CS-402 Compiler Construction
Lexical Analysis
(Documenting the Design of Lexer)

1. Language Tokens and Lexemes

Token	Description	Lexemes
INT	The data type int i.e., letters/characters i, n, t	int
CHAR	The data type char i.e., letters/character c, h, a, r	char
IF	It is a keyword made of letters i, f	if
ELIF	It is a keyword made of letters e, l, i, f	elif
ELSE	It is a keyword made of letters e, l, s, e	else
WHILE	It is a keyword made of letters w, h, i, l, e	while
INPUT	It is a keyword made of letters i, n, p, u, t	input
PRINT	It is a keyword made of letters p, r, i, n, t	print

PRINTLN	It is a keyword made of letters p, r, i, n, t, l, n	println
REL_OP	Relational operators that compare two operands	<
		<=
		>
		>=
		==
		~=
‘+’	The arithmetic operator for addition	+
‘-’	The arithmetic operator for subtraction	-
‘*’	The arithmetic operator for multiplication	*
‘/’	The arithmetic operator for division	/
ID	Letter followed by letters, digits or underscore	p2, max_ etc.
NUM	Digits that form only integers	0, 123, 99 etc.
LIT	A single letter/character enclosed in single quotes	‘x’, ‘a’ etc.
STR	Sequence of letters/characters and while spaces enclosed in double quotes	“hello there you” etc.
S_COMMENT	It’s a single line comment	// comment

M_COMMENT	It’s a multi-line/ block comment	/* comment */
‘=’	It is the assignment operator	=
INPUT_OP	It is an operator used for taking input	->
‘:’	It is the punctuation mark “:”	:
‘;’	It is the punctuation mark “;”	;
‘,’	It is the punctuation mark “,”	,
‘(’	It is the punctuation mark “(”	(
‘)’	It is the punctuation mark “)”	)
‘{’	It is the punctuation mark “{”	{
‘}’	It is the punctuation mark “}”	}
‘[’	It is the punctuation mark “[”	[
‘]’	It is the punctuation mark “]”	]

Note:

All tokens having a symbol within single quotes represent ASCII values.

For single-line comments, the token isn't meant to be sent to parser rather whenever the LA reads // it will read the complete line till newline character ('\n') and ignore it or remove it by replacing by "".

Similarly, for multi-line comments the LA will everything after /* till */ and ignore or remove it.

For tokens having multiple lexemes, each lexeme will be given a attribute value or will be recorded in the symbol table.

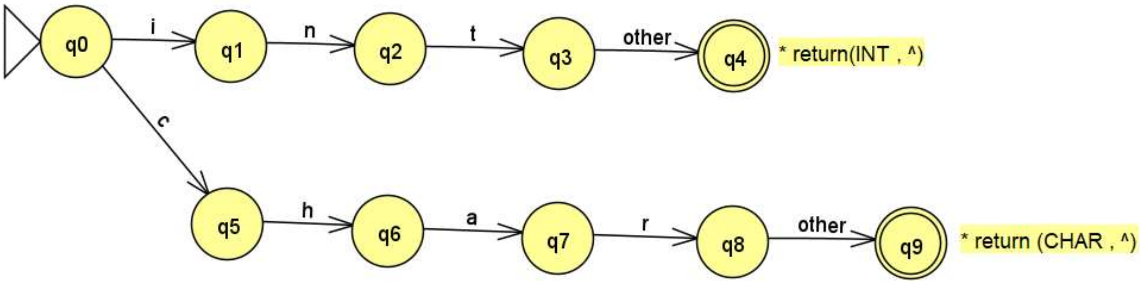
i. **Data Types:** int, char

These are basically sequence of letters.

Regular Definition

DT → int | char

Transition Diagram



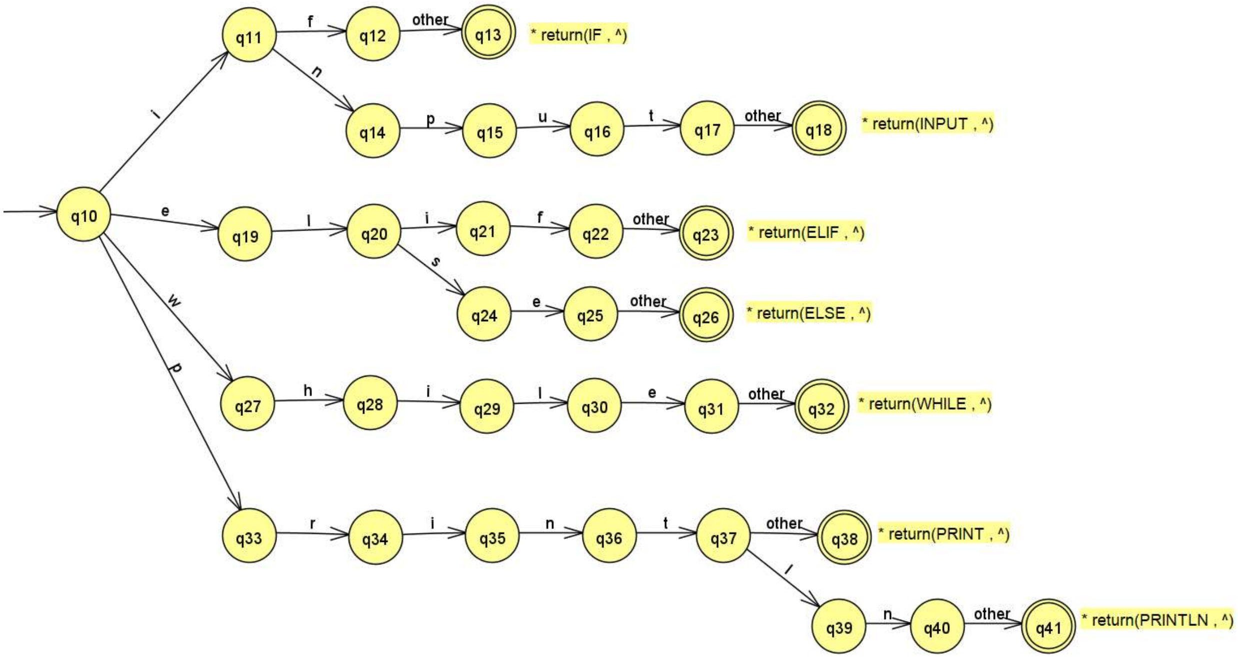
ii. **Keywords:** if, elif, else, while, input, print, println

These keywords are basically sequence of letters as well.

Regular Definition

KW → if | elif | else | while | input | ((print)(^ | ln))

Transition Diagram

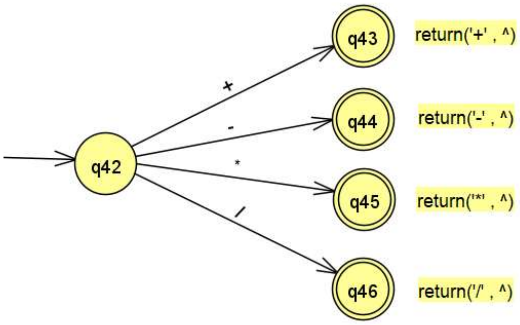


iii. **Arithmetic Operators:** +, -, *, /

Regular Definition

arithOp → + | - | * | /

Transition Diagram

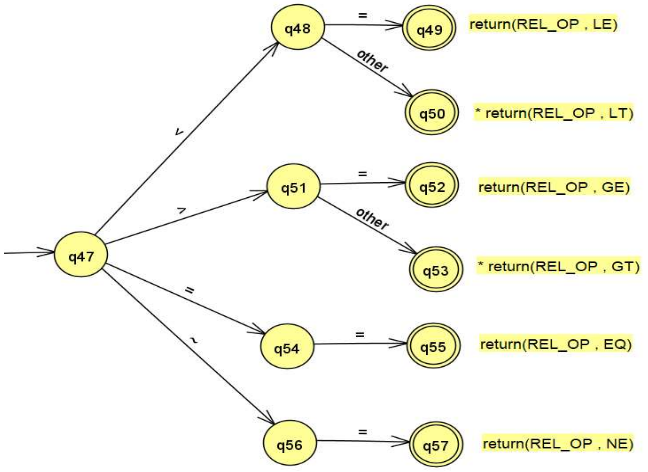


iv. **Relational Operators:** <, <=, >, >=, ==, ~=

Regular Definition

relOp → < | <= | > | >= | == | ~=

Transition Diagram



v. Comments

Single-line comments are // followed by a comment which can constitute any character till newline symbol.
Multi-line comments are comment enclosed in /* */ and can constitute any character even newline symbol.

Regular Definition

letter → A | B | … | Z | a | b | … | z

digit → 0 | 1 | … | 9

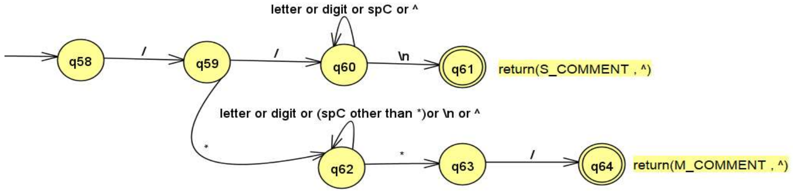
newline → \n

spC → ~ | @ | $ | % | & | * | (|) |{ | } | [|] | + | = | _ | - | \ | / | < | > | . | , | " | " | ' | space | : | ; | ?
| |

S_COMMENT → // (*letter* | *digit* | *spC* | ^)* *newline*

M_COMMENT → /* (*letter* | *digit* | *spC* | *newline* | ^)* */

Transition Diagram


letter or digit or spC or ^
q58
/
q59
/
q60
\n
q61
return(S_COMMENT , ^)
letter or digit or (spC other than *)or \n or ^
*
q62
*
q63
/
q64
return(M_COMMENT , ^)

vi. **Identifier:** a letter followed by any number of letters or digits or underscore symbol

Regular Definition

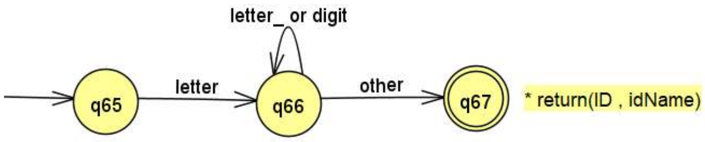
$$letter \rightarrow \mathrm{A} \mid \mathrm{B} \mid \ldots \mid \mathrm{Z} \mid \mathrm{a} \mid \mathrm{b} \mid \ldots \mid \mathrm{z}$$

$$letter\_ \rightarrow letter \mid \_$$

$$digit \rightarrow 0 \mid 1 \mid \ldots \mid 9$$

$$ID \rightarrow letter\ (letter\_ \mid digit)^{*}$$

Transition Diagram



Here, idName will be the name of the identifier recognized.

vii. **Numeric Constants:** only integers

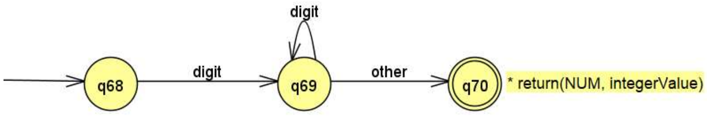
These are sequence of digits that form only integers i.e., they don't include fraction, exponential parts etc.

Regular Definition

$$digit \rightarrow 0 \mid 1 \mid \ldots \mid 9$$

$$NUM \rightarrow digit^{+}$$

Transition Diagram



Here, integerValue will be the value of the NUM recognized.

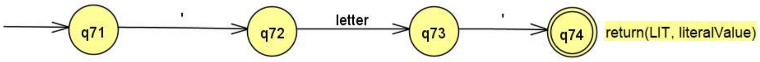
viii. **Literal Constants:** a letter enclosed in single quotes

Regular Definition

$$letter \rightarrow \mathrm{A} \mid \mathrm{B} \mid \ldots \mid \mathrm{Z} \mid \mathrm{a} \mid \mathrm{b} \mid \ldots \mid \mathrm{z}$$

<p align="center"><i>LIT</i> → ‘ (<i>letter</i>) ’</p>

Transition Diagram



Here, literalValue will be the value of the LIT recognized.

ix. **Strings:** sequence of letters and white spaces enclosed in double quotes

Regular Definition

letter → A | B | … | Z | a | b | … | z

digit → 0 | 1 | … | 9

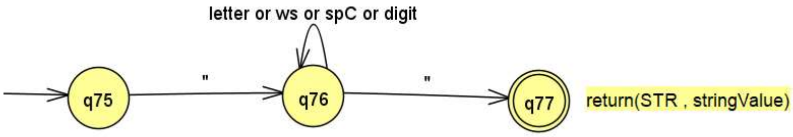
newline → \n

ws → (blank | tab | *newline*)$^+$

spC → ~ | @ | $ | % | & | * | (|) |{ | } | [|] | + | = | _ | - | \ | / | < | > | . | , | “ | ” | ' | space | : | ; | ?
| |

STR → “ (*letter* | *ws* | *spC* | *digit*)* ”

Transition Diagram

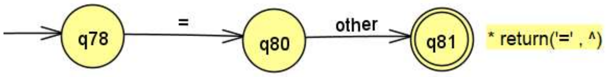


x. **Assignment Operator:** =

Regular Definition

assignOp → =

Transition Diagram

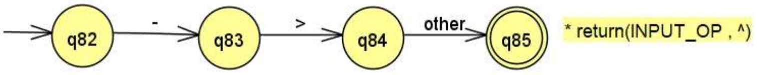


xi. **Input Operator:** ->

Regular Definition

INPUT_OP → ->

Transition Diagram



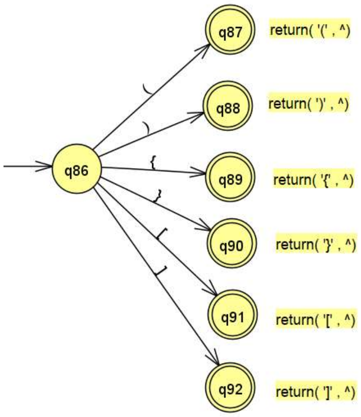
xii. Parenthesis, Braces, Square Brackets: (,), {, }, [,]

These are basically punctuation symbols being used as operators.

Regular Definition

PBC → (|) | { | } | [|]

Transition Diagram



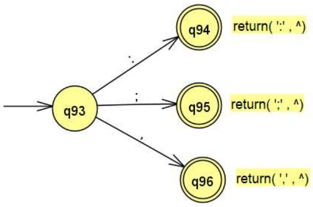
xiii. Colon, Semi Colon, Comma: :, ;, ,

These are basically punctuation symbols being used as operators.

Regular Definition

SCC → : | ; | ,

Transition Diagram



2. Overall Regular Expressions

letter → A | B | … | Z | a | b | … | z

letter_ → letter | _

digit → 0 | 1 | … | 9

NUM → digit^{+}

ID → letter (letter_ | digit)*

newline → \n

ws → (blank | tab | newline)$^{+}$

LIT → ‘ (letter) ’

STR → “ (letter | ws | spC | digit)* ”

INT → int

CHAR → char

IF → if

ELIF → elif

ELSE → else

WHILE → while

INPUT → input

PRINT → print

PRINTLN → println

INPUT_OP → ->

spC → ~ | @ | $ | % | & | * | (|) |{ | } | [|] | + | = | _ | - | \ | / | < | > | . | , | “ | ” | ' | space | : | ; | ?
| |

S_COMMENT → // (letter | digit | spC | ^)* newline

M_COMMENT → /* (letter | digit | spC | newline | ^)* */

assignOp → =

arithOp → + | - | * | /

relOp → < | <= | > | >= | == | ~=

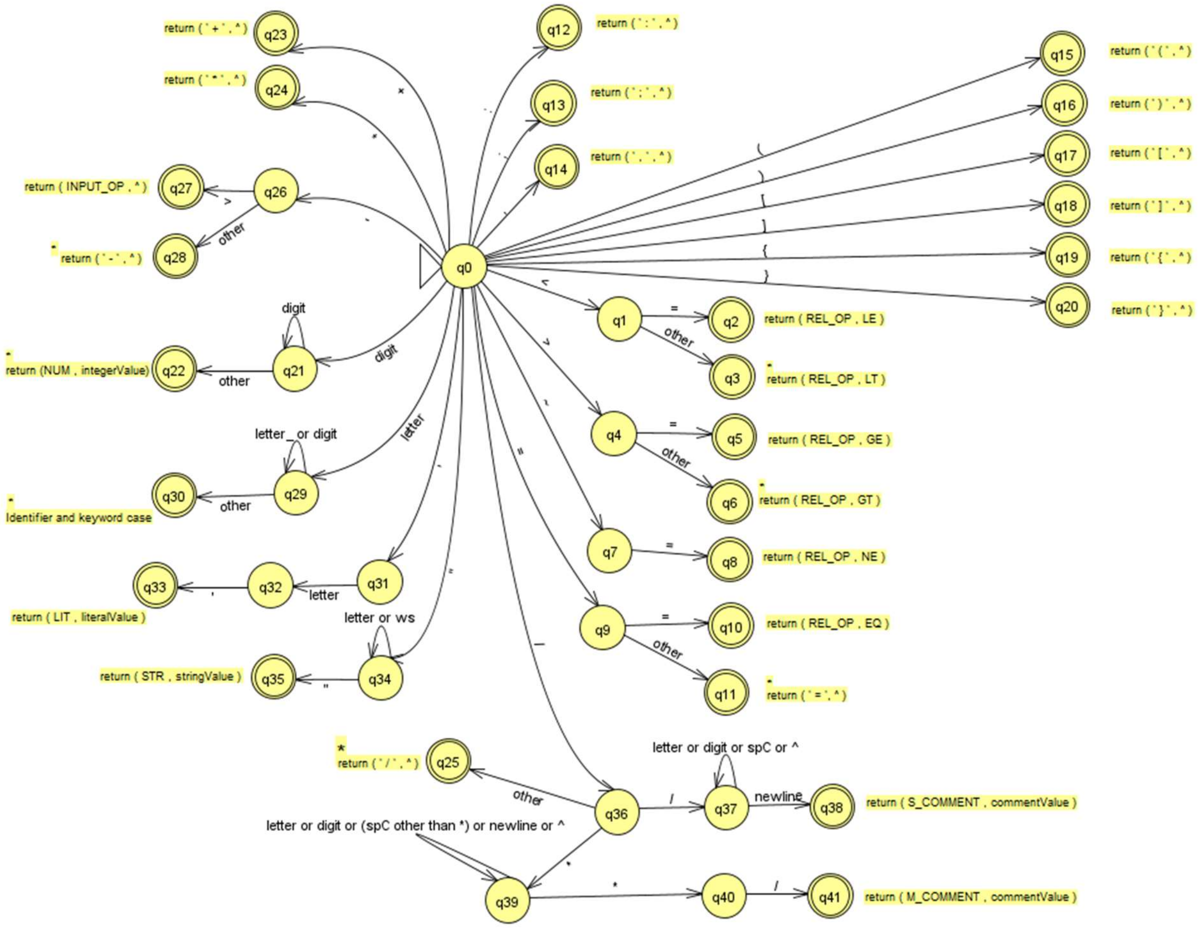
$$PBC \rightarrow (\mid) \mid \{ \mid \} \mid [\mid]$$

$$SCC \rightarrow : \mid ; \mid ,$$

Here, we have expanded the data types and keywords as individual tokens for clarity.

3. Overall Finite State Machine

Now, we will combine all the transition diagrams into one big finite state machine. It is optimized by minimizing the states to necessary ones and combines some token cases like that of identifiers and keywords.



Here, all keywords are placed into the identifier case as they all qualify as valid identifiers. Later on, keywords will be distinguished from identifiers by looking up the symbol table for reserved words.