

# Matematička teorija računarstva

## Predavanja 14

Matko Botinčan  
PMF – Matematički odjel  
31.01.2007.

## Izvori materijala za ovo predavanje

- X. Jiao – CS 3240: Languages and Computation
  - [http://www-static.cc.gatech.edu/classes/AY2007/cs3240\\_fall/](http://www-static.cc.gatech.edu/classes/AY2007/cs3240_fall/)
- T. K. Prasad – CS 780: Compiler Design and Construction I
  - <http://www.cs.wright.edu/~tkprasad/courses/cs780/cs780.html>
- J. Riely – CSC347/447: Concepts of Programming Languages
  - <http://condor.depaul.edu/~jriely/csc447winter2007/index.html>
- D. Stotts – COMP 144: Programming Language Concepts
  - <http://www.cs.unc.edu/~stotts/COMP144/>
- Q. Yi – CS5363: Programming Languages and Compilers
  - <http://www.cs.utsa.edu/~qingyi/cs5363/>

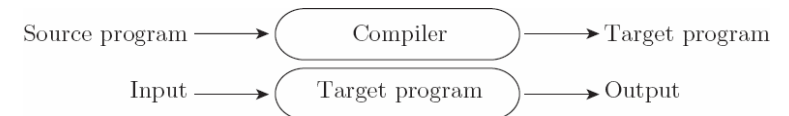
## Prevođenje programa

```
program gcd(input, output);  
var i, j: integer;  
begin  
    read(i, j);  
    while i <> j do  
        if i > j then i := i - j;  
        else j := j - i;  
    writeln(i)  
end.
```

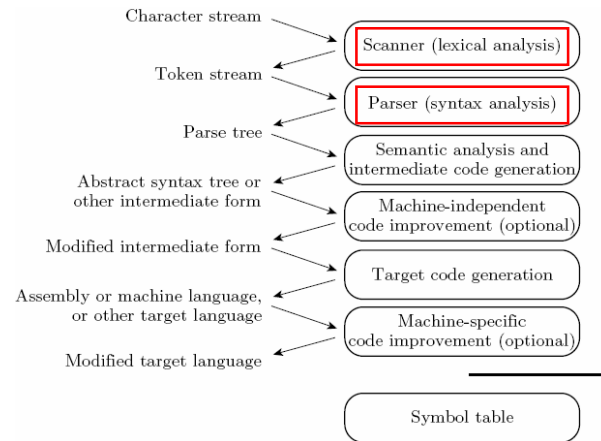


```
27bdfdd0 afbf0014 0c1002a8 00000000 0c1002a8 afa2001c 8fa4001c  
00401825 10820008 0064082a 10200003 00000000 10000002 00832023  
00641823 1483fffa 0064082a 0c1002b2 00000000 8fbf0014 27bd0020  
03e00008 00001025
```

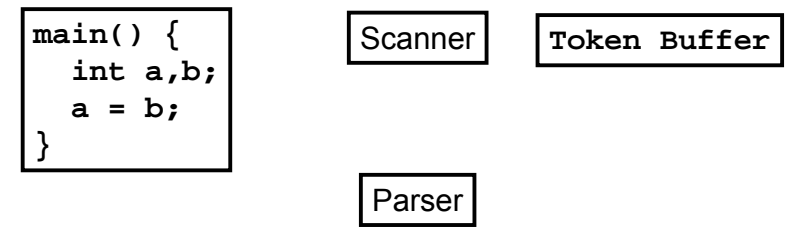
## Prevođenje programa



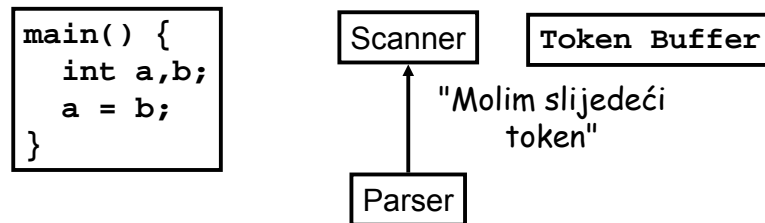
## Faze prevodenja programa



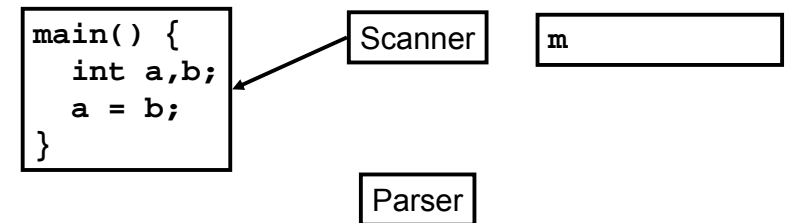
## Scanner/Parser



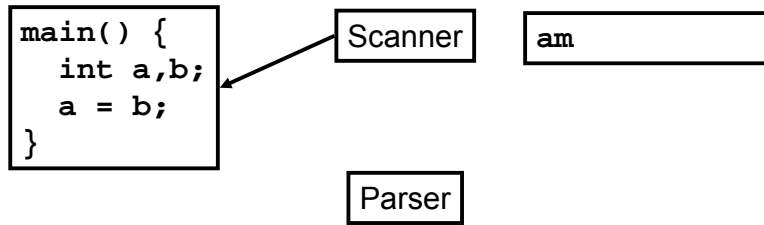
## Scanner/Parser



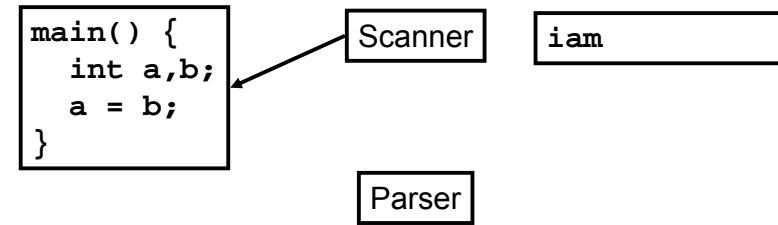
## Scanner/Parser



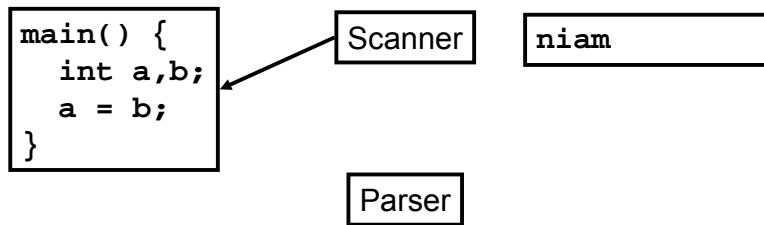
## Scanner/Parser



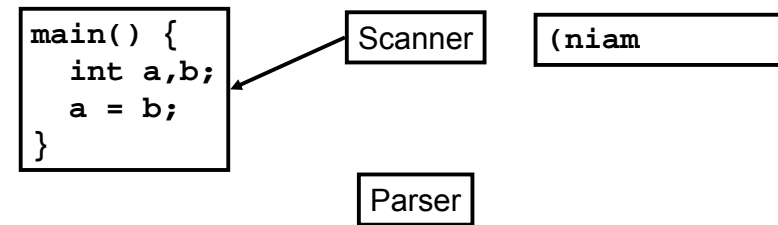
## Scanner/Parser



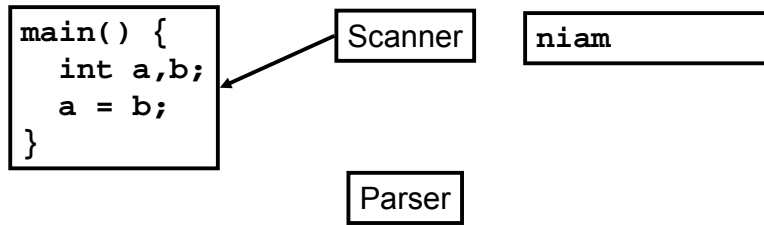
## Scanner/Parser



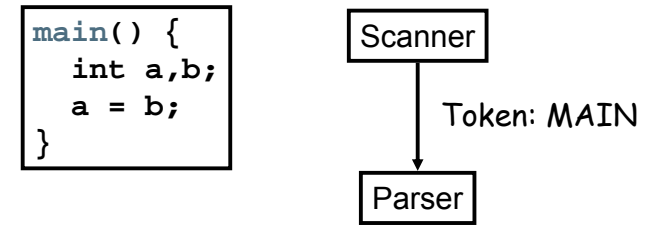
## Scanner/Parser



## Scanner/Parser

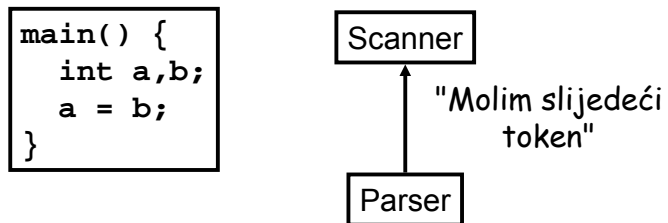


## Scanner/Parser



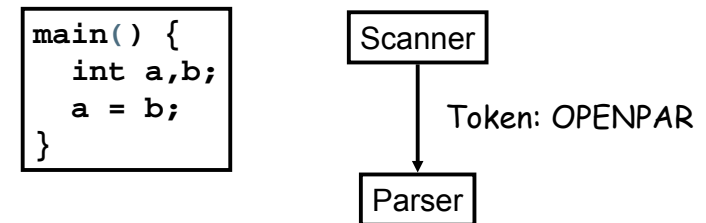
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

## Scanner/Parser



<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

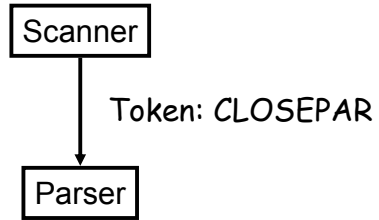
## Scanner/Parser



<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

## Scanner/Parser

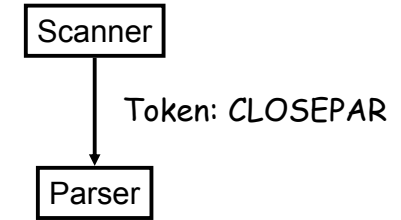
```
main() {  
  int a,b;  
  a = b;  
}
```



<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<PARAMETERS> → NULL

## Scanner/Parser

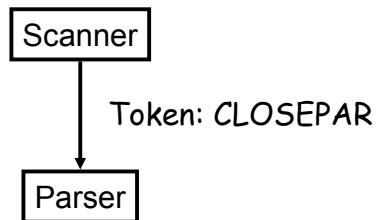
```
main() {  
  int a,b;  
  a = b;  
}
```



<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<PARAMETERS> → NULL

## Scanner/Parser

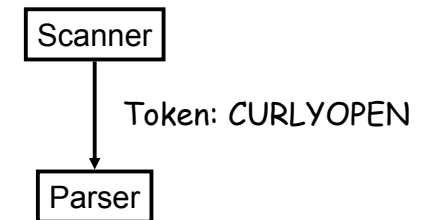
```
main() {  
  int a,b;  
  a = b;  
}
```



<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

## Scanner/Parser

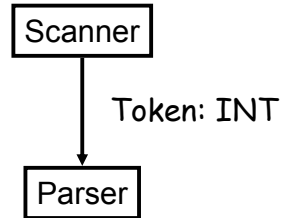
```
main() {  
  int a,b;  
  a = b;  
}
```



<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

## Scanner/Parser

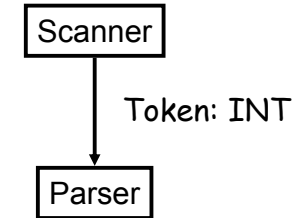
```
main() {  
  int a,b;  
  a = b;  
}
```



`<C-PROG>` → MAIN OPENPAR `<PARAMETERS>` CLOSEPAR `<MAIN-BODY>`  
`<MAIN-BODY>` → CURLYOPEN `<DECL-STMT>` `<ASSIGN-STMT>` CURLYCLOSE  
`<DECL-STMT>` → `<TYPE>`VAR`<VAR-LIST>`;  
`<TYPE>` → INT

## Scanner/Parser

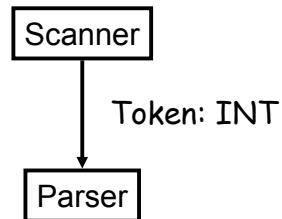
```
main() {  
  int a,b;  
  a = b;  
}
```



`<C-PROG>` → MAIN OPENPAR `<PARAMETERS>` CLOSEPAR `<MAIN-BODY>`  
`<MAIN-BODY>` → CURLYOPEN `<DECL-STMT>` `<ASSIGN-STMT>` CURLYCLOSE  
`<DECL-STMT>` → `<TYPE>`VAR`<VAR-LIST>`;  
`<TYPE>` → INT

## Scanner/Parser

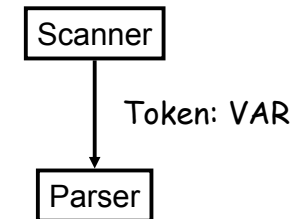
```
main() {  
  int a,b;  
  a = b;  
}
```



`<C-PROG>` → MAIN OPENPAR `<PARAMETERS>` CLOSEPAR `<MAIN-BODY>`  
`<MAIN-BODY>` → CURLYOPEN `<DECL-STMT>` `<ASSIGN-STMT>` CURLYCLOSE  
`<DECL-STMT>` → `<TYPE>`VAR`<VAR-LIST>`;  
`<TYPE>` → INT

## Scanner/Parser

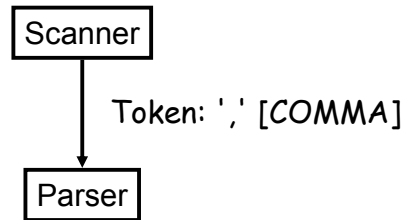
```
main() {  
  int a,b;  
  a = b;  
}
```



`<C-PROG>` → MAIN OPENPAR `<PARAMETERS>` CLOSEPAR `<MAIN-BODY>`  
`<MAIN-BODY>` → CURLYOPEN `<DECL-STMT>` `<ASSIGN-STMT>` CURLYCLOSE  
`<DECL-STMT>` → `<TYPE>`VAR`<VAR-LIST>`;  
`<VARLIST>` → , VAR `<VARLIST>`  
`<VARLIST>` → NULL

## Scanner/Parser

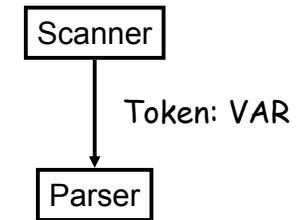
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;  
<VARLIST> → , VAR <VARLIST>  
-----  
<VARLIST> → NULL
```

## Scanner/Parser

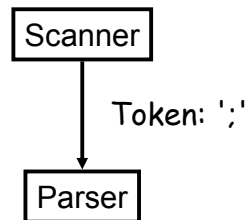
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;  
<VARLIST> → , VAR <VARLIST>  
-----  
<VARLIST> → NULL
```

## Scanner/Parser

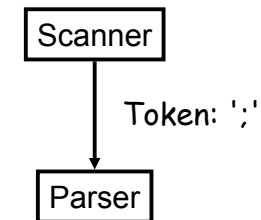
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;  
<VARLIST> → , VAR <VARLIST>  
-----  
<VARLIST> → NULL
```

## Scanner/Parser

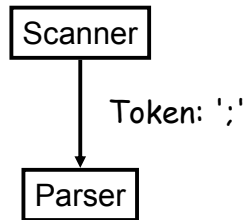
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;  
<VARLIST> → , VAR <VARLIST>  
-----  
<VARLIST> → NULL
```

## Scanner/Parser

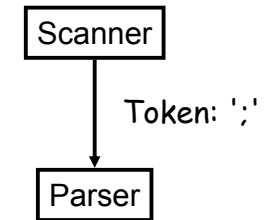
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;  
<VARLIST> → , VAR <VARLIST>  
-----  
<VARLIST> → NULL
```

## Scanner/Parser

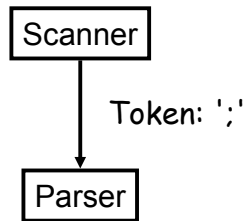
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;  
<VARLIST> → , VAR <VARLIST>  
-----  
<VARLIST> → NULL
```

## Scanner/Parser

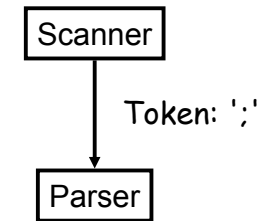
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;  
-----
```

## Scanner/Parser

```
main() {  
  int a,b;  
  a = b;  
}
```

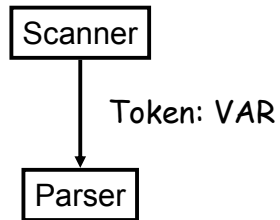


```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;  
-----
```



## Scanner/Parser

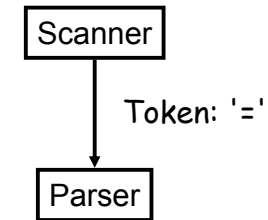
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<ASSIGN-STMT> → VAR = <EXPR>;  
<EXPR> → VAR
```

## Scanner/Parser

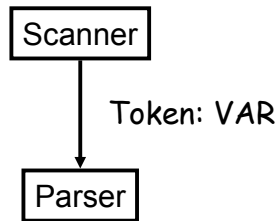
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<ASSIGN-STMT> → VAR = <EXPR>;  
<EXPR> → VAR
```

## Scanner/Parser

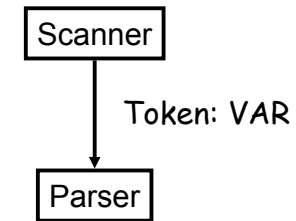
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<ASSIGN-STMT> → VAR = <EXPR>;  
<EXPR> → VAR
```

## Scanner/Parser

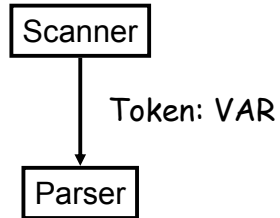
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<ASSIGN-STMT> → VAR = <EXPR>;  
<EXPR> → VAR
```

## Scanner/Parser

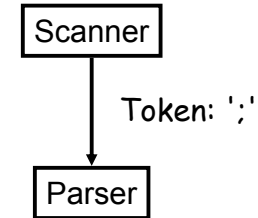
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<ASSIGN-STMT> → VAR = <EXPR>;  
<EXPR> → VAR
```

## Scanner/Parser

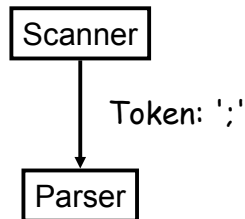
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<ASSIGN-STMT> → VAR = <EXPR>;
```

## Scanner/Parser

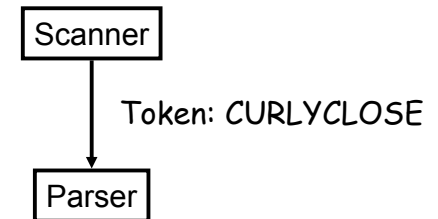
```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE  
<ASSIGN-STMT> → VAR = <EXPR>;
```

## Scanner/Parser

```
main() {  
  int a,b;  
  a = b;  
}
```



```
<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>  
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE
```

## Primjer (leksička i sintaktička analiza)

### ■ Program GCD (Pascal):

```
program gcd(input, output);
var i, j : integer;
begin
  read(i, j);
  while i <> j do
    if i > j then i := i - j
    else j := j - i;
  writeln(i)
end.
```

## Leksička analiza

### ■ Tokeni u programu GCD:

```
program gcd ( input , output ) ;
var i , j : integer ; begin
read ( i , j ) ; while
i <> j do if i > j
then i := i - j else j
:= j - i ; writeln ( i
) end .
```

## KS gramatika i parsiranje

### ■ KS gramatika za Pascal programe:

$program \rightarrow PROGRAM\ id\ ( id\ more\_ids ) ;\ block\ .$

where

$block \rightarrow labels\ constants\ types\ variables\ subroutines\ BEGIN\ stmt\ more\_stmts\ END$

and

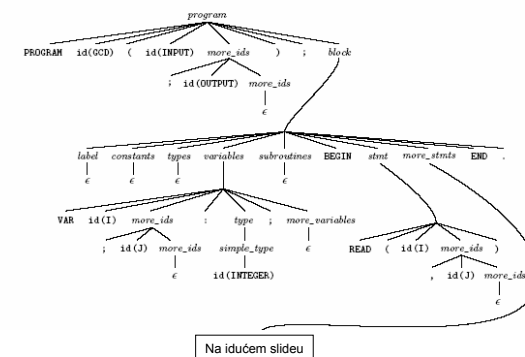
$more\_ids \rightarrow ,\ id\ more\_ids$

or

$more\_ids \rightarrow \epsilon$

## KS gramatika i parsiranje

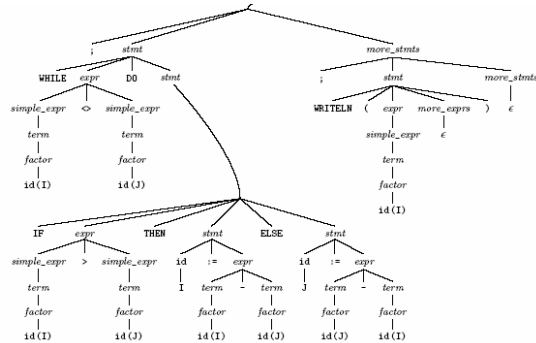
### ■ Stablo izvoda za program GCD:



Na idućem slideu

# KS gramatika i parsiranje

## ■ Stablo izvoda za program GCD:



# KS gramatika i parsiranje

## ■ Apstraktno sintakšno stablo za program GCD:

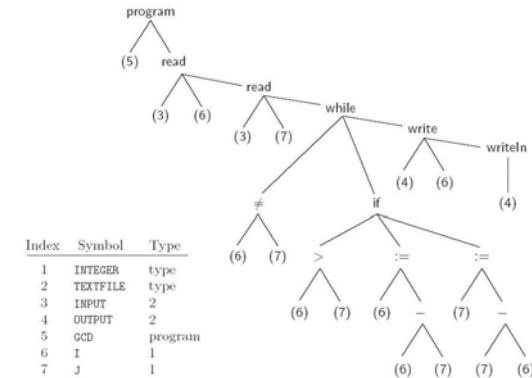


Figure 1.4: Syntax tree and symbol table for the GCD program.

## Leksička analiza

# Regularni izrazi

- Leksička struktura tokena opisuje se regularnim izrazima:
- Primjer (cijeli broj bez predznaka u Pascalu):

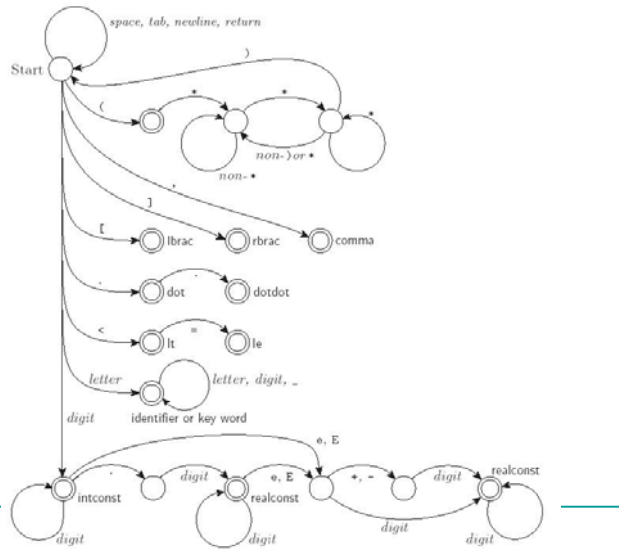
$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$unsigned\_integer \rightarrow digit \ digit^*$

$unsigned\_number \rightarrow unsigned\_integer \ ( ( \ . \ unsigned\_integer ) \mid \epsilon ) \ ( ( ( \ e \mid E ) \ ( + \mid - \mid \epsilon ) \ unsigned\_integer ) \mid \epsilon )$

## Scanner

- Iz danog regularnog izraza moguće je automatski generirati konačni automat



## Problem

- Što u ovakvom slučaju?

```
void f (float* p, float y)
{
    float x;
    x = y/*p;
}
```

(/\* je delimiter za komentare u C-u)

## Lookahead

- Kod nekih jezika odluku o tipu tokena nije moguće donijeti samo na temelju početnog prefiksa tokena
- Još jedan primjer: FORTRAN
  - Varijable ne trebaju biti eksplicitno deklarirane
  - Implicitno tipiziranje u ovisnosti u prvom karakteru
  - Realni brojevi počinju s A-H ili O-Z
  - Cijeli brojevi počinju s I-N
  - (Dizajneri jezika željeli oponašati standardne matematičke konvencije)

## Lookahead

- Tipičan izgled petlje u FORTRAN-u:

```
DO 10 I = 1,10000
```

```
...
```

```
...
```

```
10 CONTINUE
```

- Ekvivalent u C-u:

```
for (i=1; i<=10000; ++i) {
```

```
...
```

```
}
```

## Lookahead

- Sve praznine se ignoriraju, pa je  
`DO 10 I = 1, 100000`
- ekvivalentno  
`DO10I=1,100000`
- što je pak različito od  
`DO 10 I = 1. 100000`  
tj.  
`DO10I=1.100000`

## Rezultat



Sintaktička analiza

## Kontekstno slobodni jezici

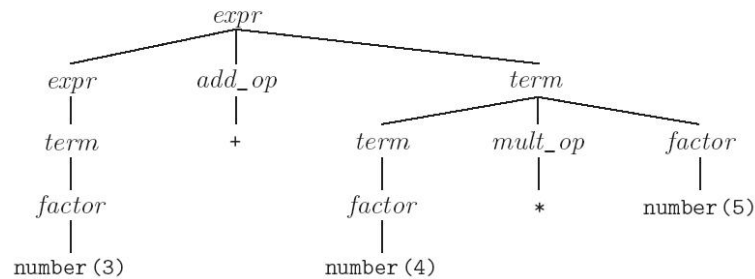
### ■ Primjer: Gramatika aritmetičkih izraza

1.  $expr \rightarrow term \mid expr \text{ add\_op } term$
2.  $term \rightarrow factor \mid term \text{ mult\_op } factor$
3.  $factor \rightarrow id \mid number \mid - factor \mid ( expr )$
4.  $add\_op \rightarrow + \mid -$
5.  $mult\_op \rightarrow * \mid /$

## Kontekstno slobodni jezici

### ■ Primjer: Stablo izvoda za izraz

3 + 4 \* 5



## Još jedan primjer – XML i DTD

- DTD je KS gramatika za opis strukture neke klase XML dokumenata

### ■ Primjer:

```
<?xml version="1.0"?>
<!DOCTYPE oldjoke [
  <!ELEMENT oldjoke (burns+, allen, applause?)>
  <!ELEMENT burns (#PCDATA | quote)*>
  <!ELEMENT allen (#PCDATA | quote)*>
  <!ELEMENT quote (#PCDATA)*>
  <!ELEMENT applause EMPTY>
]>
<oldjoke>
  <burns>Say <quote>goodnight</quote>,
  Gracie.</burns>
  <allen><quote>Goodnight, Gracie.</quote></allen>
  <applause />
</oldjoke>
```

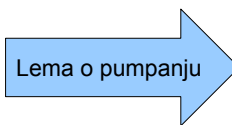
## Kontekstno slobodni jezici

- $L = \{w \in \text{ASCII}^* \mid w \text{ je sintaktički korektan C program}\}$
- Konstruiramo C program u ovisnosti o  $n$  iz leme o pumpanju:

$n=4$

```
void main(void)
{
  int v1;
  int v2;
  int v3;
  int v4;

  v1 = 0;
  v2 = 0;
  v3 = 0;
  v4 = 0;
}
```



## Kontekstno slobodni jezici

- Klasa KS jezika nije dovoljna za opis svih pravila programskih jezika:
  - "svi identifikatori moraju biti deklarirani prije upotrebe":  
 $L = \{w \# w \mid w \in (a+b)^*\}$
  - "broj formalnih parametara u deklaraciji funkcije podudara se s brojem argumenata kod funkcijskog poziva":  
 $L = \{a^m b^n c^m d^n \mid m, n \geq 1\}$

## Kontekstno slobodni jezici

- "Klasa KS jezika od velike je važnosti u računarstvu"
- Dva (kontradiktorna) problema:
  1. KS jezici nisu dovoljno snažni za opis svih pravila programskih jezika (npr. tipovnog sustava)
  2. Opis gotovo svih sintaktičkih pravila programskih jezika u pravilu ne zahtjeva punu ekspresivnost KS jezika
- Želimo parser koji parsira ulazni program duljine  $n$  u vremenu  $O(n)$
- U praksi:
  - restriktivnije gramatike – klase  $LL(k)$  i  $LR(k)$
  - ekspresivnije gramatike – atribuirane, dvorazinske,  $W$ -, DCG (Prolog)

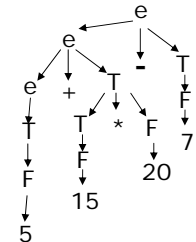
## Top-down parsiranje

- Počevši od startnog neterminalnog simbola trebamo naći najljeviji izvod

**Gramatika:**

$$E ::= E + T \mid E - T \mid T$$

$$T ::= T * F \mid T / F \mid F$$

$$F ::= n$$


**Najljeviji izvod za 5+15\*20-7:**

$$E \rightarrow E-T \rightarrow E+T-T \rightarrow T+T-T$$

$$\rightarrow F+T-T \rightarrow 5+T-T \rightarrow 5+T*F-T \rightarrow 5+F*F-T \rightarrow 5+15*F-T$$

$$\rightarrow 5+15*20-T \rightarrow 5+15*20-F \rightarrow 5+15*20-7$$

## Prediktivno parsiranje

- $LL(k)$  parser - s lijeva na desno, najljeviji izvod,  $k$ -simbola lookahead
  - Odluka o odabiru produkcije gramatike donosi se na temelju provjere narednih  $k$  tokena
  - $LL(k)$  gramatika – gramatika koju je moguće parsirati  $LL(k)$  parserom
- $LL(1)$  parser – produkcija gramatike se bira samo na temelju idućeg tokena

**Gramatika:**

$$E ::= E + T \mid E - T \mid T$$

$$T ::= T * F \mid T / F \mid F$$

$$F ::= n$$

Svaka produkcija za  $E$  započinje s  $E \rightarrow$  nije  $LL(K)$

**Ekvivalentna  $LL(1)$  gramatika :**

**Gramatika:**

$$E ::= TE'$$

$$E' ::= + TE' \mid - TE' \mid \epsilon$$

$$T ::= FT'$$

$$T' ::= *FT' \mid / FT' \mid \epsilon$$

$$F ::= n$$

## $LL(1)$ parsiranje - Primjer

**Gramatika**

$$S \rightarrow ( S ) S \mid \epsilon$$

**Ulaz**

$$( ) \$$$

- $\$$  - oznaka dna stoga i kraja ulazne riječi

$\$$



## LL(1) parsiranje - Primjer

Gramatika  
 $s \rightarrow ( s ) s \mid \epsilon$

Ulaz  
( )\$

- Stavimo početni simbol S na stog



## LL(1) parsiranje - Primjer

Gramatika  
 $s \rightarrow ( s ) s \mid \epsilon$

Ulaz  
( )\$

- Expand:  $s \rightarrow ( s ) s$



## LL(1) parsiranje - Primjer

Gramatika  
 $s \rightarrow ( s ) s \mid \epsilon$

Ulaz  
( )\$

- Match



## LL(1) parsiranje - Primjer

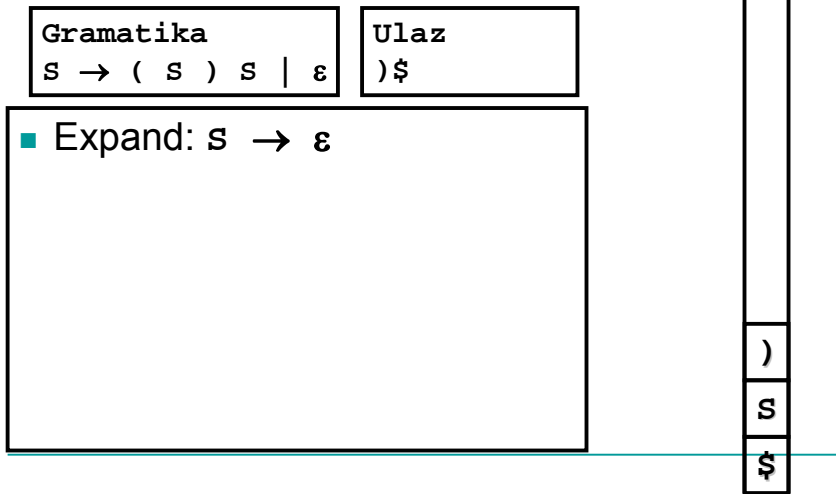
Gramatika  
 $s \rightarrow ( s ) s \mid \epsilon$

Ulaz  
)\$

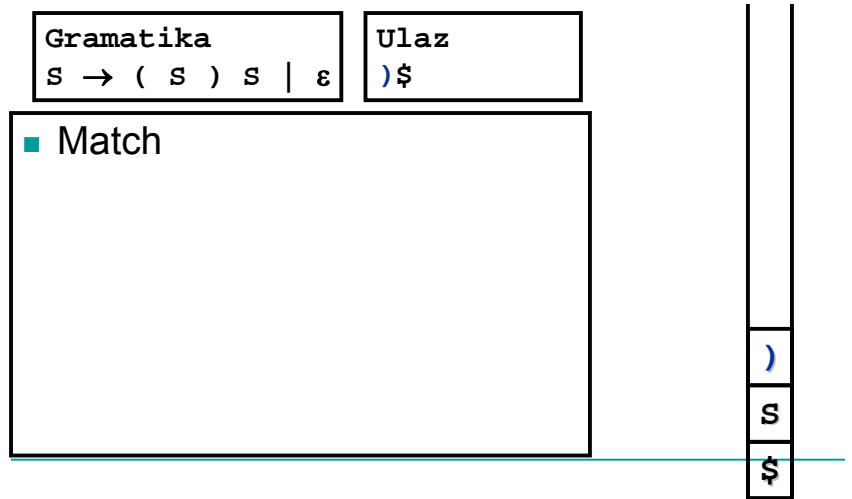
- Expand:  $s \rightarrow \epsilon$



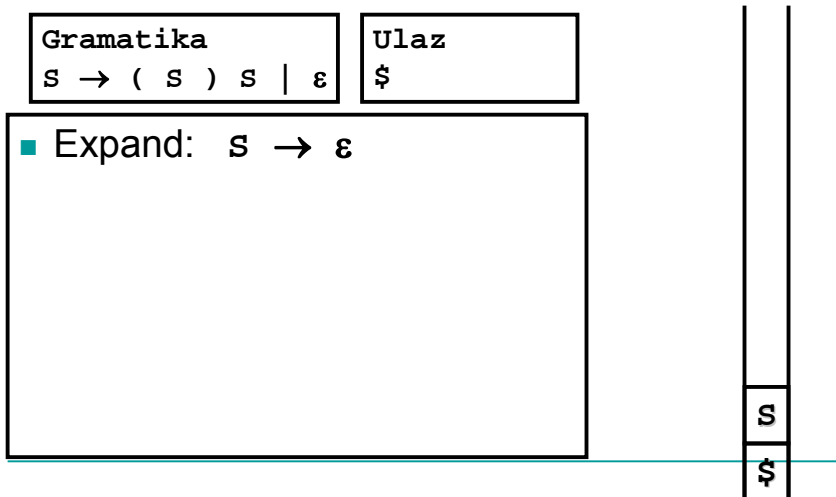
## LL(1) parsiranje - Primjer



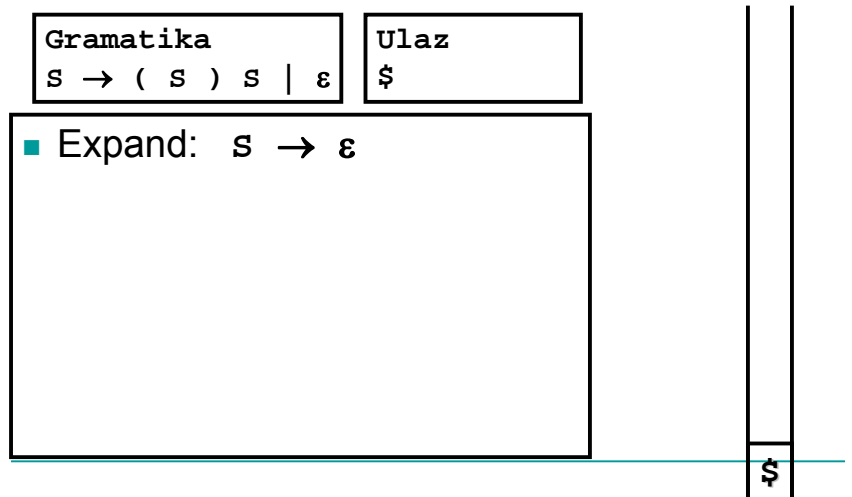
## LL(1) parsiranje - Primjer



## LL(1) parsiranje - Primjer



## LL(1) parsiranje - Primjer



## LL(1) parsiranje - Primjer

**Gramatika**  
 $S \rightarrow ( S ) S \mid \epsilon$

**Ulaz**  
 $\$$

- Parsiranje uspješno završilo!

$\$$

## Bottom-up parsiranje

- Počevši od ulaznog stringa, trebamo reducirati ulazni string na početni neterminalni simbol (ekvivalentno obrnutom najdesnijem izvodu)

**Gramatika:**

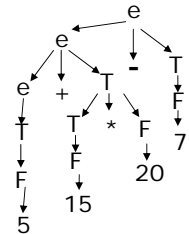
$E ::= E + T \mid E - T \mid T$   
 $T ::= T * F \mid T / F \mid F$   
 $F ::= n$

**Najdesniji izvod za  $5+15*20-7$ :**

$E \rightarrow E - T \rightarrow E - F \rightarrow E - 7 \rightarrow E + T - 7 \rightarrow E + T * F - 7$   
 $\rightarrow E + T * 20 - 7 \rightarrow E + F * 20 - 7 \rightarrow E + 15 * 20 - 7$   
 $\rightarrow T + 15 * 20 - 7 \rightarrow F + 15 * 20 - 7 \rightarrow 5 + 15 * 20 - 7$

**Bottom-up parsiranje:**

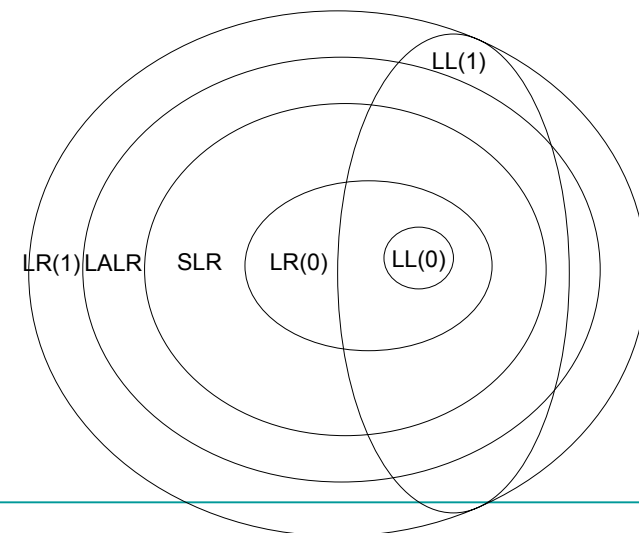
$5 + 15 * 20 - 7 \rightarrow F + 15 * 20 - 7 \rightarrow T + 15 * 20 - 7$   
 $\rightarrow E + 15 * 20 - 7 \rightarrow E + F * 20 - 7 \rightarrow E + T * 20 - 7$   
 $\rightarrow E + T * F - 7 \rightarrow E + T - 7 \rightarrow E - 7 \rightarrow E - F \rightarrow E - T \rightarrow E$



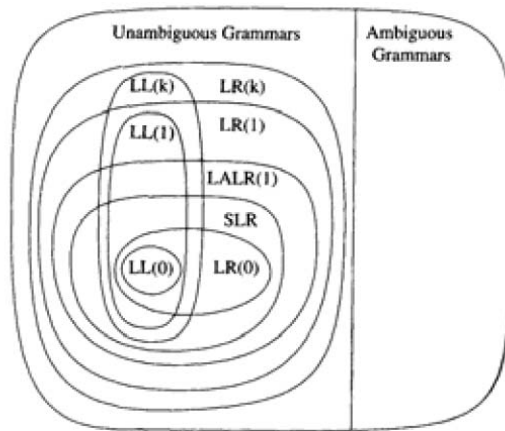
## LR(k) parsiranje

- LR(k) parser - s lijeva na desno, najdesniji izvod, k simbola lookahead
  - Odluke donose na temelju provjere narednih k tokena
  - Koriste konačne automate za konfiguraciju akcija (shift ili reduce)
    - Stanja automata pamte simbole koji su već na stogu
    - Svaki par (stanje, token) jedinstveno određuje iduću akciju
- Zašto koristiti LR parsere?
  - Prepoznaju širu klasu KS jezika od prediktivnih LL(k) parsera
  - Efikasna implementacija, nema backtrackinga
- LR(k) vs LL(k):
  - LL parseri – pogodni za "ručnu" implementaciju (jednostavan programski kod)
  - LR parseri – bitno teža "ručna" implementacija; obično se koriste automatski generatori (npr. Yacc/Bison) -> kompliciran i "ružan" programski kod

## Odnos među klasama gramatika



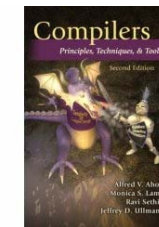
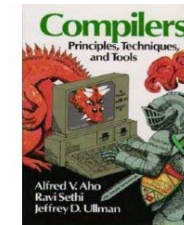
## Odnos među klasama gramatika



## The Dragon Book

### ■ **Compilers: Principles, Techniques and Tools**

- Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman (1st Edition)
- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman (2nd Edition)

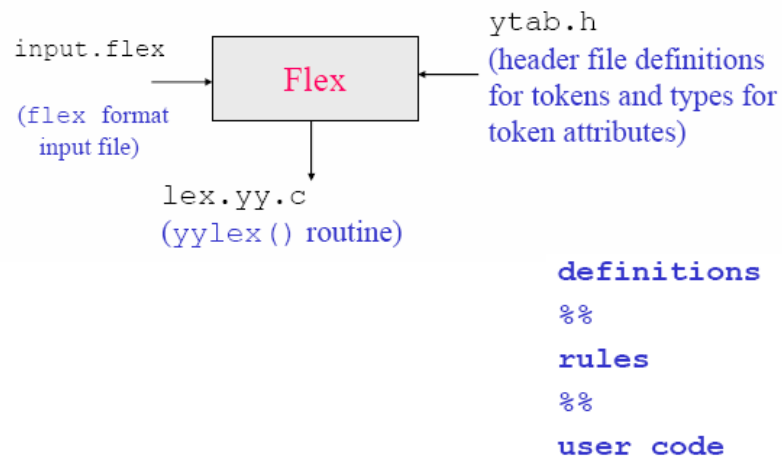


Alati za generiranje lexera / parsera

## (F)lex

- (F)lex = (Fast) Lexical Analyzer Generator
- Neke implementacije:
  - Flex for Windows
    - <http://gnuwin32.sourceforge.net/packages/flex.htm>
  - JFlex
    - <http://www.jflex.de/>

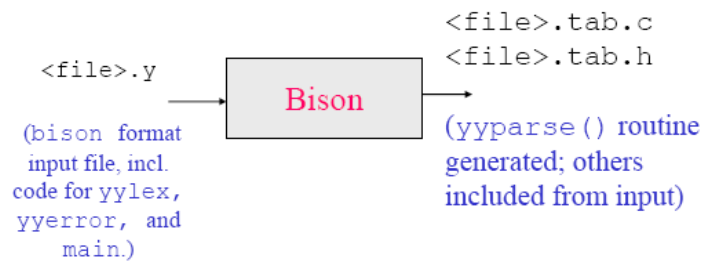
## (F)lex



## Yacc/Bison

- Yacc = Yet Another Compiler-Compiler
  - generator LALR(1) parsera
- Bison – generator LALR(1) / GLR parsera (kompatibilan s Yacc-ovom sintaksom ulaza)
- Neke implementacije:
  - Bison for Windows
    - <http://gnuwin32.sourceforge.net/packages/bison.htm>
  - BYACC/J
    - <http://byaccj.sourceforge.net/>

## Yacc/Bison



## Yacc/Bison

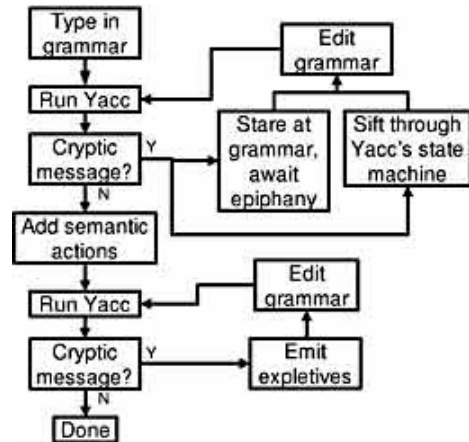
The input file consists of three sections, separated by a line with just `%%` on it:

```
%%{
C declarations (types, variables, functions,
preprocessor commands)

%}
Bison declarations (grammar symbols,
operator precedence decl.,
attribute data type)

%%
Grammar rules
%%
Additional C code (incl. scanner yylex)
```

## Ciklus razvoja parsera s Yacc/Bison-om



<http://www.acm.org/crossroads/xrds7-5/bison.html>

## ANTLR

### ■ ANTLR = ANOther Tool for Language Recognition

- generator lexera i LL(k) parsera
  - zapravo LL(\*) s linearnom aproksimacijom lookaheada i sintaktičkim/semantičkim predikatima
  - zajednička sintaksa za specifikaciju lexera i parsera

### ■ Linkovi:

- ANTLR
  - <http://www.antlr.org/>
- ANTLR plugin za Eclipse
  - <http://antreclipse.sourceforge.net/>