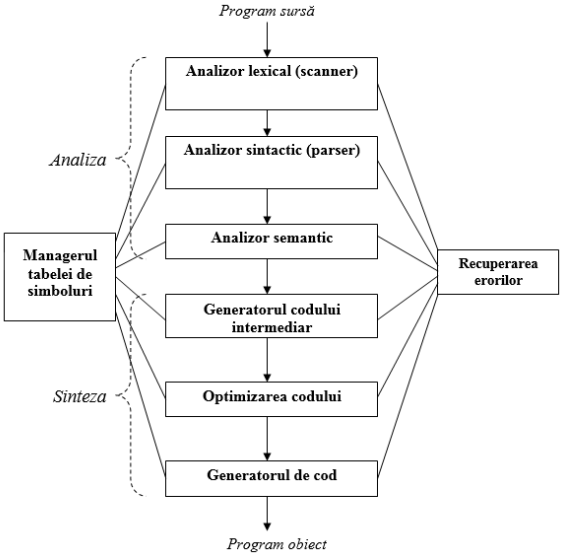
Curs Limbaje formale și compilatoare
Analiza lexicală

Universitatea *Transilvania* din Brașov
Facultatea de Matematică și Informatică

2021/2022

Etapele compilării



Analiza lexicală:

- prima fază a unui compilator

Analiza lexicală:

- prima fază a unui compilator
- citește caracterele programului sursă și produce la ieșire un șir de **token**-i

Analiza lexicală:

- prima fază a unui compilator
- citește caracterele programului sursă și produce la ieșire un șir de **token**-i
- elimină spațiile albe și comentariile

Analiza lexicală:

- prima fază a unui compilator
- citește caracterele programului sursă și produce la ieșire un șir de **token**-i
- elimină spațiile albe și comentariile
- semnalează erorile lexicale

Analiza lexicală:

- prima fază a unui compilator
- citește caracterele programului sursă și produce la ieșire un șir de **token**-i
- elimină spațiile albe și comentariile
- semnalează erorile lexicale
- corelează mesajele de eroare cu codul sursă

Noțiuni de bază:

- **token**: unitate lexicală - elementul de bază în analiza sintactică - reprezintă o mulțime de șiruri de caractere din intrare, cu aceeași funcționalitate (exemplu: constantă numerică)

Noțiuni de bază:

- **token**: unitate lexicală - elementul de bază în analiza sintactică - reprezintă o mulțime de șiruri de caractere din intrare, cu aceeași funcționalitate (exemplu: constantă numerică)
- **lexemă**: un șir de caractere care compune o unitate lexicală (exemplu: 123.5)

Noțiuni de bază:

- **token**: unitate lexicală - elementul de bază în analiza sintactică - reprezintă o mulțime de șiruri de caractere din intrare, cu aceeași funcționalitate (exemplu: constantă numerică)
- **lexemă**: un șir de caractere care compune o unitate lexicală (exemplu: 123.5)
- **pattern**: o regulă care descrie mulțimea lexemelor ce reprezintă un token (exemplu: (cifra)(cifra)*(.)(cifra)*)

Exemple cuvintele cheie, operator relațional, identificatori, constante numerice, delimitatori, etc;

Token	**Lexemă**	**Pattern**
if	if	if
for	for	for
operator relațional	$<, >, <=, >=, !=, ==$	$<$ sau $>$ sau $<=$ sau $>=$ sau $!=$ sau $==$
identificator	alpha, contor, x1	litera urmată de litere și/sau cifre
șir de caractere	"mesaj"	secvență de caractere între "

Rezultatul analizei lexicale: - șir de perechi <token, atribut>

Rolul atributului: anumiți token-i reprezintă mulțimi de mai multe lexeme (exemplu: identificator). Sunt necesare informații suplimentare păstrate în câmpul **atribut**.

Observații:

- Tokenii sunt de obicei reprezentați prin constante numerice unice.

Observații:

- Tokenii sunt de obicei reprezentați prin constante numerice unice.
- Pentru tokeni cu un singur reprezentat nu este necesar un atribut.

Observații:

- Tokenii sunt de obicei reprezentați prin constante numerice unice.
- Pentru tokeni cu un singur reprezentat nu este necesar un atribut.
- Fiecare cuvânt cheie are un token separat

Observații:

- Tokenii sunt de obicei reprezentați prin constante numerice unice.
- Pentru tokeni cu un singur reprezentat nu este necesar un atribut.
- Fiecare cuvânt cheie are un token separat
- Unitățile care au rol diferit la analiza sintactică au token separat (la analiza sintactică NU se utilizează atributul)

Observații:

- Tokenii sunt de obicei reprezentați prin constante numerice unice.
- Pentru tokeni cu un singur reprezentat nu este necesar un atribut.
- Fiecare cuvânt cheie are un token separat
- Unitățile care au rol diferit la analiza sintactică au token separat (la analiza sintactică NU se utilizează atributul)
- Operatorii aritmetici fie au fiecare token separat, fie sunt grupați după precendență.

Observații:

- Tokenii sunt de obicei reprezentați prin constante numerice unice.
- Pentru tokeni cu un singur reprezentat nu este necesar un atribut.
- Fiecare cuvânt cheie are un token separat
- Unitățile care au rol diferit la analiza sintactică au token separat (la analiza sintactică NU se utilizează atributul)
- Operatorii aritmetici fie au fiecare token separat, fie sunt grupați după precendență.
- Parserele au metode de punere în evidență a precendenței

Observații:

- Tokenii sunt de obicei reprezentați prin constante numerice unice.
- Pentru tokeni cu un singur reprezentat nu este necesar un atribut.
- Fiecare cuvânt cheie are un token separat
- Unitățile care au rol diferit la analiza sintactică au token separat (la analiza sintactică NU se utilizează atributul)
- Operatorii aritmetici fie au fiecare token separat, fie sunt grupați după precendență.
- Parserele au metode de punere în evidență a precendenței
- În cazul identificatorilor / constantelor numerice atributul = un pointer în tabela de simboluri / constante numerice

Observații:

- Tokenii sunt de obicei reprezentați prin constante numerice unice.
- Pentru tokeni cu un singur reprezentat nu este necesar un atribut.
- Fiecare cuvânt cheie are un token separat
- Unitățile care au rol diferit la analiza sintactică au token separat (la analiza sintactică NU se utilizează atributul)
- Operatorii aritmetici fie au fiecare token separat, fie sunt grupați după precendență.
- Parserele au metode de punere în evidență a precendenței
- În cazul identificatorilor / constantelor numerice atributul = un pointer în tabela de simboluri / constante numerice
- Comentariile, blank-urile, tab-urile, newline-urile se neglijează

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale:

Analiza lexicală - Exemplu

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _),

Analiza lexicală - Exemplu

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main),

Analiza lexicală - Exemplu

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _),

Analiza lexicală - Exemplu

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _), (token_PI, _),

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _), (token_PI, _), (token_BID, _),

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _), (token_PI, _), (token_BID, _), (token_INT, _),

Analiza lexicală - Exemplu

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _), (token_PI, _), (token_BID, _), (token_INT, _), (token_ID, pointer_la_x), (token_VG,_), (token_ID, pointer_la_y), (token_VG,_), (token_ID, pointer_la_min), (token_PV,_),

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _), (token_PI, _), (token_BID, _), (token_INT, _), (token_ID, pointer_la_x), (token_VG,_), (token_ID, pointer_la_y), (token_VG,_), (token_ID, pointer_la_min), (token_PV,_), (token_ID, pointer_la_x), (token_ATR, _), (token_NUM, 3), (token_PV, _),

Analiza lexicală - Exemplu

Programul sursă:

```
int main(){
    int x, y, min;
    x = 3;
    y = 4;
    if(x < y)
        min = x;
    else
        min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _), (token_PI, _), (token_BID, _), (token_INT, _), (token_ID, pointer_la_x), (token_VG,_), (token_ID, pointer_la_y), (token_VG,_), (token_ID, pointer_la_min), (token_PV,_), (token_ID, pointer_la_x), (token_ATR, _), (token_NUM, 3), (token_PV, _), (token_ID, pointer_la_y), (token_ATR, _), (token_NUM, 4), (token_PV, _),

Analiza lexicală - Exemplu

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _), (token_PI, _), (token_BID, _), (token_INT, _), (token_ID, pointer_la_x), (token_VG,_), (token_ID, pointer_la_y), (token_VG,_), (token_ID, pointer_la_min), (token_PV,_), (token_ID, pointer_la_x), (token_ATR, _), (token_NUM, 3), (token_PV, _), (token_ID, pointer_la_y), (token_ATR, _), (token_NUM, 4), (token_PV, _),(token_IF, _), (token_PD, _), (token_ID, pointer_la_x), (token_OPR, LS), (token_ID, pointer_la_y), (token_PI, _),

Analiza lexicală - Exemplu

Programul sursă:

```
int main(){
      int x, y, min;
      x = 3;
      y = 4;
      if(x < y)
            min = x;
       else
            min = y;
}
```

Rezultatul analizei lexicale: (token_INT, _), (token_ID, pointer_la_main), (token_PD, _), (token_PI, _), (token_BID, _), (token_INT, _), (token_ID, pointer_la_x), (token_VG,_), (token_ID, pointer_la_y), (token_VG,_), (token_ID, pointer_la_min), (token_PV,_), (token_ID, pointer_la_x), (token_ATR, _), (token_NUM, 3), (token_PV, _), (token_ID, pointer_la_y), (token_ATR, _), (token_NUM, 4), (token_PV, _),(token_IF, _), (token_PD, _), (token_ID, pointer_la_x), (token_OPR, LS), (token_ID, pointer_la_y), (token_PI, _), (token_ID, pointer_la_min), (token_ATR, _), (token_ID, pointer_la_x), (token_PV, _), (token_ELSE, _), (token_ID, pointer_la_min), (token_ATR, _), (token_ID, pointer_la_y), (token_PV, _), (token_BII)

Analiza lexicală - Citirea codului sursă

Există trei metode de abordare a implementării unui analizor lexical:

1. Utilizarea unui generator de analizor lexical - ex LEX, QUEX, ANTLR

Analiza lexicală - Citirea codului sursă

Există trei metode de abordare a implementării unui analizor lexical:

1. Utilizarea unui generator de analizor lexical - ex LEX, QUEX, ANTLR
2. Scrierea analizorului lexical într-un limbaj de programare, utilizând facilitățile de intrare/ieșire

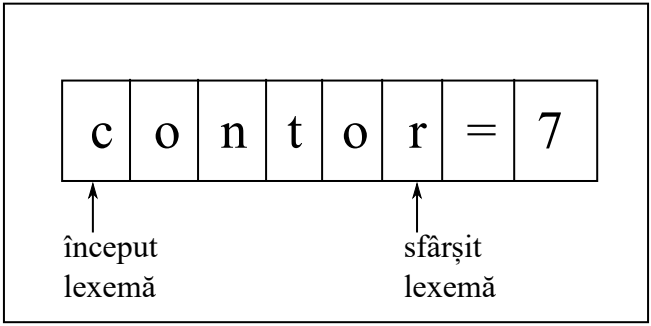
Analiza lexicală - Citirea codului sursă

Există trei metode de abordare a implementării unui analizor lexical:

1. Utilizarea unui generator de analizor lexical - ex LEX, QUEX, ANTLR
2. Scrierea analizorului lexical într-un limbaj de programare, utilizând facilitățile de intrare/ieșire
3. Scrierea analizorului lexical în limbaj de programare, cu gestionarea manuală a intrărilor și ieșirilor

Buffer-ul de intrare

Observație: există o serie de momente când analizorul lexical trebuie să privească înainte (*look ahead*) un număr de caractere pentru a determina lexema.

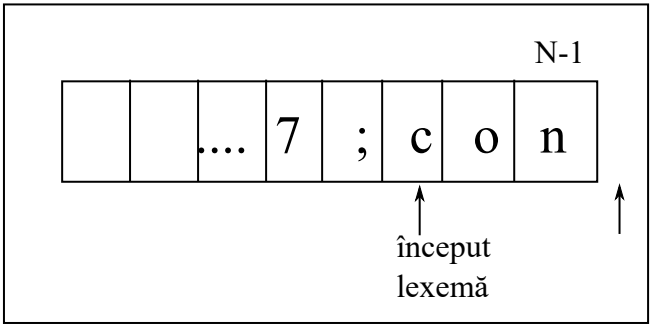


$\Rightarrow$ este necesar un *buffer* de intrare în care se încarcă un număr N de caractere din fișierul codului sursă.

Buffer-ul de intrare

```
{
    //..cod sursa
    x = x + 7;
    contor = 15;
    //..cod sursa
}
```

Problemă: o lexemă nu a mai avut complet loc în buffer. Din codul alăturat, din identificatorul *contor* nu au încăput la finalul buffer-ului decât primele 3 caractere.



Ce facem?

Buffer-ul de intrare

- Citirea întregului cod sursă dintr-o dată - Problematic

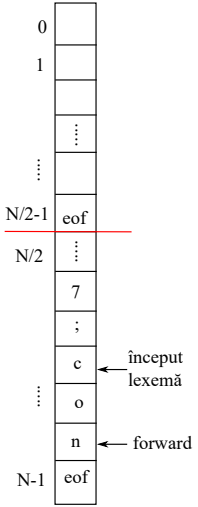
Buffer-ul de intrare

- Citirea întregului cod sursă dintr-o dată - Problematic
- Citirea unui rând întreg din fișier, până la întâlnirea caracterului *endline* - doar dacă limbajul analizat nu permite ruprea lexemelor pe mai multe rânduri

Buffer-ul de intrare

- Citirea întregului cod sursă dintr-o dată - Problematic
- Citirea unui rând întreg din fișier, până la întâlnirea caracterului *endline* - doar dacă limbajul analizat nu permite ruprea lexemelor pe mai multe rânduri
- Folosirea unui buffer de dimensiune N, partajat în două jumătăți folosind santinele.

Buffer-ul de intrare partajat în două jumătăți:



Algoritm 1: Managementul buffer-ului de intrare

```
//avansez la caracterul urmator in intrare
forward ← forward + 1
daca forward = eof atunci
    daca forward la sfarsitul primei jumatati atunci
        reincarca a doua jumatate
        forward ← forward + 1
    sfarsit_daca
    altfel
        daca forward la sf. celei de-a doua jumatati atunci
            reincarca prima jumatate
            muta curent la inceputul buffer-ului
        sfarsit_daca
        altfel
            //am ajuns la finalul codului sursa return
        sfarsit_daca
    sfarsit_daca
sfarsit_daca
```

Erori lexicale

Există foarte puține erori la faza de analiză lexicală:

- Caractere care nu fac parte din vocabularul permis

Erori lexicale

Există foarte puține erori la faza de analiză lexicală:

- Caractere care nu fac parte din vocabularul permis
- Comentarii care nu au fost închise

Erori lexicale

Există foarte puține erori la faza de analiză lexicală:

- Caractere care nu fac parte din vocabularul permis
- Comentarii care nu au fost închise
- Șiruri de caractere care nu au fost închise

Erori lexicale

Există foarte puține erori la faza de analiză lexicală:

- Caractere care nu fac parte din vocabularul permis
- Comentarii care nu au fost închise
- Șiruri de caractere care nu au fost închise
- Șiruri de caractere, identificatori, constante numerice prea lungi

Erori lexicale

Există foarte puține erori la faza de analiză lexicală:

- Caractere care nu fac parte din vocabularul permis
- Comentarii care nu au fost închise
- Șiruri de caractere care nu au fost închise
- Șiruri de caractere, identificatori, constante numerice prea lungi
- Șiruri de caractere care se întind pe mai multe rânduri

Modul de tratare a erorilor

- Eliminarea unui caracter din intrare

Modul de tratare a erorilor

- Eliminarea unui caracter din intrare
- Înlocuirea unui caracter cu altul

Modul de tratare a erorilor

- Eliminarea unui caracter din intrare
- Înlocuirea unui caracter cu altul
- Generare de token eroare

Modul de tratare a erorilor

- Eliminarea unui caracter din intrare
- Înlocuirea unui caracter cu altul
- Generare de token eroare
- Modul panică: se elimină din intrare toate caracterele, până la găsirea unui token valid.

Expresii regulate: O expresie regulată se poate defini recursiv prin:

1. $\emptyset$ este o expresie regulată care reprezintă limbajul vid
2. λ este o expresie regulată care reprezintă limbajul $L = \{\lambda\}$
3. Dacă $a \in \Sigma$ atunci a este o expresie regulată care reprezintă limbajul $L = \{a\}$
4. Dacă r și s sunt expresii regulate care reprezintă limbajele $L(r)$ și respectiv $L(s)$ atunci următoarele expresii sunt regulate:
   a. **alternarea** $(r)|(s)$ reprezintă limbajul $L(r) \cup L(s)$
   b. **concatenarea** $(r)(s)$ reprezintă limbajul $L(r)L(s)$,
   c. **închiderea Kleene** r^* reprezintă limbajul $L(r)^*$.

Expresii regulate - Exemple: peste mulțimea literelor și cifrelor

$(a|b|c|d|e|....|z|A|B|C|...|Z)(a|b|c|d|e|....|z|A|B|C|...|Z|0|1|2|3|...|9)^*$ reprezentând mulțimea identificatorilor

$(1|2|3|4|5|6...|9)(0|1|2|3|4|5|6...|9)^*|0$ reprezentând mulțimea numerelor naturale

Observații:

- în expresiile de mai sus "..." sunt în locul resului de litere respectiv cifre, pentru a simplifica scrierea
- se observă că expresiile devin foarte lungi și greu de scris / citit $\Rightarrow$ pot fi înlocuite prin șiruri de *definiții regulate*

Definiții regulate. Dacă Σ alfabet, atunci un șir de definiții regulate este:

$$\begin{aligned} d_1 &\to r_1 \\ d_2 &\to r_2 \\ &\dots \\ d_n &\to r_n \end{aligned}$$

r_i expresie regulată peste $\Sigma \cup \{d_1, d_2, ..., d_{i-1}\}$.

Definiții regulate. Dacă Σ alfabet, atunci un șir de definiții regulate este:

$$\begin{aligned} d_1 &\to r_1 \\ d_2 &\to r_2 \\ &\dots \\ d_n &\to r_n \end{aligned}$$

r_i expresie regulată peste $\Sigma \cup \{d_1, d_2, ..., d_{i-1}\}$.

Exemplu:

$$<\text{litera mare}> \to A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z$$

Definiții regulate. Dacă Σ alfabet, atunci un șir de definiții regulate este:

$$d_1 \to r_1$$
$$d_2 \to r_2$$
$$\dots$$
$$d_n \to r_n$$

r_i expresie regulată peste $\Sigma \cup \{d_1, d_2, ..., d_{i-1}\}$.

Exemplu:

<litera mare>$\to A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z$

<litera mica>$\to a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$

Definiții regulate. Dacă Σ alfabet, atunci un șir de definiții regulate este:

$$d_1 \to r_1$$
$$d_2 \to r_2$$
$$\dots$$
$$d_n \to r_n$$

r_i expresie regulată peste $\Sigma \cup \{d_1, d_2, ..., d_{i-1}\}$.

Exemplu:

<litera mare>$\to A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z$

<litera mica>$\to a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$

<litera>$\to$<litera mare> | <litera mica>

Definiții regulate. Dacă Σ alfabet, atunci un șir de definiții regulate este:

$$d_1 \to r_1$$
$$d_2 \to r_2$$
$$\dots$$
$$d_n \to r_n$$

r_i expresie regulată peste $\Sigma \cup \{d_1, d_2, ..., d_{i-1}\}$.

Exemplu:

<litera mare>$\to A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z$

<litera mica>$\to a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$

<litera>$\to$<litera mare> | <litera mica>

<cifra>$\to 0|1|2|3|4|5|6|7|8|9$

Specificarea token-ilor

Definiții regulate. Dacă Σ alfabet, atunci un șir de definiții regulate este:

$$\begin{aligned} d_1 &\to r_1 \\ d_2 &\to r_2 \\ &\dots \\ d_n &\to r_n \end{aligned}$$

r_i expresie regulată peste $\Sigma \cup \{d_1, d_2, ..., d_{i-1}\}$.

Exemplu:

<litera mare>$\to A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z$

<litera mica>$\to a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$

<litera>$\to$<litera mare> | <litera mica>

<cifra>$\to 0|1|2|3|4|5|6|7|8|9$

<id>$\to$<litera> (<litera> | <cifra>)*

Definiții regulate. Dacă Σ alfabet, atunci un șir de definiții regulate este:

$$d_1 \to r_1$$
$$d_2 \to r_2$$
$$\dots$$
$$d_n \to r_n$$

r_i expresie regulată peste $\Sigma \cup \{d_1, d_2, ..., d_{i-1}\}$.

Exemplu:

<litera mare>$\to A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z$

<litera mica>$\to a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$

<litera>$\to$<litera mare> | <litera mica>

<cifra>$\to 0|1|2|3|4|5|6|7|8|9$

<id>$\to$<litera> (<litera> | <cifra>)*

<numar real>$\to$<cifra>$^+$ (. <cifra>$^+$)?(E$(+|-)$? <cifra>$^+$)?

Recunoașterea token-ilor - cu ajutorul unei funcții **nexttoken()** care returnează o pereche $<$ token, atribut $>$.

Metode de recunoaștere

1. Metoda manuală: - pe baza caracterului de început se stabilește despre ce token ar putea fi vorba, apoi se identifică lexema.

Recunoașterea token-ilor - cu ajutorul unei funcții **nexttoken()** care returnează o pereche $<$ token, atribut $>$.

Metode de recunoaștere

1. Metoda manuală: - pe baza caracterului de început se stabilește despre ce token ar putea fi vorba, apoi se identifică lexema.

2. Metoda automatelor finite: pentru fiecare token se construiește un automat finit determinist, iar automatele se leagă între ele.

1. Metoda manuală - schițarea algoritmului

Algoritm 2: Funcția nexttoken

Intrare: buffer-ul de intrare, poziția începutului lexemei

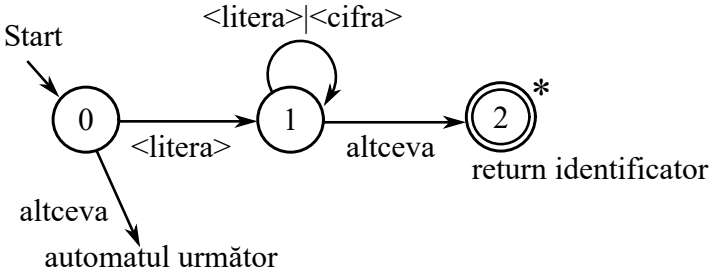
Iesire: perechea <token, atribut> identificată

```
daca caract_curent este litera atunci
    return getIdentificator()
sfarsit_daca
altfel
    daca caract_curent este cifra atunci
        return getConstataNumerica()
    sfarsit_daca
    altfel
        daca caract_curent ='<' atunci
            return getOpRelational() i
        sfarsit_daca
        altfel
            //... alte opțiuni
        sfarsit_daca
```

1. Metoda manuală - Observații

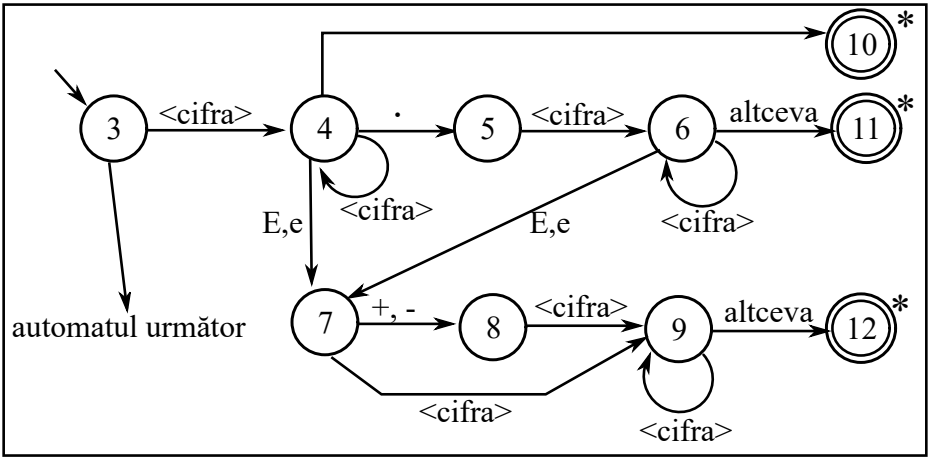
- ușor de scris odată ce s-au stabilit *pattern*-urile token-ilor
- greu de modificat, dacă se dorește modificarea limbajului
- problematic, dacă există mai mulți tokeni, pentru care o lexemă poate începe cu același caracter / secvență de caractere

2. Metoda automatelor finite - pentru fiecare token se construiește un automat finit.
Exemplu - automat pentru identificatori

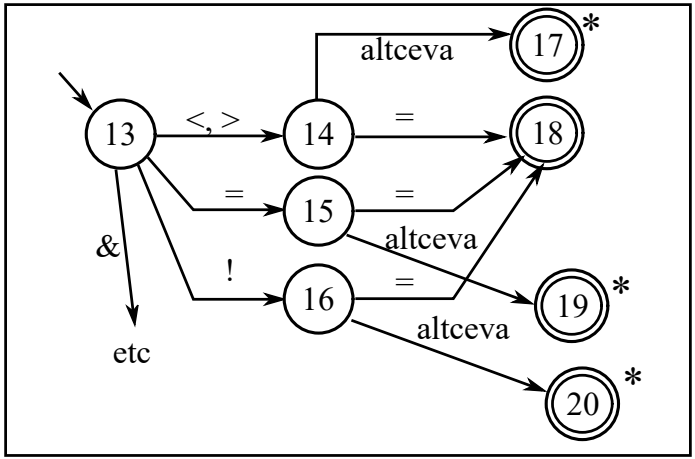


2. Metoda automatelor finite - pentru fiecare token se construiește un automat finit.

Exemplu - automat pentru constante numerice



2. Metoda automatelor finite - pentru fiecare token se construiește un automat finit.
Exemplu - automat pentru operatori (parțial)



Utilizarea generatoarelor de analizator lexical

Limbaju LEX - Cursul următor.