

Tehnici de compilare curs ①

Limboje de calculator

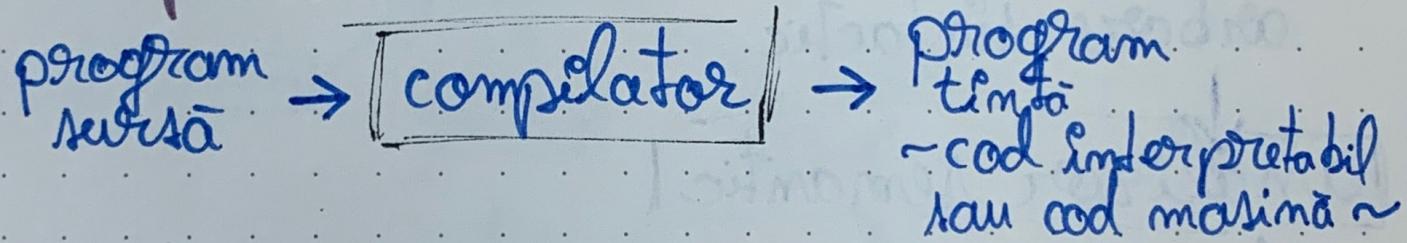
① Limboj cod maximă (de nivel 0)
- acrise în binar

② Limboje de aramblare (de nivel 1)
- adresările registerii
- mnemonice pt instructiuni
* ADD 20 00

③ Limboje de nivel înalt (nivel 2)

④ Limboje specializate

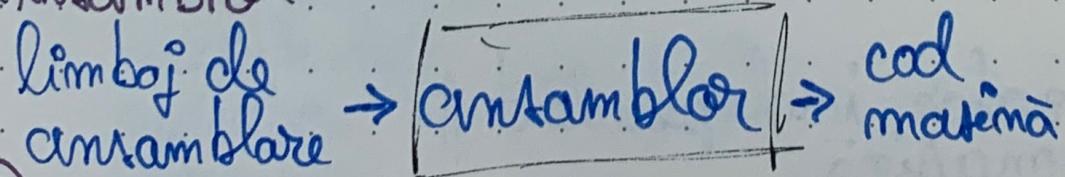
Compilatoare



Interpretor

- interpretează instrucțiunile pe rând și le executa direct
- * nu compilează
- * rulare mai lentă

Arsamblor



Preprocessor

- Linker
- Loader

Fazele compilării

Nos de caractere

Preprocessor

se modifica

anализатор лексический (scanner)

~ face analiza
lexicala

Mr. de Jokemuri

↓ analizor sintactic (parțial)

arborescens *infanticus*

Analizor Semantic

arbore modificat

✓ generador de cod intermedio

cod intermedior

Optimizator independent de masină

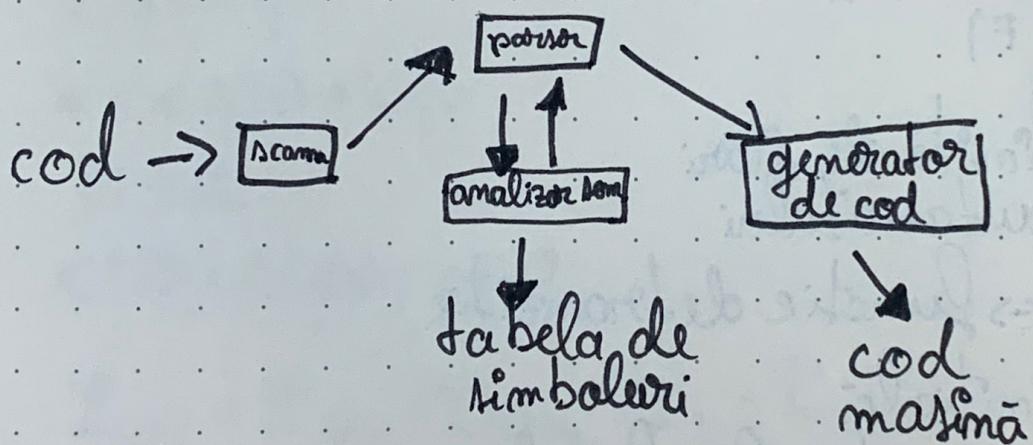
↓
cod intermediate optimization

generator de cod

cod matematică

optimizator de cod
dependent de masină

program scris în
cod matematic optimizat



Analiza lexicală

- constă în transformarea sirului de caractere corespunzător programului într-un sir de tokeni
ex: identificatori

cuvinte cheie
operatori
comenziorii
separatori

tokenuri

- expresii exprimatici
expresii regulate
expresie regulată peste Σ

∅ exp. reg.: peste Σ care descrie lb ∅ {a}{a}

$\forall a \in \Sigma$

Fie $p \neq q$ exp. reg. peste Σ care deserviu
 $L(p), L(q) \in \Sigma^*$

$$p \sqcup q \rightarrow L(p) \cup L(q)$$

$$p \cdot q \rightarrow L(p)L(q)$$

$$p^* \rightarrow L(p^*)$$

$$L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$

$$L \in \Sigma^*, L^* = \{\lambda\} \cup \{L \cup L \cdot L \cup L \cdot L \cdot L \dots\}$$

Automate finite deterministic

$$A = (Q, \Sigma, \delta, q_0, F)$$

Q - multime finita de stari

Σ - alfabetul automotului

$\delta: Q \times \Sigma \rightarrow Q$ - functie de transitiie

$q_0 \in Q$ stare initiala

$F \subseteq Q$ multimea stariilor finale

$$L(A) = \{w \in \Sigma^* \mid (q_0, w) \xrightarrow{*} (q, \lambda), q \in F\}$$

$$(p, aw) \xrightarrow{*} (q, w) \text{ daca } q = \delta(p, a)$$

Nedeterministe

$$\delta: Q \times \Sigma \rightarrow \overline{2^Q}$$

$$(p, aw) \xrightarrow{*} (q, w) \text{ daca } q \in \delta(p, a)$$

neeterminate cu 2 tranziții

$$\delta: Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$$

$(p, a) \xrightarrow{} (q, w)$ dacă $q \in \delta(p, a)$
 $a \in \Sigma \cup \{\lambda\}$

pt $q \in Q$ notăm $\langle q \rangle = \{ \Delta \in Q \mid \exists p_0..p_m \in Q, p_0 = q, P_m = \Delta$
 $P_i \in \delta(P_{i-1}, \lambda), \text{ pt } i = \overline{1, m}$
 $\wedge m \geq 0 \}$

* $q \in \langle q \rangle$

* $\langle \langle q \rangle \rangle = \langle q \rangle$

$M \subseteq Q$

$\langle M \rangle = \bigcup \langle q \rangle, q \in Q$

$$L_{AFD} = L_{AFN} = L_{AFN_2} = L_3 = L_{ExpReg}$$

Analizor lexical

- este o subrutină
- scanarea programului cursă
- face peste comentarii, spații, etc.
- furnizează tokenul curent
- semnalază eventuale erori lexice

Deroriere lexicală

- numim deroriere lexicală peste Σ ExpReg de forma
 $e(e_1 | e_2 | \dots | e_m)^*$ unde $e_i \in \text{ExpReg}$ peste Σ

$\{e_i\}$ reprezintă o categorie de tokeni
 $\{\Sigma\}$ multimea caracterelor admise în limbaj

Def Fie $w \in L(e)$

numerean interpretarea lui w un sir de forma

$(w_1, i_1) \dots (w_t, i_t)$

cu $w = w_1 w_2 \dots w_t$ și $w_j \in L(e_{ij})$, $1 \leq i_j \leq m$,
 $1 \leq j \leq t$

Spunem că e este ambiguă dacă

$w \in L(e)$ care are minimum 2 interpretări

Spunem că o derivare lexicală

$e = (e_1 | e_2 | \dots | e_m)^*$ este orientată dreapta dacă pt
v $w \in L(e)$ nu poate fi orice interpretare

$(w_1, i_1) \dots (w_t, i_t)$ unde este cel mai lung prefix al
sirului w_j, w_{j+1}, \dots, w_t

Teorema

Se poate decide algoritmice dacă o anumită
derivare lexicală este orientată dreapta.

Implementare Scanner

1) ExpReg \rightarrow ANF₂ \rightarrow AFD

2) ExpReg \rightarrow AFD

Notă:

50% Lab - 2 proiecte

examen cu cartile permisă cu ex
similară ca cele de la seminar

examen 50%

seminar are bonus la rezolvare