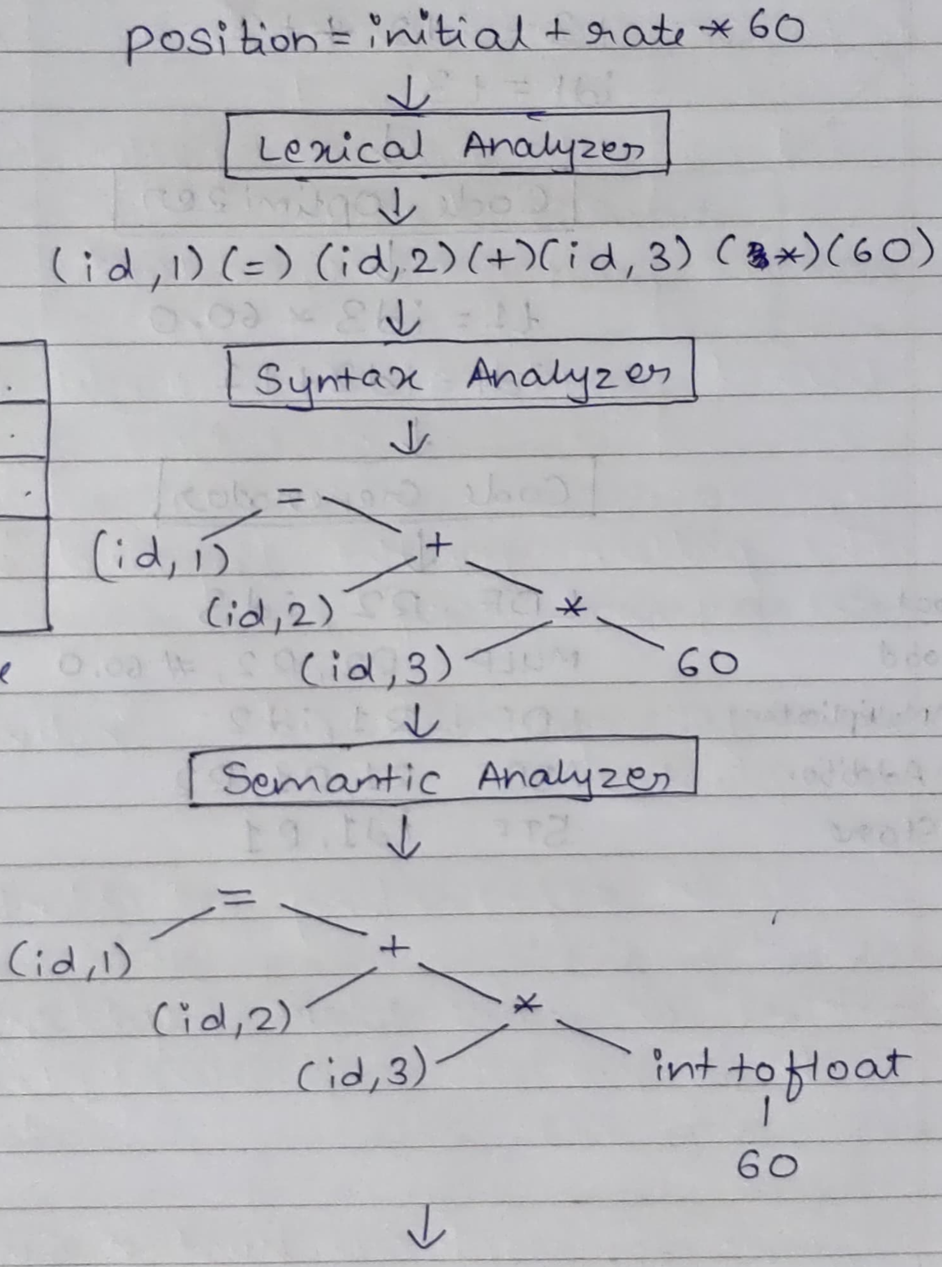
- Diff b/w compiler & interpretor
- Language Processing System(Explanation}

Structure of Compiler

Eg:



1	position	- - .
2	initial	- - .
3	rate	- - .

Symbol Table

Lexeme: Lexical analyser reads the source program and groups then into meaningful sequences called lexemes. For each lexeme, lexical analyser generates a token.

↓

Intermidiat Code Generator

↓

```
t1 = inttofloat (60)
t2 = id 3 * t1
t3 = id2 + t2
id1 = t3
```

↓

Code optimizer

↓

```
t1 = id3 * 60.0
id1 = id2 + t1
```

↓

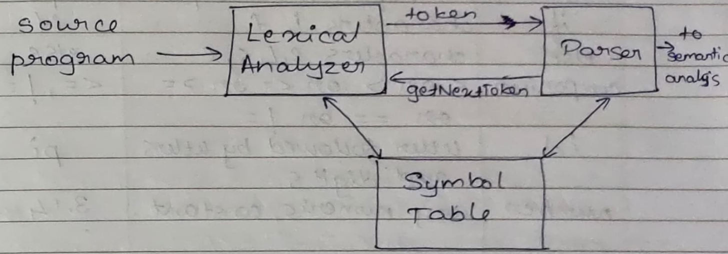
Code Generador

↓

```
F-Float               LDF   R2, id3
LD-Load               MULF  R2, R2, #60.0
MUL→Multiplication    LDF   R1, id2
ADD→Addition          ADDF  R1, R1, R2
ST→Store              STF   id1, R1
```

▸ Lexical Analyzer

- The role of Lexical Analyzer



- Lexical ~~Aa~~ Analysis vs Parsing

Lexical Analysis is also known as scanning

Parsing is also known as Syntax Analysis

- Token

A token is a pair consisting of a token name & an optional ~~au~~ attribute value

- Pattern

A pattern is a description of the form that the leximes of a total make take

- Lexemes

Lexemes is a Sequence of characters in a source program that matches the pattern for a token

In many programming languages the following covers most of the tokens

Token	Informal description	sample lexemes
if	characters i, f	if
else	characters e, l, s, e	else
comparison	< or > or <= or >= or == or !=	<=, !=
id	letter followed by letters and digits	pi
number	any numeric constant	3.14
literal	anything but "surrounded by"	"Hello"

1) The pattern for the keyword is same as the keyword itself

2) Tokes ~~for~~ operat~~ion~~ors either individualls or in classes

3) One token representing all identifiers

4) ~~Lett~~ One or more tokens representing constants such as numbers and literal strings

5) Tokens for each punctuation symbol such as left and right parenthesis ',' and ';'.

* Attributes for token

E = M * C ** 2

<id,1> <=> <id,2> <*> <id,3> <**> ~~<>~~

1	E	...
2	M	...
3	C	...

* Lexical Errors

1) Panic mode
We delete successive characters from the remaining input until the lexical analyser can find a well formed token at the beginning of what input is left

Delete one character from the remaining input.

Insert the ~~rem~~ missing character into remaining input

Replace a character by another character

Transpose two adjacent characters

~~*~~ Specification of tokens

1) Strings & Languages
2) Operation and languages
3) Regular expressions

* union, concatenation & closure

5m Imp * Regular Def^n s

$d_1 \rightarrow r_1$
; $d_2 \rightarrow r_2$
⋮
$d_n \rightarrow r_n$

For notational convinience we give names to certain regular expressions & use those names in subsequent expressions.

If Σ is an alphabut then the regular expression is a sequence of definitions of the form

$d_1 \rightarrow r_1$,
$d_2 \rightarrow r_2$
⋮
$d_n \rightarrow r_n$

where

→ each d_i is a new symbol not in Σ and not the same as any other d's

→ Each r_i is a regular expression over the alphabet $\Sigma \cup \{d_1, d_2 \ldots d_{r-1}\}$

- Write regular def^n for an identifier
  d_1 letter → a|b|c ..... |z |A|B ..... |Z r_1
  d_2 digit → 0|1|.....|9 r_2
  d_3 id → letter(letter|digit) r_3

digit → 0|1|....|9
digits → digit(digit)*
~~Operatio~~ optional fraction → .digits | ε
optional exponent → (E(+|-|ε)digits)|ε
number → digits optional Fraction optional Exponent

* Input Buffering

India Belgaum karnataka

E = M * C * * 2

E	=	M	*	C	*	*	2	eof		

→ Each buffer is of the same size n and n is size of a disc block i.e 4096 byte

→ Using 1 system read command we can read n characters into the buffer

→ If xn less than n characters remains in the input file then a special character represented by 'eof' marks the end of the source files

→ Two pointers to the input are maintained
1) Lexeme begin - marks the beginning of the current lexeme.

2) Forward- Forward scans ahead until a pattern match is found.

B

* Sentinel

* Recognition of tokens

stmt → if expr then stmt
      | if expr then stmt else stmt
      | E

expr → term relop term
      | term

term → id
      | number

digit → 0|1| - - - - |9
digits → digit(digit)*
number → digits (. digits)?(E [+−]?digits)?
letter → [A − Z a−z]
id → letter (letter | digits)*
if → if
else → else
then → then
relop → < | > | <= | >= | = | <>
ws (white space) → (blank| tab| newline)+

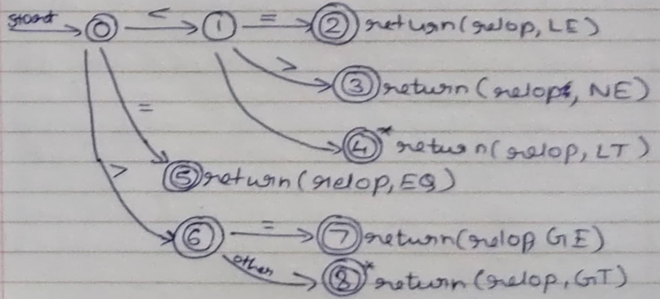
relop-relational operator

~~Any ws~~

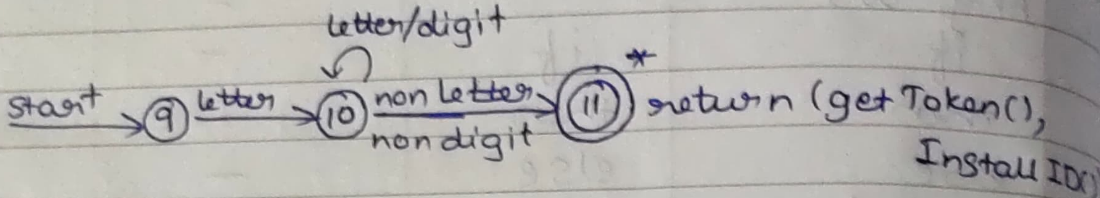
Lexeme	Token	Attribute value
Any ws	−	−
if	if	−
then	then	−
else	else	−
any id	id	pointer to table entry
any number	number	pointer to table entry
<	relop	LT
>	relop	GT
<=	relop	LE
>=	relop	GE
=	relop	EQ
~~≠~~ <>	relop	NE → not equal to

* Transition Diagram

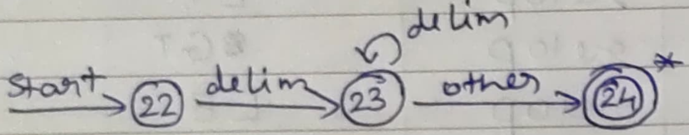
▸ Transition Diagram for Relational operators



- Transitional diagram for identifiers and keywords



- Transition diagram for white space



- Transition diagram for unsigned numbers