C2.
C3.

TC

Alg Scanner - V1
while not (eof(f)) do
{ detect (token);
classify (token);
codify (token);
}

1.) Detect tokens

if (x <= R) or (x == y) { x = x+1;
y = y-x; }

- separators - elems of the lang {, (,), }, ;,
- white spaces: space, new line, indent (tab)

if ␣␣ (x <= 2)
or
(x == y)

Remarks:
1.) white spaces have no meaning
if there are several white spaces -> eliminated
2.) lookahead technique:
< = > when 2] find this, look ahead for "="

2.) Classify:

put them in a list classes

1.) - reserved words - meaning defined by the lang
- cannot change the meaning
(keywords - can be changed)

: if, then, while,

2.) - operators + , - , ... and, or, not, <, >, <=, =, ...

3.) - separators : (,), [,], {, }, ,, ;, :

4.) - identifiers - RULES!

5.) - Constants - RULES!

Remark:

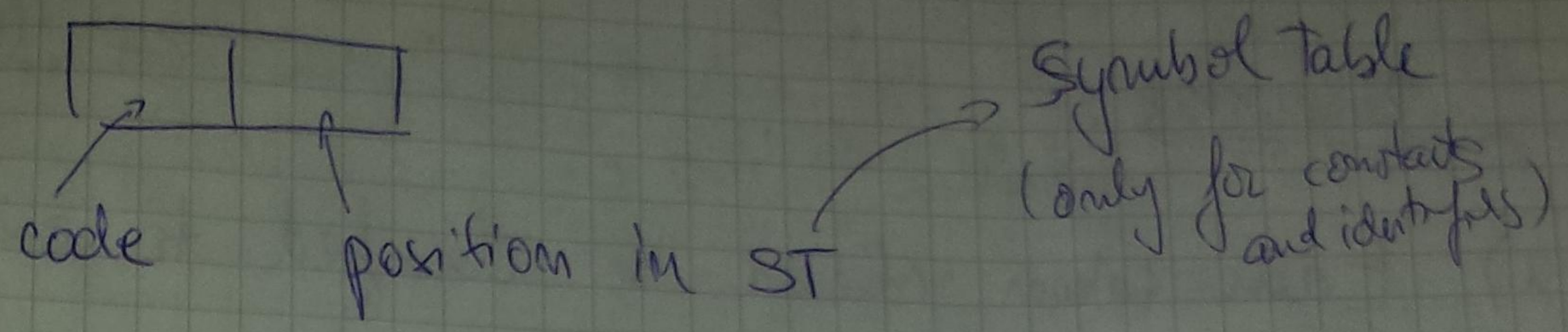
A lexical error appears when a token cannot be classified!

3.) Codify

1	identifier					37	OR
2	constant	15	+	20	<	40	(
3	if	16	-	21	<=	41	)
4	then	17	*	22	==	42	{
5	while					43	}
⋮		⋮		⋮		⋮	
				30	=	50	;

not pos in ST

PIF (Program Internal Form)



if token ≠ identifier
or
token ≠ constant) code ≥ 3

pos in ST = −1 (does not exist)

```
Alg Scanner V2
while not eof(f) do
  { detect (token) ;
    if (token ∈ reserved words) or (token ∈ operators)
  or (token ∈ seperator)
      then  generate_PIF( code[token], -1 )
      else ( if (token ∈ identifier)
                { Kod = pos(token, ST);
                  generatePIF(1, Kod);
                }
            else (if (token is Constant)
                { Kod = pos(token, ST);
                  generatePIF(2, Kod);
                }
```

else "Lexical error" token (the current token

}

Remarks:

1.) Each token should be classified to only 1 class.
if - reserved word, and not identifier, num
if it respects the rule.

2.) pos (token, ST)
search token in ST
if found then {return position}
else {insert token ST.
return position;}

Symbol Table
- collects all the information about symbolic na
(ideff, const, ...)
- each symbolic name, appear once!

name	type	mem_addr	value

int a;

1. → ST - unique → const, identif.

2. → 2ST → one for identifiers → IT
→ one for constants → CT

imperative — declarative

structural — procedural (modular procedures and functions) — functional — logical

no goto
loops and ifs.
no func.

pre, post, in — order
lexicograph.

! Organize ST?

- fast alg. for search and insert.

1.)- unsorted table → order in which id/const are found in source program.
searching → bad! (liniar, etc.)

2.)- sorted table →
lexicographic order: searching → binary
insert

3.) Binary Search Tree (BST) search → better
insert → better

4. Hash Table

$h : K \rightarrow A$

K - set of keys

A - positions into ST

- $k_1, k_2, k_1 = k_2$

$h(k_1) = h(k_2)$

↳ collision

$$h(\text{token}) = \sum_{(c \in \text{token})} \text{Ascii code}(c) \bmod m$$

$|M| = A$

Scanner

1.) - eliminates white spaces

2.) - Scanner eliminates comments

```
function f-( )
  { int i;
    . . . .;
    function g( )
      { . . .
      }
  }
function h( ) { . . . }
```

main :

int i;

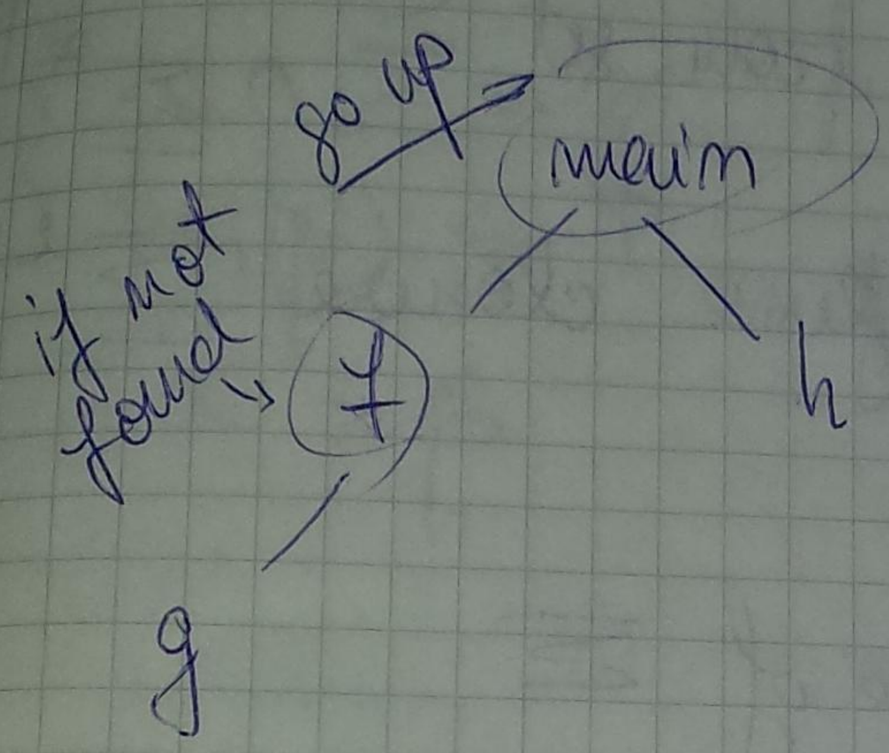
1. ST → corresponding to each module

ST → main

ST → f

ST → g

ST → h



visibility domain

Introduction to Formal Language

A man has a dog.

<sentence> → <noun-part> <verb part>

<noun part> → <art> <noun>

<art> → a

<art> → the

<art> → a | the

<noun> → man | dog | cat

~~<verb> → <verb> <object-part~~

<verb> → has <object-part>

<obj part> → <art> <noun>

correct syntactically but not semantically

alphabet = { a, the, man, dog, cat, has }
{ <sentence>, <nounpart>, <noun>, ... }

Def A formal language L : $L \subseteq \Sigma^*$

is a subset of all possible combinations of zero or more elements from set Σ

Σ^* → transitive and reflexive closure of Σ

Σ^+ → transitive closure of Σ

$\varepsilon \in \Sigma^0$ = empty word (λ)

Σ^* - all epsilon

Σ^2 - pairs

Def 2 A grammar is a $G = (N, \Sigma, P, S)$

where : - N - set of non terminal symbols, $|N| < \infty$
- Σ - set of terminal symbols, alphabet, $|\Sigma| < \infty$
- P - set of productions

$P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

- S - $S \in N$ - starting symbol

non terminal symbol

Ex. $\Sigma = \{a, the \ldots\}$

$N = \{\langle sentence \rangle, \langle pre\text{-}pos \rangle \ldots\}$

$P = \langle sentence \rangle \rightarrow \langle noun\text{-}phrase \rangle \langle verb \rangle$

$S = \langle sentence \rangle$

<u>Remarks:</u>

1.) $N \cap \Sigma = \emptyset$

2.) $(\alpha, \beta) \in P$ production rule

$\alpha \rightarrow \beta$ rule