

Limbaje formale și tehnici de compilare

Curs 5: implementarea diagramei de tranziții;
implementare folosind stări implicite

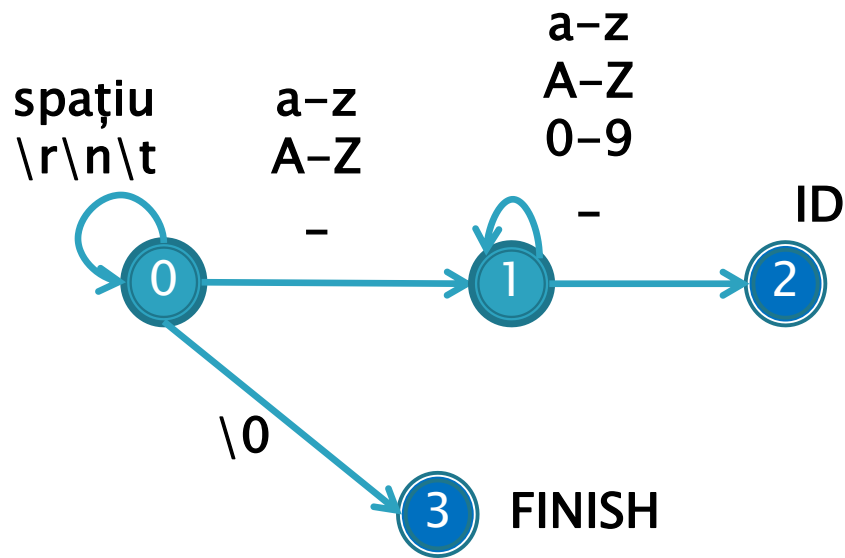
Sarcini la implementarea DT

- ▶ Extragerea atomilor lexicali
- ▶ Preluarea informațiilor lexicale asociate: caracterele din care sunt constituiți identificatorii sau șirurile de caractere, valorile numerice ale numerelor, ...
- ▶ Identificarea cuvintelor cheie
- ▶ Setarea în atomi a informațiilor de localizare: linie, coloană, nume fișier sursă

```
// aria cercului
int main(){
    double r;           // raza
    printf("raza: ");
    scanf("%g",&r);
    printf("aria=%g\n",3.14*r*r);
    return 0;
}
```

```
TYPE_INT, ID:main, LPAR, RPAR, LACC
TYPE_DOUBLE, ID:r, SEMICOLON
ID:printf, LPAR, STR:"raza: ", RPAR,
SEMICOLON ID:scanf, LPAR, STR:"%g",
COMMA, AND, ID:r, RPAR, SEMICOLON
ID:printf, LPAR, STR:"aria=%g\n",
COMMA, DOUBLE:3.14, MUL, ID:r, MUL,
ID:r, RPAR, SEMICOLON RETURN, INT:0,
SEMICOLON RACC
```

DT de implementat



ID = [a-zA-Z_] [a-zA-Z0-9_]*

// cuvinte cheie

IF = 'if'

ELSE = 'else'

FINISH = '\0'

SKIP = [\r\n\t] // se elimină

```
enum{ID, IF, ELSE, FINISH};
typedef struct{
    int cod;
    int linie;
    char text[TEXT_MAX];
}Atom;
Atom atomi[ATOMI_MAX];
int nAtomi=0;
```

Extragerea atomilor lexicali

- ▶ Se poate face:
 - ▶ **Simultan cu citirea caracterelor de intrare** (codul sursă)
 - necesarul de memorie este mai mic; se pot produce rezultate înainte de terminarea sursei (ex: când se procesează date din rețea)
 - ▶ **După citirea caracterelor într-un buffer** – se poate integra mai ușor cu alte module, deoarece este independentă de proveniența codului; în general este mai rapidă
- ▶ Extragerea se poate face folosind o funcție care extrage câte un atom la fiecare apel (**getNextToken**) sau se pot extrage toți atomii cu un singur apel (**getTokens**)

Algoritm de extragere – parte comună

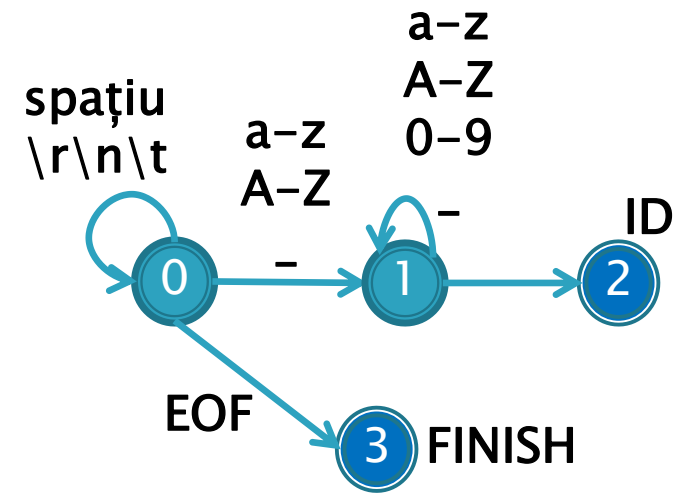
- ▶ În oricare dintre cele două situații, extragerea unui nou atom începe întotdeauna din starea 0 a DT
- ▶ La fiecare caracter nou, se testează toate tranzițiile din starea curentă și, dacă există o tranziție care poate consuma caracterul citit, se urmează această tranziție, schimbându-se starea curentă. Caracterul se va consuma.
- ▶ Dacă din starea curentă nu există nicio tranziție care consumă caracterul curent, dar există o tranziție epsilon (de tip **else**), se va urma această tranziție în starea următoare, fără a se consuma caracterul curent
- ▶ Dacă din starea curentă nu există nicio tranziție care consumă caracterul curent și nu există nici tranziție de tip **else**, se raportează eroare
- ▶ Când se ajunge într-o stare finală, se adaugă atomul nou citit în lista de atomi și se revine în starea 0

Extragere simultan cu citirea

- ▶ Se consideră o variabilă globală **oldCh** (*old char*) în care se memorează caracterul, dacă la pasul anterior acesta nu s-a consumat (imediat anterior a fost o tranziție **else** sau o stare finală). În caz că s-a consumat caracterul, **oldCh** va avea o valoare care nu poate fi un caracter (ex: -1)
- ▶ Dacă **oldCh** nu conține un caracter, se citește un caracter de la intrare, altfel se preia caracterul din **oldCh** și se marchează **oldCh** ca fiind gol (-1)
- ▶ Dacă nu se consumă caracterul curent, acesta se va stoca în **oldCh**

Extragere simultan cu citirea. Exemplu

```
int oldCh=-1;    // -1 => nu există caracter memorat
int getNextTk(){
    int s=0;      // starea curentă
    int ch;       // caracterul curent
    for(;;){      // buclă infinită
        if(oldCh== -1){ch=fgetc(fis);}else{ch=oldCh;oldCh=-1;}
        switch(s){
            case 0:if(isalpha(ch)||ch=='_'){s=1;}
                    else if(ch==EOF){s=3;oldCh=ch;}
                    else if(ch==' '||c=='\r'||ch=='\n'||ch=='\t'){
                        err("caracter invalid");
                        break;
                    }
            case 1:if(isalnum(ch)||ch=='_'){
                    }
                    else {s=2;oldCh=ch;}
                    break;
            case 2:oldCh=ch;
                    addTk(ID);
                    return ID;
            case 3:oldCh=ch;
                    addTk(FINISH);
                    return FINISH;
            default:err("stare invalidă %d cu caracterul %c (%d)",s,ch,ch);
        }
    }
}
```

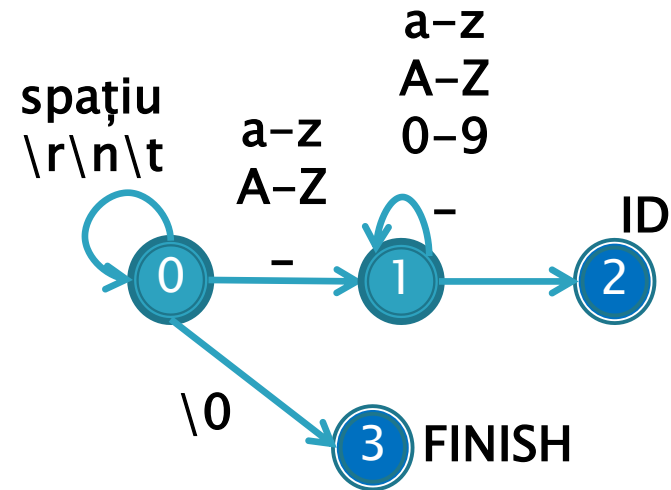


Extragere după citire

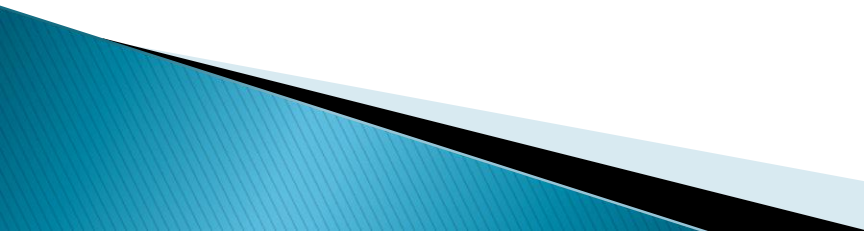
- ▶ Considerăm că toate caracterele de intrare sunt într-un buffer și că există un cursor în interiorul bufferului, poziționat pe caracterul curent
- ▶ Cursorul poate fi implementat ca un pointer, ca un iterator sau ca un index într-un vector de caractere
- ▶ La fiecare pas, se preia caracterul de la poziția cursorului
- ▶ Dacă se consumă caracterul, se incrementează cursorul, altfel cursorul rămâne pe poziția curentă

Extragere după citire. Exemplu

```
char *pch;           // cursorul, setat inițial pe începutul bufferului
int getNextTk(){
    int s=0;          // starea curentă
    int ch;           // caracterul curent
    for(;;){          // buclă infinită
        ch=*pch;       // se preia caracterul curent
        switch(s){
            case 0:if(isalpha(ch)||ch=='_'){s=1;pch++;}
                    else if(ch=='\0'){s=3;}
                    else if(ch==' '||c=='\r'||ch=='\n'||ch=='\t'){pch++;}
                    else err("caracter invalid");
                    break;
            case 1:if(isalnum(ch)||ch=='_'){pch++;}
                    else {s=2;}
                    break;
            case 2:addTk(ID);
                    return ID;
            case 3:addTk(FINISH);
                    return FINISH;
            default:err("stare invalidă %d cu caracterul %c (%d)",s,ch,ch);
        }
    }
}
```



Preluarea informațiilor lexicale

- ▶ În cazul identificatorilor și a constantelor (numere, caractere, șiruri de caractere) este necesar să se preia literele/valorile lor din șirul de intrare
 - ▶ Se pot aplica două metode:
 1. Pe măsură ce se consumă caractere, acestea vor fi acumulate într-un buffer, care va fi prelucrat ulterior. Dacă bufferul este de dimensiune fixă, la adăugare de caractere trebuie efectuate verificări de depășire.
 2. Se setează un pointer/iterator/index la începutul atomului. La sfârșitul atomului se preiau toate caracterele dintre începutul atomului (setat anterior) și poziția curentă.
- 

Identificarea cuvintelor cheie

- ▶ Deoarece cuvintele cheie (keywords) trebuie tratate diferit de identificatorii obișnuiți, ele vor fi extrase ca atomi cu coduri specifice
- ▶ Se pot folosi două metode:
 1. După ce s-a extras un identificator, se va face verificarea dacă acesta este un cuvânt cheie. Dacă este, se va adăuga cuvântul cheie respectiv, altfel se va adăuga ca identificator. Există metode de a se optimiza găsirea cuvintelor cheie, de exemplu știindu-se lungimea identicatorului găsit, prima literă din el, etc, în așa fel încât să se reducă numărul de comparații necesare
 2. Pe DT se figurează inclusiv cuvintele cheie și se face eliminarea ambiguităților dintre identicatori și cuvinte cheie. Se asigură o identificare directă, rapidă a cuvintelor cheie, dar DT este mai complexă (acest proces se poate automatiza cu unelte specifice)

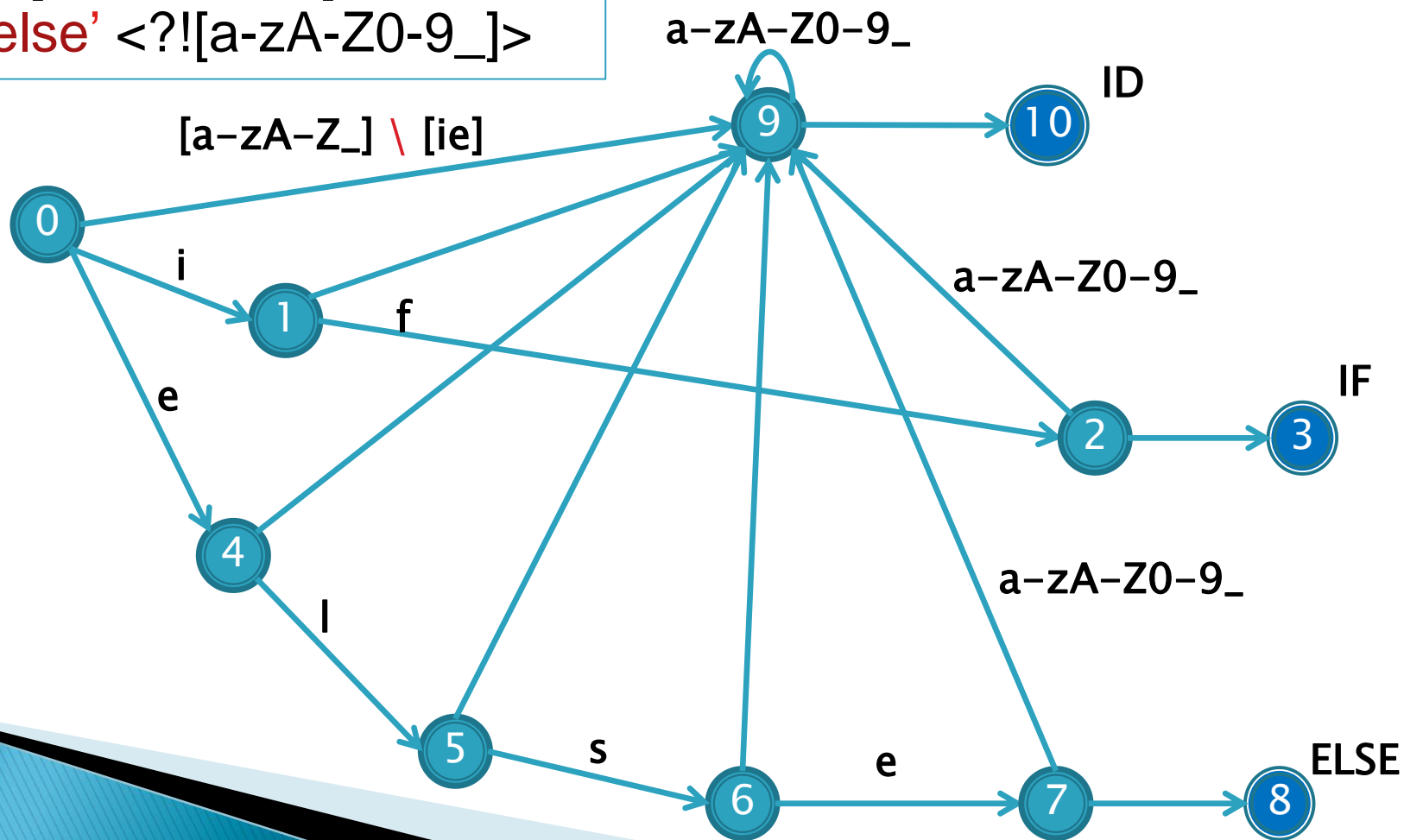
Figurarea cuvintelor cheie pe DT

$e_1 <?=e_2>$ - consumă e_1 doar dacă este urmată de e_2
 $e_1 <?!e_2>$ - consumă e_1 doar dacă nu este urmată de e_2

ID = $[a-zA-Z_][a-zA-Z0-9_]^*$

IF = 'if' $<?![a-zA-Z0-9_]>$

ELSE = 'else' $<?![a-zA-Z0-9_]>$



Setarea informațiilor de localizare (IL)

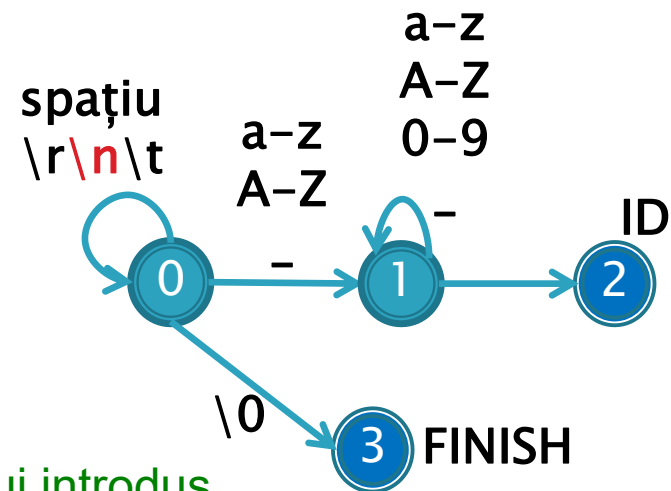
- ▶ Informațiile de localizare (nume fișier, linie, coloană) se pot prelua folosind variabile globale inițializate corespunzător
- ▶ Când se consumă caractere, se vor actualiza și IL (ex: când se întâlnește `\n`, se incrementează linia)
- ▶ Dacă LP permite ca un fișier să includă alt fișier (C/C++), se poate folosi o stivă de structuri IL. Când se include un fișier nou, IL curente se depun în stivă și se inițializează un nou set de IL pentru fișierul nou inclus. Când se revine în fișierul original, se scot de pe stivă IL salvate, astfel încât se reface poziția de la includere.
- ▶ Când se adaugă un nou atom, subrutina de adăgare a atomului (`addTk`) va prelua IL și le va seta în atomul nou adăugat

Exemplu

```
#define CHECKADD    if(nBuf==ID_MAX-1)err("depasire lungime id");else buf[nBuf++]=ch;
int linie=1;        // linia curentă
int getNextTk(){
    char buf[ID_MAX];
    int nBuf=0;

    .....
    case 0:if(isalpha(ch)||ch=='_'){s=1;pch++;buf[nBuf++]=ch;}
           else if(ch==' '||c=='\r'||ch=='\t'){pch++;}
           else if(ch=='\n'){linie++;pch++;}
           break;
    case 1:if(isalnum(ch)||ch=='_'){pch++;CHECKADD}
           else {s=2;}
           break;
    case 2:buf[nBuf]='\0';
           if(!strcmp(buf,"if")){addTk(IF);return IF;}
           if(!strcmp(buf,"else")){addTk(ELSE);return ELSE;}
           addTk(ID);           // addTk setează și linia atomului introdus
           setTkText(buf);      // setează câmpul text al ultimului atom introdus
           return ID;

    .....
}
```

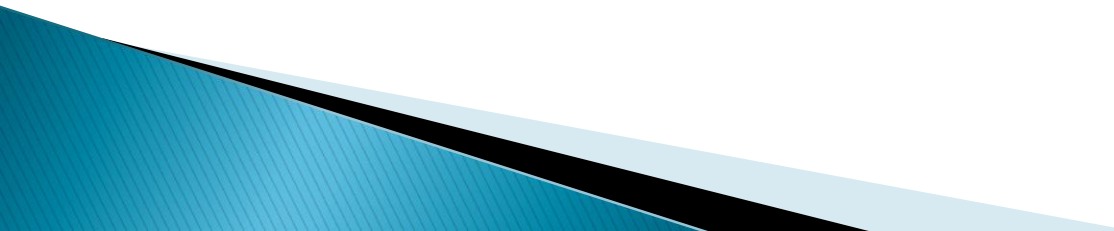


Implementarea folosind stări implicite

- ▶ Până acum, la implementarea ANLEX, stările din DT au fost codificate în mod explicit, folosind o instrucțiune **switch** cu câte un **case** pentru fiecare stare
- ▶ Se poate renunța la folosirea stărilor, dacă pentru fiecare definiție regulată (DR) se implementează în mod direct componentele ei, de la începutul și până la sfârșitul DR
- ▶ În acest fel, chiar poziția curentă în cod ne arată unde anume suntem în DR

```
// ID = [a-zA-Z_] [a-zA-Z0-9_]*  
ch=*pch;  
if(isalpha(ch)||ch=='_'){  
    ch=*++pch;  
    while(isalnum(ch)||ch=='_')ch=*++pch;  
    addTk(ID);  
    return ID;  
}
```

Avantajele acestei implementări

- ▶ Codul este mai compact și în general mai ușor de citit
 - ▶ În general programul este mai rapid, deoarece se elimină setările și testările stărilor
 - ▶ Se pot implementa direct multe expresii regulate pentru sarcini mai simple: introducere date utilizator, validare date, etc.
 - ▶ Diagrama de tranziții devine opțională și este necesară doar în cazuri cu DR/nedeterminări complexe. Pe măsură ce crește experiența programatorului, diagrama de tranziții se folosește din ce în ce mai rar.
- 

Algoritm de implementare

- ▶ Pentru DR/nedeterminări complexe, se face diagrama de tranziții. Pentru cazuri mai simple, se poate porni direct de la DR
- ▶ Pentru fiecare DR se implementează componentele acesteia, folosind construcții specifice de cod
- ▶ Opțional, secvențele de **if**-uri care testează caractere simple se pot înlocui cu o instrucțiune **switch**. Dacă între testele pentru caractere simple se află și teste mai complexe (ex: pentru intervale de caractere), acestea se pot muta pe ramura de **default** de la **switch**.
- ▶ Se adaugă în cod alte aspecte necesare, gen preluarea informațiilor lexicale asociate și a locației, recunoașterea cuvintelor cheie

Implementarea componentelor DR

```
// e
if(e){
  next
  ...
}
```

```
// e1 e2
if(e1){
  next
  if(e2){
    next
    ...
  }
}
```

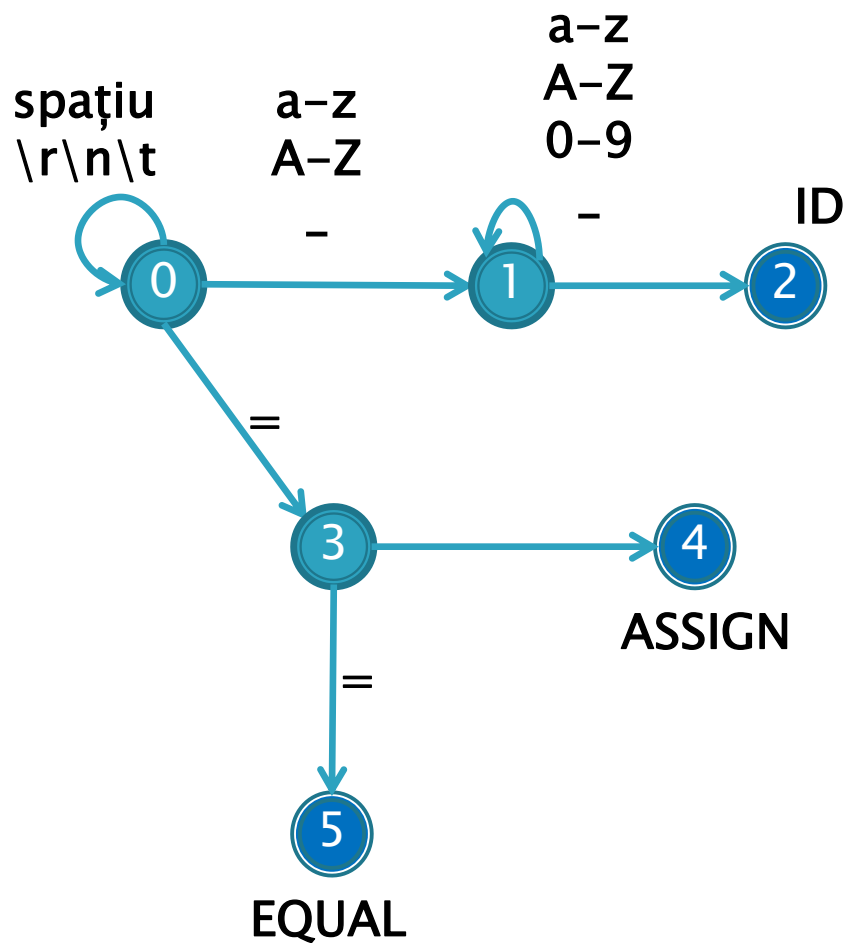
```
// e1 | e2
if(e1){
  next
  ...
}
else if(e2){
  next
  ...
}
```

```
// e*
while(e)next
...
```

```
// e?
if(e)next
...
```

- ▶ **next** – se trece la următorul caracter
- ▶ **...** – continuarea codului cu următoarele componente
- ▶ **e⁺ → e e^{*}**

Exemplu



ID = [a-zA-Z_] [a-zA-Z0-9_]*

ASSIGN = [=]

EQUAL = [=][=]

// cuvinte cheie

IF = 'if'

ELSE = 'else'

SKIP = [\r\n\t] // se elimină

Consumarea caracterelor

```
int getNextTk(){
    char ch=*pch;           // preluare caracter curent
    for(;;){
        if(isalpha(ch)||ch=='_'){
            ch=*++pch;           // next
            while(isalnum(ch)||ch=='_')ch=*++pch;
            addTk(ID);return ID;
        }
        if(ch=='='){
            ch=*++pch;
            if(ch=='='){
                pch++;           // nu mai este nevoie de setarea lui ch
                addTk(EQUAL);return EQUAL;
            }else{              // tranziția de tip else
                addTk(ASSIGN);return ASSIGN;
            }
        }
        if(ch==' '||ch=='\r'||ch=='\n'||ch=='\t')ch=*++pch; // rămâne în bucla for
    }
}
```

La consumarea unui caracter: dacă este nevoie ulterior de noul caracter, se actualizează și **ch**, altfel este suficient să se treacă la următorul (**pch++**)
pch – poziția curentă în șirul de intrare (variabilă globală)

Folosire switch

```
int getNextTk(){
    char ch=*pch;                // preluare caracter curent
    for(;;){
        switch(ch){
            case ' ':case '\r':case '\n':case '\t':ch=*++pch;break;
            case ';':pch++;addTk(SEMICOLON);return SEMICOLON;    // adăugat: SEMICOLON = ;
            case ':':pch++;addTk(COLON);return COLON;            // adăugat: COLON = :
            case '(':pch++;addTk(LPAR);return LPAR;              // adăugat: LPAR = \(
            case ')':pch++;addTk(RPAR);return RPAR;              // adăugat: RPAR =\)
            case '=':
                ch=*++pch;
                if(ch=='='){
                    pch++;
                    addTk(EQUAL);return EQUAL;
                }else{
                    addTk(ASSIGN);return ASSIGN;
                }
            default:
                if(isalpha(ch)||ch=='_'){
                    for(ch=*++pch;isalnum(ch)||ch=='_';ch=*++pch){}
                    addTk(ID);return ID;
                }
        }
    }
}
```

Tratarea erorilor

- ▶ În cod se pot adăuga teste pentru detectarea situațiilor de eroare, pentru ca acestea să poată fi afișate
- ▶ Se vor raporta erori în toate cazurile în care caracterul curent nu permite avansarea către sfârșitul unei DR
- ▶ Funcția **lexErr** (lexical error) folosită în cod, realizează următoarele acțiuni:
 - ▶ folosește informația de localizare curentă pentru a afișa poziția actuală a erorii
 - ▶ afișează mesajul de eroare, inclusiv cu valoarea unor posibile variabile, așa cum face **printf**
 - ▶ iese din program (nu se face revenire din eroare)

Cod care tratează erorile

```
switch(ch){
  case '\\':
    ch=*++pch;
    if(ch!='\\'){
      ch=*++pch;
      if(ch=='\\'){
        pch++;
        addTk(CHAR);return CHAR;
      }else lexErr("lipseste ' la sfarsitul constantei caracter");
    }else lexErr("constanta caracter vida");
  default:
    if(isdigit(ch)){
      for(ch=*++pch;isdigit(ch);ch=*++pch){}
      if(ch=='.'){
        ch=*++pch;
        if(isdigit(ch)){
          for(ch=*++pch;isdigit(ch);ch=*++pch){}
          addTk(REAL);return REAL;
        }else lexErr("dupa punctul zecimal trebuie sa fie minim un digit")
      }else{
        addTk(INT);return INT;
      }
    }
    else lexErr("caracter invalid: %c (ASCII %d)",ch,ch);
}
```

CHAR = \ ' [^]' \ '

INT = [0-9]+

REAL = [0-9]+ \ . [0-9]+

Implementarea altor sarcini lexicale

- ▶ Alte sarcini lexicale: preluarea informațiilor lexicale asociate, identificarea cuvintelor cheie, setarea informațiilor de localizare
- ▶ În principiu se implementează la fel ca la algoritmi cu stări explicite:
 - ▶ când se consumă caractere, acestea se salvează într-un buffer sau se marchează poziția de început și de sfârșit a lor în șirul de intrare
 - ▶ când se recunoaște un ID, înainte de a fi adăugat ca ID se verifică dacă este un cuvânt cheie
 - ▶ când se trece la linie nouă, se incrementează o variabilă globală

Cod care implementează și alte sarcini

```
int getNextTk(){
    char buf[ID_MAX], ch=*pch;
    int nBuf=0;                                // numărul de caractere din buf
    for(;;){
        switch(ch){
            case ' ':case '\r':case '\t':ch=*++pch;break;
            case '\n':                          // \n se tratează separat, pentru a se actualiza linia curentă
                ch=*++pch;
                linie++;
                break;
            default:
                if(isalpha(ch)||ch=='_'){
                    buf[nBuf++]=ch;
                    for(ch=*++pch;isalnum(ch)||ch=='_';ch=*++pch)CHECKADD
                    buf[nBuf]='\0';
                    if(!strcmp(buf,"if")){addTk(IF);return IF;}
                    if...                       // teste pentru toate cuvintele cheie
                    addTk(ID);
                    setTkText(buf);
                    return ID;
                }
            }
        }
    }
}
```