 37: INVALID_IDENTIFIER -> &pqr678 (ERROR: identifier cannot start with special character)
 - ❖ Line 37: INVALID_IDENTIFIER -> *stu901 (ERROR: identifier cannot start with special character)
 - ❖ Line 38: INVALID_NUMBER_FORMAT -> 123abc456 (ERROR: identifier cannot start with digit)
 - ❖ Line 38: INVALID_NUMBER -> @123abc (ERROR: invalid number - cannot start with special character)
 - ❖ Line 38: INVALID_NUMBER -> #456def (ERROR: invalid number - cannot start with special character)

Uniqueness Requirement

Unique Keywords:

I designed **25 Urdu-script keywords** for my Mini C-like language. These include:

hum, tum, Wapas, Agar, Adadi, Mantiqi, shuru, khatam, jabtak, warna, magar, wo, socho, dalo, rahko, pakka, aur, keliye, lo, sahi, galat, ya, nahi, kaam, wapis, hatao,

Unique Operators:

I defined custom operators in addition to common ones:

- Standard operators: +, -, *, /, =
- Additional operators: ++, --, +=, <-, =~

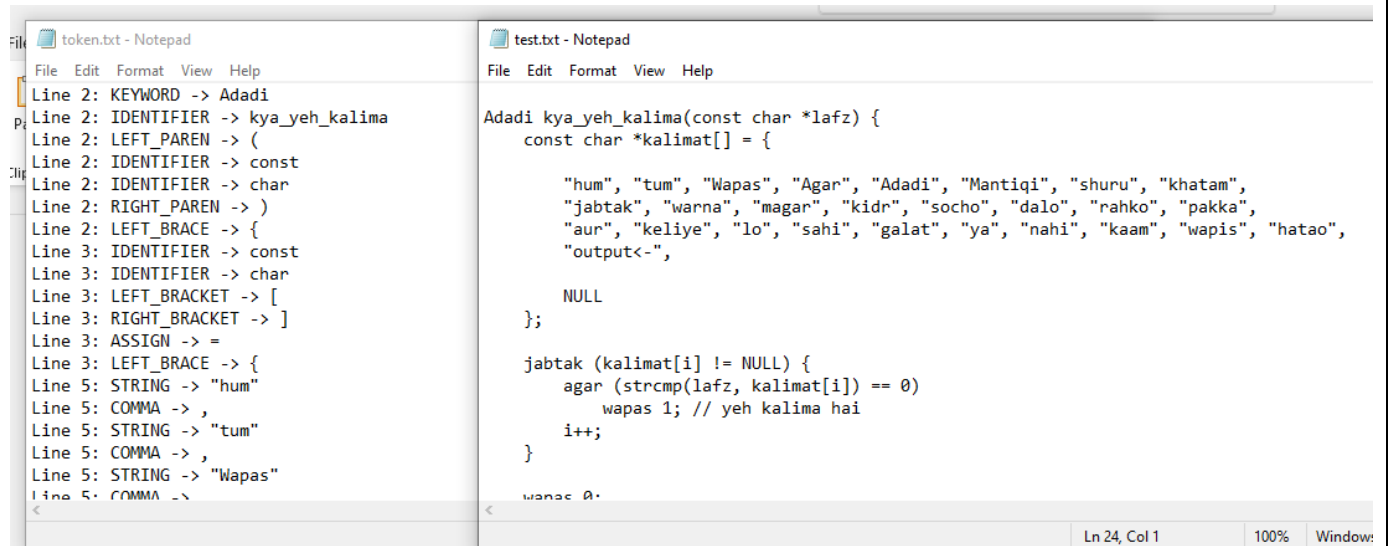
Unique Punctuations:

I chose punctuations that are commonly used in C++, but for uniqueness, my program emphasizes:

;	SEMICOLON
,	COMMA
(LEFT PAREN
)	RIGHT PAREN
{	LEFT BRACE
}	RIGHT BRACE
[LEFT BRACKET
]	RIGHT BRACKET
:	COLON
?	QUESTION

Why it's unique:

- These keywords are based on Urdu words to make the language more readable in a regional style with error detection
- Operators like <- and =~ are not standard C++ operators, which makes my language different
- Each keyword has a specific meaning in code, shuru = start of program, khatam = end, Wapas = return..



Unique Sample Program:

- I wrote a **220 line** Mini C++-like program using these keywords, operators, and punctuations.
- The program uses my Urdu-style syntax and cannot be confused with other students' programs.

Example test lines:

```
Adadi kya_yeh_kalima(const char *lafz) {
```

```
    const char *kalimat[] = {
```

```
        "hum", "tum", "Wapas", "Agar", "Adadi", "Mantiqi", "shuru", "khatam",
```

```
        "jabtak", "warna", "magar", "kidr", "socho", "dalo", "rahko", "pakka",
```

```
        "aur", "keliye", "lo", "sahi", "galat", "ya", "nahi", "kaam", "wapis", "hatao",
```

```
        "output<-",
```

```
        NULL
```

```
    };
```

```
@1 @2 @3 @4 @5 @6 @7 @8 @9 @0
```

```
#a #b #c #d #e #f #g #h #i #j
```

```
valid123 + @invalid * 45.67.89 / #wrong
```

```
"proper string" + 'bad char' * 12.34.56 / @test
```

```
@@ ## $$ %% ^^ && ** () << >>
```

```

+++ --- *** /// \\ <<< >>> ===

123start @begin #first $variable %percent ^caret &ref *ptr
456end @test #second $money %value ^power &addr *pointer
789middle @var #third $price %rate ^exp &mem *data
012begin @123 #456 $789 %012 ^345 &678 *901

+++ --- *** /// \\ <<< >>> === != >=< <=>

&&& ||| !!! ??? ::: ... ,,, ;;;

@#!$%^&*()_+=[\{}|;':",./<>? 123abc 45.67.89 "unclosed 'char" /* unfinished

+++ --- *** /// \\ <<< >>> === [] {} ;: ?. ,, .. @1 #a $2 %b ^c &d *e

"backslash \\\\" too many"

"mixed \x\y\z invalid"

123ab+ 45.67.89 * @invalid / #wrong % $error

"string" + 'char' * 12.34.56 / @test #value $result

// single line comment with // inside

// another // comment // with // multiple // slashes

/// triple slash

//// four slashes

```

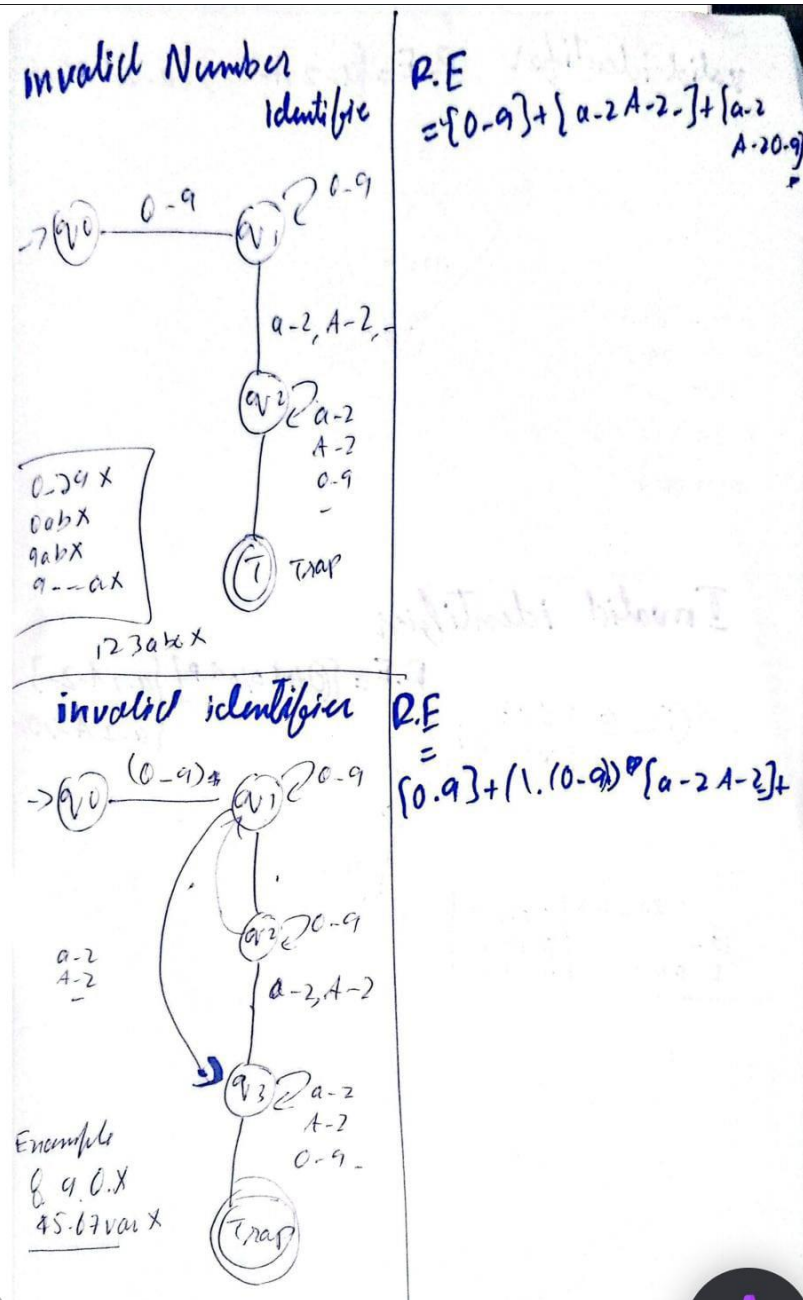
Sample tokens keywords:

hum, tum, Wapas, Agar, Adadi, Mantiqi, shuru, khatam, jabtak, warna, magar, wo, socho, dalo, rahko, pakka, aur, keliye, lo, sahi, galat, ya, nahi, kaam, wapis, hatao,

TOKEN With Regex Defination

Token Type	Regex	Description
Keywords	"hum", "tum", "Wapas", "Agar", "Adadi", "Mantiqi", "shuru", "khatam", "jabtak", "warna", "magar", "wo", "socho", "dalo", "rahko", "pakka", "aur", "keliye", "lo", "sahi", "galat", "ya", "nahi", "kaam", "wapis", "hatao", "output<-"	Reserved Urdu-style keywords.
Identifier	[a-zA-Z_] [a-zA-Z0-9_]*	Valid variable or function names.
Invalid Identifier	[@#\$\$%^&*] [a-zA-Z_] [a-zA-Z0-9_]*	Identifiers starting with special symbols.
Integer Number	[+-]? [0-9]+	Whole numbers with optional sign.
Floating Number	[0-9]*\.[0-9]+ ([Ee] [+ -]? [0-9]+)?	Decimal numbers, optional exponential.
Exponential Number	[0-9]+ ([Ee] [+ -]? [0-9]+)	Scientific notation numbers.
Invalid Number (multiple dots)	[0-9]+ (\.[0-9]+) {2, }	Numbers with multiple decimal points.
Invalid Number (multiple exponents)	[0-9]+ ([Ee] [+ -]? [0-9]+) {2, }	Numbers with multiple exponential parts.
Invalid Mixed Number	[0-9]+ [a-zA-Z_] [a-zA-Z0-9_]*	Numbers mixed with letters.
Operator	+, -, *, /, %, ++, --, =, +=, -=, *=, /=, %=, <=<=, >>=, &=, ^=, `	=, ==, !=, >, <, >=, <=, &&,
Invalid Operator (consecutive)	+++, ---, ***, ///, \\\	Consecutive operators not allowed.
Punctuation	,, ,, (,), {, }, [,], :, ?, .	Statement separators and block delimiters.
String Literal	" [^\n]* "	Strings in double quotes.
Invalid String	" [^\n]*	Unterminated string literal.
Character Literal	' [^\n]* ', ' \[nrt0abfv\\'\"] '	Single characters, valid escape sequences.
Invalid Character	' [^\n]* ', ' \[^nrt0abfv\\'\"] '	Unterminated or invalid char literal.
Single-line Comment	// [^\n]*	Ignored content after //.
Multi-line Comment	/* ... */	Ignored content between /* and */.
Whitespace	[\t]+	Spaces or tabs.
Newline	\r\n	\n

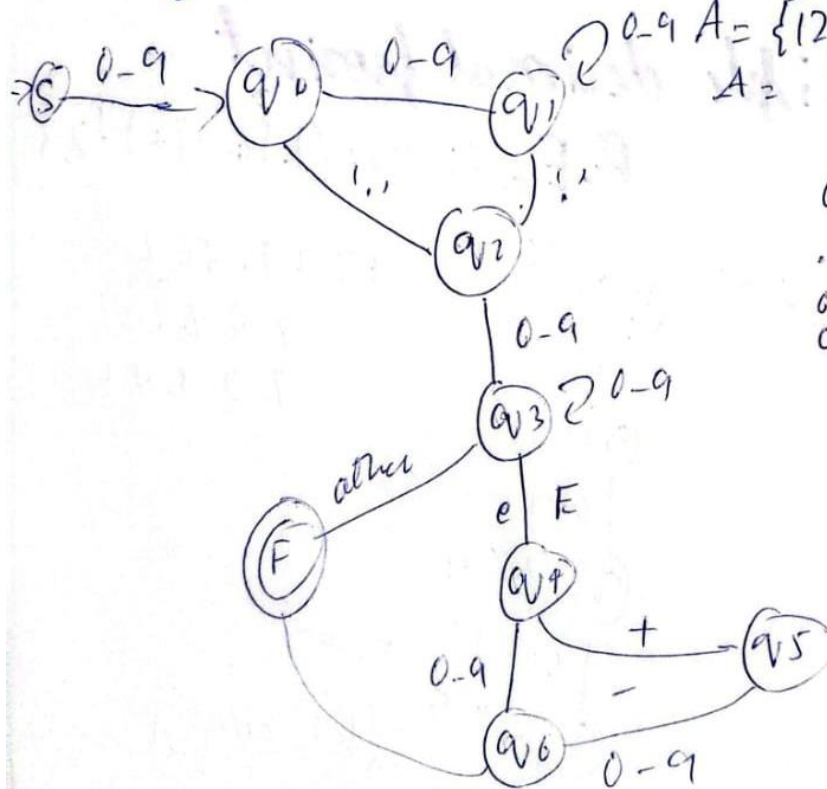
- Draw **finite automata (FA)** diagrams for Identifiers and Numbers (hand-drawn or digital).



float with exponent

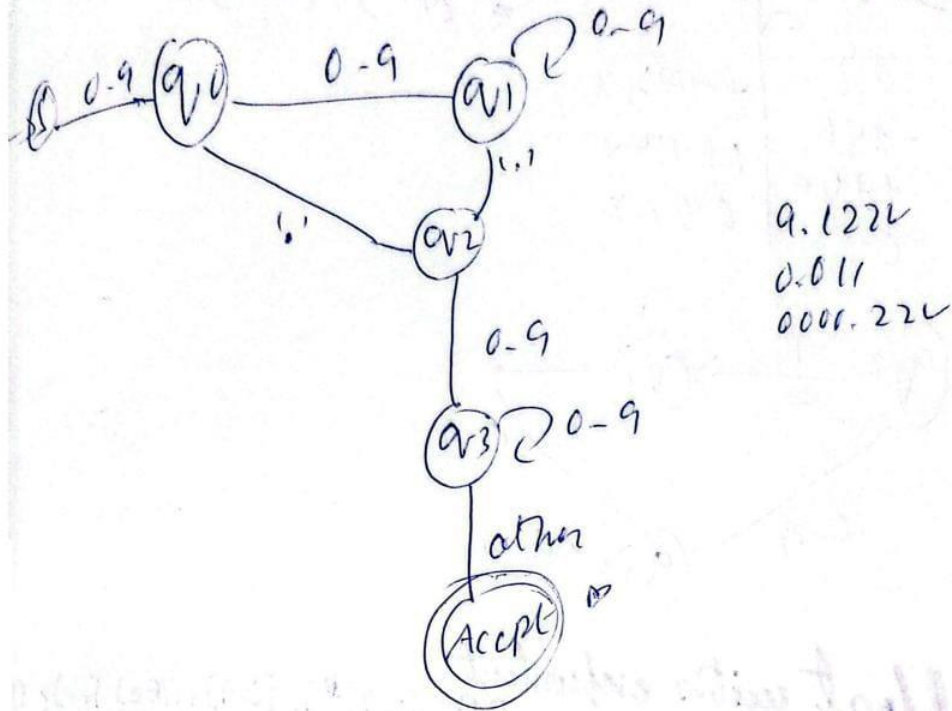
R.E. = $[0.9]^* \mid [0.9]^+ (Ee) (+|-)? [0-9]^*$

$A = \{12.34E+56\}$
 $A = \dots$



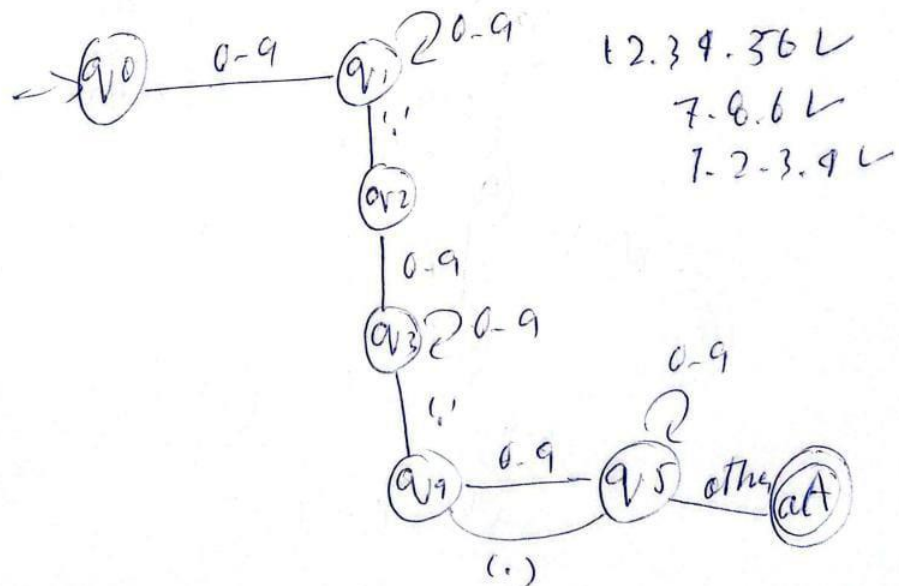
0.92
 .999
 0.99~
 00.99~

(float) P.E = $\{0.9\}^* 1. \{0.9\}^+$

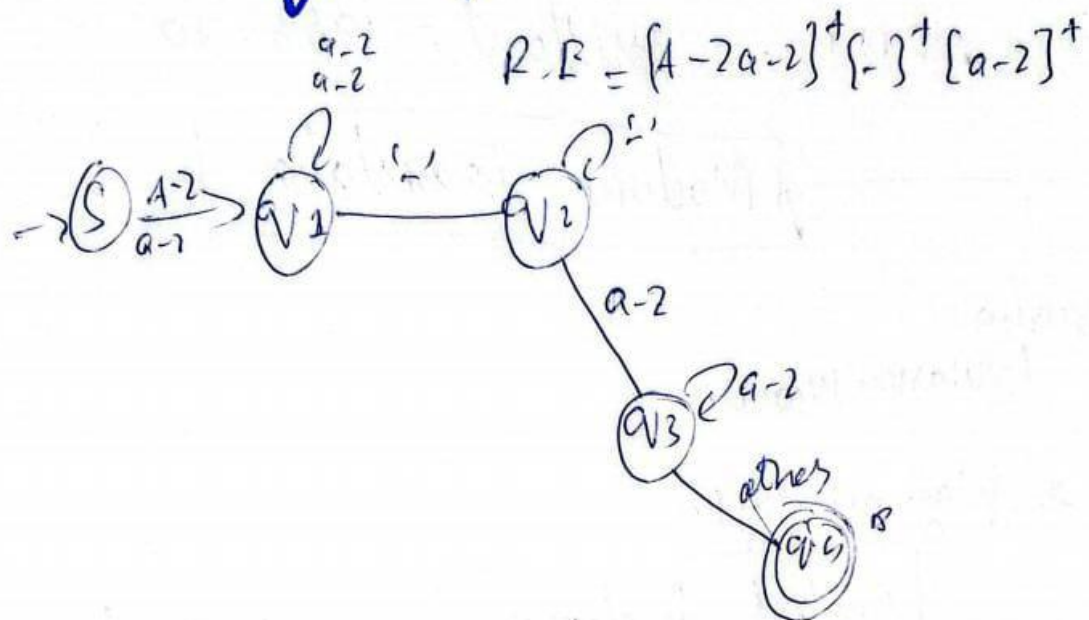


Multiple decimal point

P.E = $\{0.9\}^+ (1. \{0.9\}^+)^{\{2,3\}}$



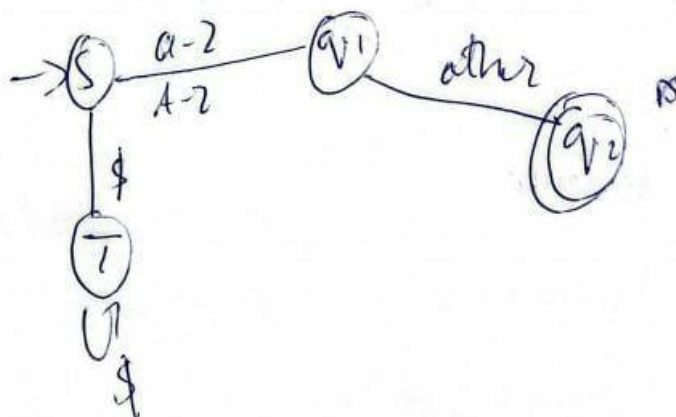
valid identifier
 full-path



Invalid identifier.

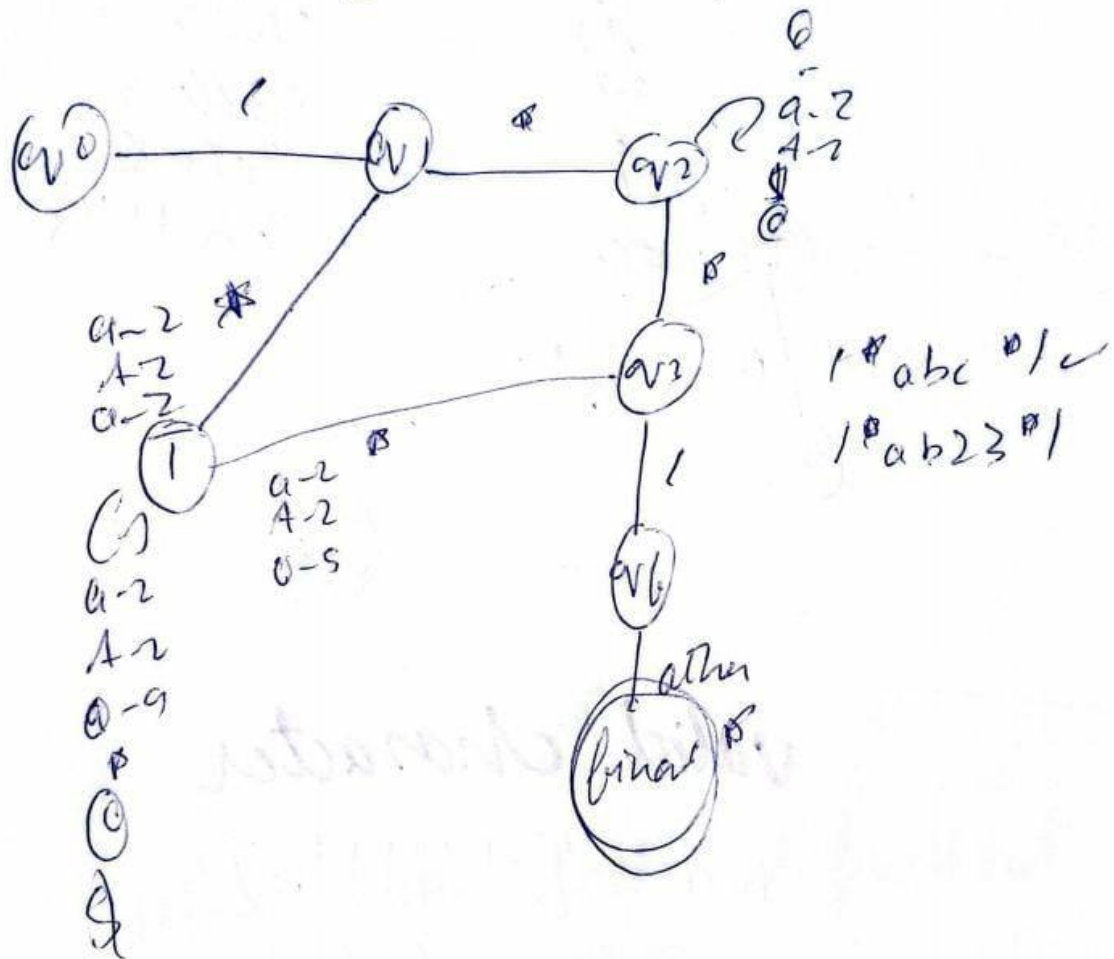
\$ variable

$$R.E = [^{\$}] [A-Za-z]$$

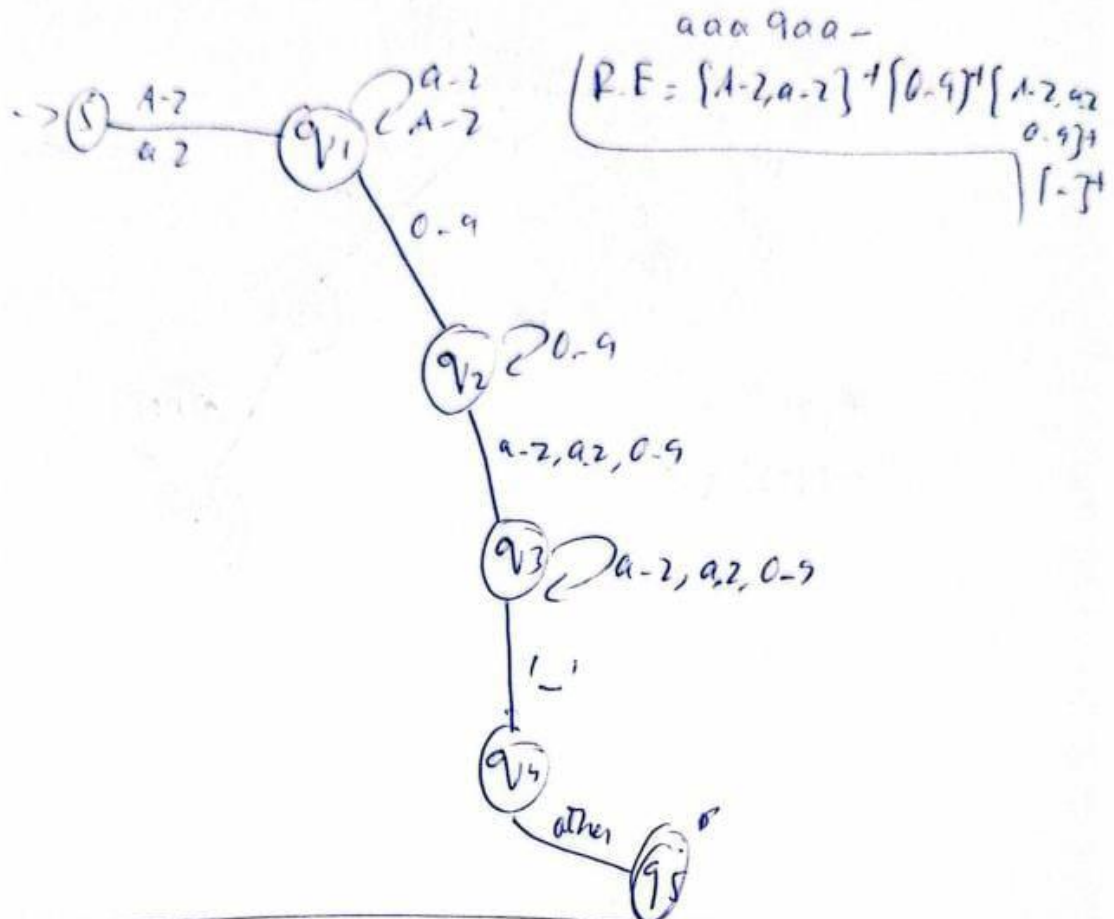


Multi line compiler

$$RE = 11^* (\{^1\} / \setminus \{^1\} /)^* 1^*$$



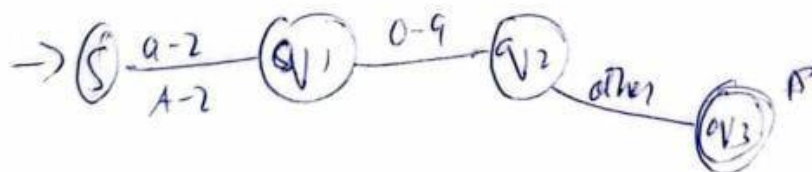
identifiers (Alphanumeric in start
letter/digits Next but
Must end with _)



valid identifiers

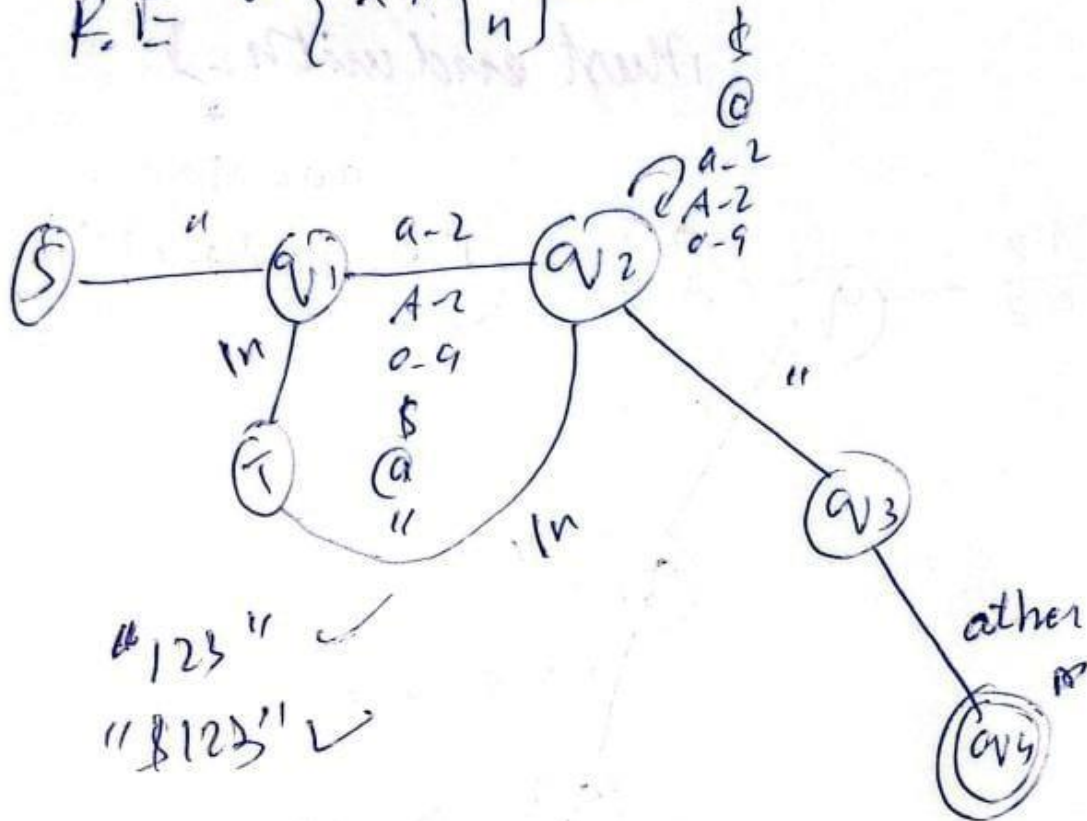
valid 123

$R.E = \{A-Z, a-z\}^+ \{0-9\}^+$



valid string

RE $\{^{\wedge} | n\}^{\$}$



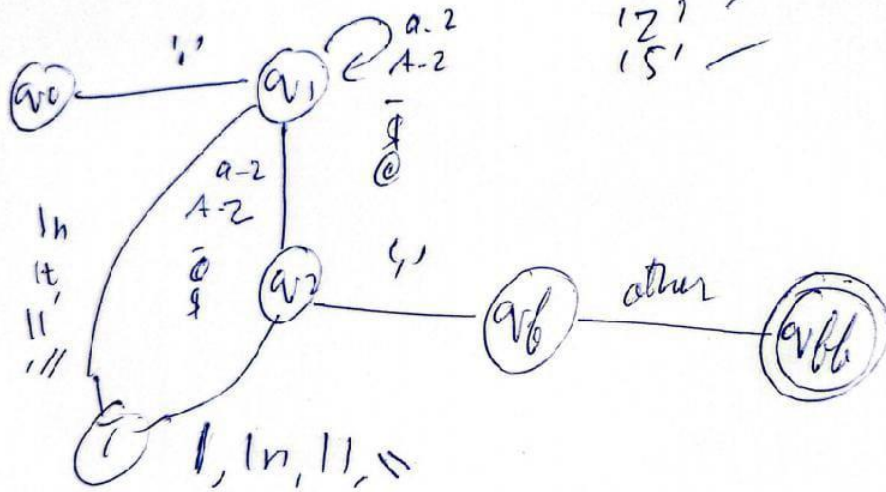
"123" ✓

"\$123" ✓

valid character

$RE = \{^{\wedge} ' | | | \backslash n \}$

'a' ✓
'z' ✓
'5' ✓




- Include a brief explanation of your **choice of 25 keywords** and why you designed them.

Keyword	Purpose / Meaning
hum	Represents the concept of “we” or default context, used for defining the main block or general scope.
tum	Represents “you”, could be used for user input or specific function scope.
Wapas	Acts as <code>return</code> in functions, signaling output of a value.
Agar	Conditional <code>if</code> statement.
Adadi	Represents numbers or numeric type, similar to <code>int</code> or <code>float</code> .
Mantiqi	Logical expressions, similar to boolean conditions.
shuru	Marks the start of a block, like <code>{</code> .
khatam	Marks the end of a block, like <code>}</code> .
jabtak	Represents a <code>while</code> or looping construct.
warna	Represents <code>else</code> in conditionals.
magar	Represents <code>else if</code> .
wo	Used for <code>switch</code> or selection statement.
socho	Represents <code>read</code> or input operations.
dalo	Represents <code>write</code> or output operations.
rahko	Used for <code>continue</code> statement in loops.
pakka	Used for <code>break</code> statement in loops.
aur	Logical AND operator.
keliye	Used in loops, similar to <code>for</code> .
lo	Assignment operator for special cases (<code>output <-</code>).
sahi	Represents <code>true</code> .
galat	Represents <code>false</code> .
ya	Logical OR operator.
nahi	Logical NOT operator.
kaam	Function keyword, similar to <code>void</code> .

• Source Code

i. Flex file (`urdu_lexer.l`) 700 lines of code.

 `urdu_lexer.l` - Notepad

File Edit Format View Help

```
%{
#include <stdio.h>
#include <string.h>
#include <ctype.h>

FILE *token_file;
FILE *error_file;
int line_no = 1;
int total_tokens = 0;
int total_errors = 0;


int in_comment = 0;

int is_keyword(const char *s) {
    const char *keywords[] = {
        "hum", "tum", "Wapas", "Agar", "Adadi", "Mantiqi", "shuru", "khatam",
        "jabtak", "warna", "magar", "wo", "socho", "dalo", "rahko", "pakka",
        "aur", "keliye", "lo", "sahi", "galat", "ya", "nahi", "kaam", "wapis", "hatao",
        "output<-", NULL
    };
    for (int i = 0; keywords[i] != NULL; i++)
        if (strcmp(s, keywords[i]) == 0)
            return 1;
    return 0;
}

const char* get_operator_name(const char *s) {
    if (strcmp(s, "+") == 0) return "PLUS";
    if (strcmp(s, "-") == 0) return "MINUS";
    if (strcmp(s, "*") == 0) return "MULTIPLY";
    if (strcmp(s, "/") == 0) return "DIVIDE";
    if (strcmp(s, "%") == 0) return "MODULO";
    if (strcmp(s, "++") == 0) return "INCREMENT";
}
```

<

SAMPLE TEST.TXT

 test.txt - Notepad

File Edit Format View Help

```
Adadi kya_yeh_kalima(const char *lafz) {
    const char *kalimat[] = {

        "hum", "tum", "Wapas", "Agar", "Adadi", "Mantiqi", "shuru", "khatam",
        "jabtak", "warna", "magar", "kidr", "socho", "dalo", "rahko", "pakka",
        "aur", "keliye", "lo", "sahi", "galat", "ya", "nahi", "kaam", "wapis", "hatao",
        "output<-",

        NULL
    };

    jabtak (kalimat[i] != NULL) {
        agar (strcmp(lafz, kalimat[i]) == 0){
            wapas 1; // yeh kalima hai
            i++;
        }

        wapas 0;
    }

    Adadi shuru() {
        likho("Roman Urdu Keyword Check Shuru Hua\\n");

        agar (kya_yeh_kalima("hum"))
            likho("Mila: hum\\n");
        warna
            likho("Nahi Mila\\n");

        khatam 0;
    }

    123start @begin #first $variable %percent ^caret &ref *ptr
    456end @test #second $money %value ^power &addr *pointer
    789middle @var #third $price %rate ^exp &mem *data
    012begin @123 #456 $789 %012 ^345 &678 *901
    <
```

TOKEN.TXT


token.txt - Notepad

```
File Edit Format View Help
Line 2: KEYWORD -> Adadi
Line 2: IDENTIFIER -> kya_yeh_kalima
Line 2: LEFT_PAREN -> (
Line 2: IDENTIFIER -> const
Line 2: IDENTIFIER -> char
Line 2: RIGHT_PAREN -> )
Line 2: LEFT_BRACE -> {
Line 3: IDENTIFIER -> const
Line 3: IDENTIFIER -> char
Line 3: LEFT_BRACKET -> [
Line 3: RIGHT_BRACKET -> ]
Line 3: ASSIGN -> =
Line 3: LEFT_BRACE -> {
Line 5: STRING -> "hum"
Line 5: COMMA -> ,
Line 5: STRING -> "tum"
Line 5: COMMA -> ,
Line 5: STRING -> "Wapas"
Line 5: COMMA -> ,
Line 5: STRING -> "Agar"
Line 5: COMMA -> ,
Line 5: STRING -> "Adadi"
Line 5: COMMA -> .
```

token.txt - Notepad

```
File Edit Format View Help
Line 6: COMMA -> ,
Line 6: STRING -> "socho"
Line 6: COMMA -> ,
Line 6: STRING -> "dalo"
Line 6: COMMA -> ,
Line 6: STRING -> "rahko"
Line 6: COMMA -> ,
Line 6: STRING -> "pakka"
Line 6: COMMA -> ,
Line 7: STRING -> "aur"
Line 7: COMMA -> ,
Line 7: COMMA -> ,
Line 7: STRING -> "lo"
Line 7: COMMA -> ,
Line 7: STRING -> "sahi"
Line 7: COMMA -> ,
Line 7: STRING -> "galat"
Line 7: COMMA -> ,
Line 7: STRING -> "ya"
Line 7: COMMA -> ,
Line 7: STRING -> "nahi"
Line 7: COMMA -> ,
Line 7: STRING -> "kaam"
Line 7: COMMA -> ,
Line 7: STRING -> "wapis"
Line 7: COMMA -> ,
Line 7: STRING -> "hatao"
Line 7: COMMA -> ,
Line 8: STRING -> "output<-"
Line 8: COMMA -> ,
Line 10: IDENTIFIER -> NULL
Line 11: RIGHT_BRACE -> }
Line 11: SEMICOLON -> ;
Line 13: KEYWORD -> jabtak
Line 13: LEFT_PAREN -> (
Line 13: IDENTIFIER -> kalimat
```

Error.txt

 error.txt - Notepad

File Edit Format View Help

```
Line 2: INVALID_IDENTIFIER -> *lafz (ERROR: identifier cannot start with special character)
Line 3: INVALID_IDENTIFIER -> *kalimat (ERROR: identifier cannot start with special character)
Line 7: INVALID_NUMBER_MULTIPLE_EXPONENTS -> "keliye" (ERROR: invalid number - multiple exponents)
Line 23: INVALID_NUMBER_MULTIPLE_EXPONENTS -> "Roman Urdu Keyword Check Shuru Hua\\n" (ERROR: invalid
Line 33: INVALID_NUMBER_FORMAT -> 123start (ERROR: identifier cannot start with digit)
Line 33: INVALID_IDENTIFIER -> @begin (ERROR: identifier cannot start with special character)
Line 33: INVALID_IDENTIFIER -> #first (ERROR: identifier cannot start with special character)
Line 33: INVALID_IDENTIFIER -> $variable (ERROR: identifier cannot start with special character)
Line 33: INVALID_IDENTIFIER -> %percent (ERROR: identifier cannot start with special character)
Line 33: INVALID_IDENTIFIER -> ^caret (ERROR: identifier cannot start with special character)
Line 33: INVALID_IDENTIFIER -> &ref (ERROR: identifier cannot start with special character)
Line 33: INVALID_IDENTIFIER -> *ptr (ERROR: identifier cannot start with special character)
```

1. Demo Video (5 minutes max)

- i. Demonstrate running your scanner on your unique program.
- ii. Explain how your regex works for at least 2 tokens.
- iii. Highlight one error-handling ex

Rubrics (10 Marks)

Criteria	Marks	Description
Regex Definitions & Correctness	2	Regex table is complete, accurate, and handles edge cases.
Flex Implementation (Tokenization)	2	Scanner correctly recognizes tokens and prints them with line numbers.
Error Handling	1	Invalid tokens are reported with line numbers.
Uniqueness & Creativity	3	Student-designed 15 keywords + operators + punctuations, plus unique sample program.
Diagrams (FA for Identifier & Number)	1	Clear, correct diagrams provided.
Demo & Explanation (Video& Viva)	1	Student demonstrates execution and explains regex.