

# Programski prevodioci

## 02 Leksička analiza

Fakultet tehničkih nauka, Novi Sad  
22-23/Z  
Dunja Vrbaški

## Leksička analiza - Skener



Prva faza prevođenja: podeliti ulazni string na smislene delove

Nazivi: skener ili lekser

Skener – skeniramo tekst na ulazu, slovo po slovo, leksemu po leksemu

Lekser – lekseme

```
if (x == 5)
{
    y = 3; //komentar
}
```

if	(	x	==	5	)	{	y	=	3	;	}
----	---	---	----	---	---	---	---	---	---	---	---

lekseme

*Zašto su neke lekseme označene na slici?*

```
if (x == 5)
{
    y = 3; //komentar
}
```



Zapravo, prepoznavamo i beline

U nekim jezicima – utiču na značenje  
Mogu biti od koristi za druge aktivnosti

## Šta bi bila greška u leksičkoj analizi?

```
if (8k == 5)
{
    y = 3 * 2;
}
```

Ako je naš jezik takav da:

- identifikator ne može da počne brojem niti broj može da se završi slovom k tj ne postoji ništa što bi moglo da se prepozna da je deo jezika
- ne prepoznaje \* kao deo jezika

Šta znači “deo jezika”?

**Tokeni** - prepoznate lekseme se klasifikuju po tipu uz eventualno dodavanje korisnih informacija

Na primer:

Leksema	Vrsta tokena	Vrednost tokena
if	Ključna reč	if
while	Ključna reč	while
=	Dodela	
==	Relacioni operator	==
<	Relacioni operator	<
+	Aritmetički operator	+
;	Tačka zarez	
x	Identifikator	x
y	Identifikator	y
5	Literal	5

*Vrsta tokena se u literaturi često zove **klasa tokena***

Tokeni je par koga čine vrsta tokena i vrednost tokena.

- Vrsta tokena je apstraktna reprezentacija klase jezičkih jedinica.
- Vrednost tokena nije obavezna i najčešće predstavlja konkretnu, prepoznatu, vrednost za konkretan ulazni string.

Lekseme su instance tokena koje su prepoznate prilikom obrade ulaznog stringa.



Na primer:

Leksema	Vrsta tokena	Vrednost tokena
if	Ključna reč	if
while	Ključna reč	while
=	Dodela	
==	Relacioni operator	==
<	Relacioni operator	<
+	Aritmetički operator	+
;	Tačka zarez	
x	Identifikator	x
y	Identifikator	y
5	Literal	5

Leksema	Vrsta tokena	Vrednost tokena
if	Ključna reč	if
while	Ključna reč	while
=	Operator	=
==	Operator	==
<	Operator	<
+	Operator	+
;	Tačka zarez	
x	Identifikator	x
y	Identifikator	y
5	Literal	5

Implementacija, na primer:

Leksema	Vrsta tokena (neformalno)	Vrsta tokena (npr enum)	Vrednost tokena
if	Ključna reč	KW	If
while	Ključna reč	KW	while
=	Dodela	ASG	
==	Relacioni operator	ROP	==
<	Relacioni operator	ROP	<
+	Aritmetički operator	AOP	+
;	Tačka zarez	SC	
x	Identifikator	ID	x
y	Identifikator	ID	y
5	Literal	LIT	5

*Kog tipa su vrednosti?*

Token može evidentirati i poziciju u izvornom kodu radi informativnijih poruka o greškama:

- pamti se linija
- pamti se linija i kolona
- pamti se pomeraj od početka teksta

Mogu se pamtit i druge informacije (atributi).

Zašto, uopšte, pravimo (i pamtimo) tokene?

Za leksičku analizu nam, u osnovnom slučaju, tokeni nisu neophodni.

Dovoljno je da delove stringa prepoznamo kao elemente jezika i javimo da li ima grešaka ili ne tj da li je program napisan korišćenjem ispravnih reči.

# Implementacija

Kako implementirati skener?

Kako implementirati skener?

Intuitivno:

- čitamo string sa ulaza, karakter po karakter
- prepoznavamo lekseme
- (kreiramo tokene)
- informišemo o rezultatu (ok/greška)

Kako implementirati skener?

ručno – (while, if/switch, lookahead); proizvoljno

koristeći alat – (Flex, Lex, Antlr...); za to nam treba neka formalizacija jezika

Bez obzira na pristup, pojavljuju se određeni problemi. Na primer:

- potrebno je da  $\leq$  bude ispravno prepoznato (ne kao 2 operatora već kao jedan)
- ako postoje rezervisane reči potrebno je da one ne budu prepoznate kao opšti identifikatori
- pojava greške – da li odmah stati sa obradom i prijaviti grešku ili dati informaciju i nastaviti, pokušati dalje
- beline su važne ili ne
- signal za kraj (može biti važno za naredne faze)
- ...



Takođe, postavlja se pitanje šta je ulaz za sledeću fazu (sintaksna analiza)?

- obraditi ceo ulazni string i generisati niz tokena koji se u celosti predaje narednoj fazi
- prosleđivati, na zahtev, token po token narednoj fazi

Mogu se koristiti **regularni izrazi** za opis leksike (leksičke gramatike) programskog jezika.  
(formalizacija)

### Zašto bismo to radili?

- Jednostavna formalna reprezentacija leksike jezika
- Flex - na osnovu leksike definisane regularnim izrazima automatski kreira skener.
- Ručno - možemo slično primeniti i iskoristiti za prepoznavanje leksema

U nastavku:

- regularni izrazi
- kako se prave skeneri na osnovu regularnih izraza
- kako se koristi Flex

Regularni izrazi se koriste za opis regularnih jezika.

Regularni jezici su vrsta **formalnih jezika**.

Kontekstno slobodni jezici su, takođe, vrsta formalnih jezika.

Leksika velikog broja programskih jezika predstavlja regularan jezik.

Velik deo sintakse velikog broja programskih jezika predstavlja kontekstno slobodni jezik.

Obratiti pažnju – Kad pričamo o gramatikama PL obično mislimo na gramatiku sintakse. Međutim, imamo bar dve gramatike: leksike i sintakse.

## Prirodni jezik

- skup slova (**azbuka**, alfabet)
- skup **reči** građenih nad azbukom
- skup **rečenica** građenih nad skupom reči
- Reči prate **pravila** leksike
- Rečenice prate **pravila** sintakse jezika
- Rečenice prate **pravila** semantike jezika

*Danas pada kiša.*

*Danas će padati kiša.*

*Danas je padala kiša.*

~~*Je danas oni padala.*~~

~~*Juče će padati kiša.?*~~

Rečnik – skup reči koje se nalaze u jeziku.

Skup pravila za prirodni jezik praktično predstavlja pravila konverzacije.

Pravila nisu jednostavna i teže se formalno opisuju.

# Teorija automata i formalnih jezika

- formalni jezici
- gramatike
- automati

Formalni jezici se mogu definisati gramatikama koje predstavljaju skup formalno definisanih pravila pomoću kojih se formiraju reči tog jezika.

Na formalan način opisujemo jezike, definišemo gramatike, istražujemo njihove osobine.

Osnova za unapređenje praktičnih rešenja.

# Formalni jezik

## Osnovni pojmovi

Azbuka  $\Sigma$  - skup simbola

Reč (string)  $\omega$  - konačan niz simbola iz azbuke

Prazna reč  $\lambda$  ( $\epsilon$ )

Skup svih reči uključujući i praznu  $\Sigma^*$

Jezik  $L$  - podskup  $\Sigma^*$

*Obratiti pažnju – azbuka ne mora biti sačinjena od “slova” kao u prirodnim jezicima  
karakter → reči  
reči → rečenice*

**Definicija:** Neka je  $\Sigma$  neka azbuka. Svaki podskup  $L$  skupa svih reči  $\Sigma^*$  nad  $\Sigma$  zovemo jezik nad  $\Sigma$ .

Primetiti:

- Ne spominju se pravila, gramatike
- Jezik je definisan kao rečnik, a ne preko nekih svojih pravila
- Nad jednom azbukom možemo imati više jezika

## Primeri

$$\Sigma = \{a, b\}$$

$$L_1 = \{aaa, ab, baba\}$$

$$L_2 = \{a, aa, aaa, aaaa, \dots\}$$

$$L_3 = \{a, b, aabbb, aaab, aabb, \dots\}$$

$$L_4 = \{\varepsilon, a, b, aabbb, aaab, aabb, \dots\}$$

$$L_5 = \{\varepsilon, abba, bbba, baaab, ab, \dots\}$$



Konačan skup reči – mogle bi se sve reči zapisati

Beskonačan skup reči – da li možda postoji neko pravilo? Da li se nekako formalno mogu opisati ti jezici?

Koja su pravila?

$$\Sigma = \{a, b\}$$

$$L_1 = \{aaa, ab, baba\}$$

$$L_2 = \{a, aa, aaa, aaaa, \dots\}$$

$$L_3 = \{a, b, aabbb, aaab, aabb, \dots\}$$

$$L_4 = \{\varepsilon, a, b, aabbb, aaab, aabb, \dots\}$$

$$L_5 = \{\varepsilon, abba, bbba, baaab, ab, \dots\}$$

# Kako možemo definisati pravila?

## Gramatike

Definisanje pravila po kom se grade reči/stringovi  
(*mehanizam za generisanje*)

$$S_0 \rightarrow aS_1$$

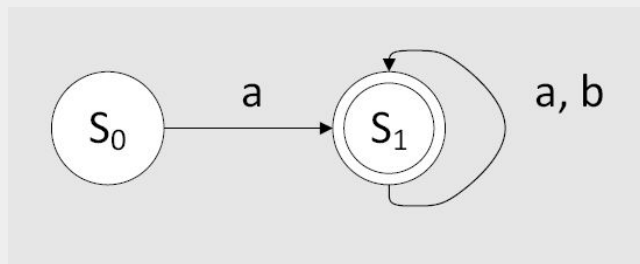
$$S_1 \rightarrow aS_1$$

$$S_1 \rightarrow bS_1$$

$$S_1 \rightarrow \varepsilon$$

## Automati

Prihvataju ili ne prihvataju određenu reč kao reč  
nekog jezika  
(*mehanizam za prepoznavanje*)



*Zapravo su apstraktne mašine, matematičke konstrukcije, modeli računanja, ali imaju zgodnu vizuelnu reprezentaciju*

# Gramatike

**Definicija:** Gramatika je uređena četvorka  $G = (\Sigma, N, P, S)$  gde je:

- $\Sigma$  skup simbola (terminala)
- $N$  skup pojmova (neterminala)
- $P$  skup pravila izvođenja
- $S$  početni pojam
  
- $\Sigma, N, P$  su neprazni, konačni, skupovi
- $\Sigma \cap N = \emptyset$
- $P$  je skup pravila u obliku  $(\Sigma \cup N)^* \mathbf{N} (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$

**Definicija:** Gramatika je uređena četvorka  $G = (\Sigma, N, P, S)$  gde je:

- $\Sigma$  skup simbola (terminala)
- $N$  skup pojmova (neterminala)
- $P$  skup pravila izvođenja
- $S$  početni pojam
  
- $\Sigma, N, P$  su neprazni, konačni, skupovi
- $\Sigma \cap N = \emptyset$
- $P$  je skup pravila u obliku  $(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$

Primer 1

$$\begin{aligned} S_0 &\rightarrow aS_1 \\ S_1 &\rightarrow aS_1 \\ S_1 &\rightarrow bS_1 \\ S_1 &\rightarrow \varepsilon \end{aligned}$$

Primer 2

$$\begin{aligned} S &\rightarrow aSa \\ S &\rightarrow bSb \\ S &\rightarrow a \\ S &\rightarrow b \\ S &\rightarrow \varepsilon \end{aligned}$$

# Konačni automati

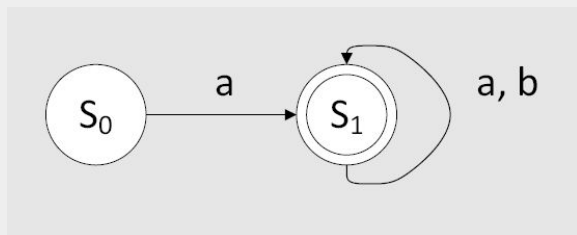
**Definicija:** Konačni deterministički automat je uređena četvorka  $A = (S, \Sigma, \sigma, s_0, F)$  gde je:

- $S$  - skup stanja
- $\Sigma$  - alfabet
- $\sigma$  - funkcija prelaza,  $\sigma: S \times \Sigma \rightarrow S$
- $s_0$  - inicijalno stanje
- $F$  - skup ciljnih stanja

**Definicija:** Konačni deterministički automat je uređena četvorka  $A = (S, \Sigma, \sigma, s_0, F)$  gde je:

- $S$  - skup stanja
- $\Sigma$  - alfabet
- $\sigma$  - funkcija prelaza,  $\sigma: S \times \Sigma \rightarrow S$
- $s_0$  - inicijalno stanje
- $F$  - skup ciljnih stanja

## Primer



# Regularni jezici

Nad azbukom  $\Sigma$ , regularni jezici su:

- Prazan skup je regularan jezik
- Jezik  $\{\epsilon\}$  je regularan jezik
- Za svaki simbol/slovo  $a \in \Sigma$  jezik  $\{a\}$  je regularan
- Ako su  $L_1$  i  $L_2$  regularni jezici onda su regularni jezici i  $L_1 \cup L_2$ ,  $L_1 \cdot L_2$  i  $L_1^*$

$L_1 \cdot L_2 = L_1 L_2$  konkatencija

$L^*$  je unija jezika  $L^0, L^1, L^2, \dots$  gde je  $L^0 = \{\epsilon\}, L^1 = L, L^2 = LL, \dots$



## Regularni izrazi

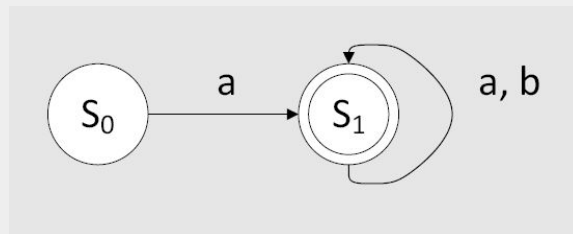
Nad azbukom  $\Sigma$ , regularni izrazi su:

- svi elementi azbuke, prazna reč  $\varepsilon$  , prazan skup
- ako su  $\alpha$  i  $\beta$  regularni izrazi onda su to i  $\alpha \cdot \beta$ ,  $\alpha \mid \beta$ ,  $\alpha^*$

Teorija:

regularni jezik  $\Leftrightarrow$  regularni izrazi  $\Leftrightarrow$  regularna gramatika  $\Leftrightarrow$  konačni deterministički automat.

$a(a|b)^*$

$$\begin{array}{l} S_0 \rightarrow aS_1 \\ S_1 \rightarrow aS_1 \\ S_1 \rightarrow bS_1 \\ S_1 \rightarrow \varepsilon \end{array}$$


*Primetiti: Nismo precizirali šta je regularna gramatika.*

# Hijerarhija Čomskog

(Noam Chomsky)

Gramatike:

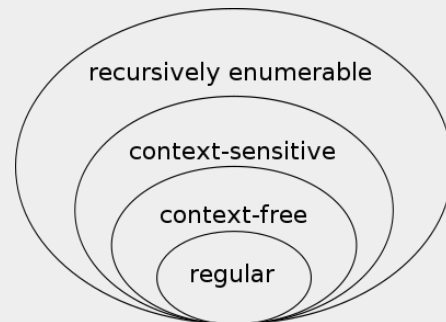
Tip 3 – regularne gramatike (**regularni jezici**)

Tip 2 – kontekstno slobodne gramatike (kontekstno slobodni jezici / context free)

Tip 1 – kontekstno osetljive gramatike (kontekstno osetljivi (zavisni) jezici / context sensitive)

Tip 0 – gramatike bez restrikcija (jezici sa fraznom strukturom / rekurzivno nabrojivi)

$T_3 \subseteq T_2 \subseteq T_1 \subseteq T_0$  (Tjuringove mašine)



Teorija automata i formalnih jezika se bave pitanjima koja se pojavljuju u okviru izučavanja formalnih jezika. Na primer:

- koje klase gramatika i automata postoje i kakve jezike generišu?
- šta važi za takve jezike?
- da li za automat postoji gramatika i obrnuto? da li znamo oblik?
- kako od jedne reprezentacije doći do druge?
- kako je povezano izračunavanje, pa i čitavo računarstvo, sa ovim?
- ...
- Tjuringove mašine, izračunljivost, odlučivost
- mašine
- P vs NP problem

Gramatike za opis leksike programskog jezika su znatno jednostavnije nego gramatike za opis sintakse programskog jezika.

Leksika velikog broja programskih jezika predstavlja regularan jezik.

Velik deo sintakse velikog broja programskih jezika predstavlja kontekstno slobodni jezik.



Formalno, regularni izrazi imaju samo: |, konkatenaciju i \* (definicija regularnog jezika)

Postoje proširenja u bibliotekama/alatima

Postoji POSIX (IEEE) standard

- BRE i ERE (Basic i Extended)

Alat - ne računati da je uvek standard ispoštovan niti da radimo sa regularnim jezicima

Određena proširenja su dodata radi lakšeg zapisivanja, ali imaju svoju ekvivalentnu formu koja koristi početni skup operatora ( $R^+ = RR^*$ )

*U praksi - moramo precizirati:*

- *koja je azbuka*
- *šta tačno želimo da predstavimo (regex za email?)*
- *da li i koje biblioteke/alate koristimo*

*Praksa != formalne definicija RE*

Oznaka	Primer	Formiranje	Skup reči
		Prazan string	
	a	Svaki simbol (karakter) je RE	
	ab	Vrši se konkatencija dva RE	"ab"
*	ab*	RE na koji se odnosi se ponavlja 0 ili više puta	"a", "ab", "abb", ....
+	ab+	RE na koji se odnosi se ponavlja 1 ili više puta	"ab", "abb", ...
?	ab?	RE na koji se odnosi se ponavlja 0 ili 1 put	"a", "ab"

*RE – Regular Expression, regularni izraz*



Oznaka	Primer	Formiranje	Skup reči
	a b	Alternativa	“a”, “b”
( )	a(b c)	Grupisanje	“ab”, “ac”
[ ]	[abc]	Alternative	“a”, “b”, “c”
[ - ]	[a-c]	Opseg	“a”, “b”, “c”
[ ^ ]	[^abc]	Alternative koje ne odgovaraju navedenom	“d”, “e”,... (ostalo iz azbuke)
{m, n}	a{1, 3}	Broj ponavljanja	“a”, “aa”, “aaa”
.	a.	Bilo koji znak (osim nove linije) <i>[a.b] – tu se često interpretira baš kao znak “.”</i>	“aa”, “ab”, “ac”...
\	a\.	Escape, specijalne karaktere tretiramo kao obične	“a.”

## flex

Flex je alat za generisanje skenera.

Na osnovu regularnih izraza koji predstavljaju gramatiku leksike nekog jezika generiše funkciju koja predstavlja skener za taj jezik.

Skener može da uradi dva zadatka:

- utvrđuje ispravnost ulaznog stringa odnosno da li prepoznate lekseme pripadaju definisanom jeziku
- kreira tokene na osnovu prepoznatih leksema

Međutim, tokeni su potrebni tek drugoj fazi (sintaksa) → nećemo ih sada razmatrati

Ideja:

**MI:** Leksika se predstavi regularnim izrazima

**TEORIJA:** Za svaki regularni izraz postoji odgovarajući konačni automat i zna se precizan postupak kako se formira.

**ALAT:** Generiše funkciju koja simulira rad odgovarajućeg automata koji će biti zadužen za prepoznavanje

→

napišemo regularne izraze

dobijemo gotovu mašinu / alat / implementaciju

## TEORIJA napomene

Regularni izrazi, nederministički i deterministički konačni automati – svi predstavljaju istu klasu jezika

Ali “regularni izrazi” u kontekstu formalne računarske nauke!

Neke specifikacije, alati, dozvoljavaju kreiranje šireg skupa.

## POSTUPAK

1. Regularni izrazi koji predstavljaju leksičku gramatiku se navode u specifikaciji (`jezik.1`)
2. Specifikacija se prosleđuje flex programu. [`$ flex jezik.1`]
3. Rezultat je izlazni fajl (`lex.yy.c`) u kom je definisana funkcija `int yylex(void)`
4. Nakon kompajliranja dobija se izvršni fajl [`$ gcc -o skener lex.yy.c`]
5. Pokreće se
  - interaktivno – ulaz je tastatura (`$ ./skener`)
  - vrši se analiza ulaznog fajla (`$ ./skener < test.txt`)

## Primer

$$L = \{0, 1, 42, 402, 000150...\}$$

Jezik L - skup svih reči nad azbukom koju čine cifre, bez prazne reči

Regularni izrazi:

$[0-9]^+$

# jezik.1

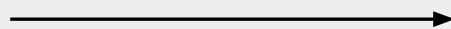
```
%option noyywrap
```

```
%%
```

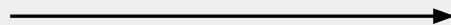
```
[0-9] +
```

```
%%
```

```
int main() {  
    yylex();  
}
```



Opcije i definicije



Pravila



Korisnički kod

```
%option noyywrap
```

```
$ flex --noyywrap jezik.1
```

Kada flex dobije indikaciju da je završeno čitanje sa jednog ulaza tada on proverava vrednost funkcije yywrap().

Povratna vrednost ukazuje na to da li se nastavlja sa sledećim ulazom (npr. novi fajl) [povratna vrednost je 0] ili je obrada završena [povratna vrednost je 1].

Ako nije definisana ova funkcija onda:

- ili je potrebno definisati gore navedenu opciju (kao da je yywrap vratio 1)
- ili se mora linkovati biblioteka koja ima ovu funkciju (libfl biblioteka, povratna vrednost 1)

Pokretanje:

```
$ flex jezik.1
```

Nakon izvršavanja kreiran je fajl `yy.lex.c`

(> 1500 redova koda)

```
#line 3 "lex.yy.c"

#define YY_INT_ALIGNED short int

/* A lexical scanner generated by flex */

#define FLEX_SCANNER
#define YY_FLEX_MAJOR_VERSION 2
#define YY_FLEX_MINOR_VERSION 6
#define YY_FLEX_SUBMINOR_VERSION 4
#if YY_FLEX_SUBMINOR_VERSION > 0
#define FLEX_BETA
#endif

/* First, we deal with platform-specific or compiler-specific issues. */

/* begin standard C headers. */
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

/* end standard C headers. */

/* flex integer type definitions */
```



Kompajliranje:

```
$ gcc lex.yy.c
```

```
$ gcc -o jezik lex.yy.c
```

*Opcija **-o** : definisanje naziva programa. Ako je izostavljeno biće kreiran **a.out** fajl.*

Pokretanje (interaktivno):

`$ ./jezik`

ctrl+D - izlazak

```
→ 123
→ abc
→ 123abc
→ abc
→ ab123c
abc
```

- sve dok ulazni string odgovara nekom regularnom izrazu –  
ne postoji podrazumevana akcija  
odnosno pročitani tekst se preskače i nastavlja se sa radom  
(za sada je sve u redu, nastavi)
- ukoliko pročitano ne odgovara nijednom regularnom izrazu – podrazumevana akcija  
je kopiranje tog teksta na izlaz

## jezik.1

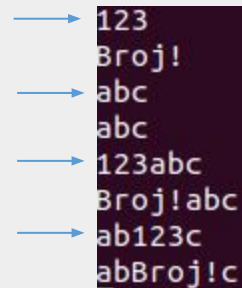
```
%option noyywrap
```

```
%%
```

```
[0-9]+ { printf("Broj!"); }
```

```
%%
```

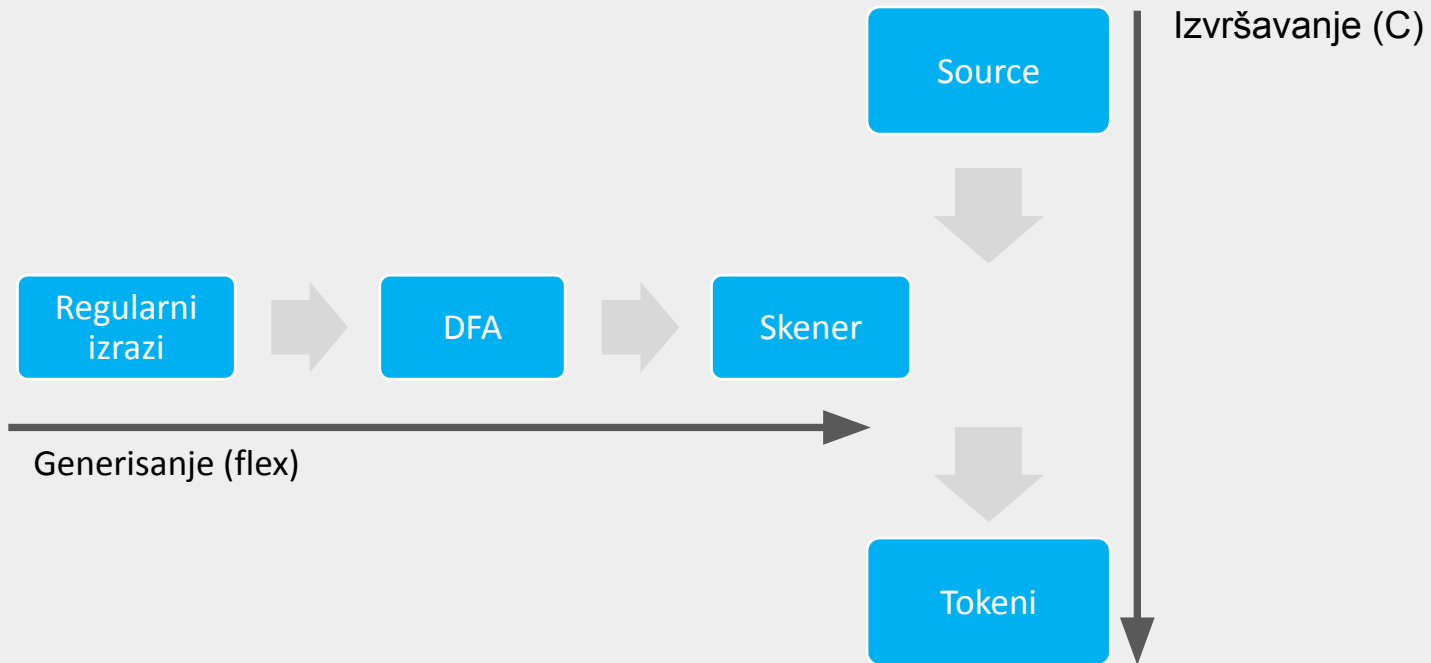
```
int main() {  
    yylex();  
}
```



A terminal window with a dark purple background showing the output of the lexer. Blue arrows point to each line of output. The output consists of six lines: '123', 'Broj!', 'abc', 'abc', '123abc', and 'Broj!abc'. The last two lines are concatenated on the same line as the previous ones, resulting in 'ab123c' and 'abBroj!c'.

```
→ 123  
→ Broj!  
→ abc  
→ abc  
→ 123abc  
→ Broj!abc  
→ ab123c  
→ abBroj!c
```

Možemo dodati korisničke  
akcije



Neki problemi:

- “<=” treba prepoznati kao jedan operator  $\leq$ , a ne kao dva:  $<$  i  $=$

“Maximal munch” rule

Pronalazi se najduži string koji odgovara nekim RE

- “if” treba prepoznati kao ključnu reč, a ne kao identifikator koji je definisan, na primer, kao `[a-zA-Z][a-zA-Z0-9]*`

Bira se prvi pronađeni RE  $\rightarrow$  Npr, patern za if treba navesti pre patern za identifikator

Flex nije jedini alat

Nastao kao unapređenje Lex alata

Koriste se i drugi: Antlr, re2c, javaCC,...

Prednosti i mane korišćenja alata naspram samostalno (ručno) implementiranog?

## Zadaci (neobavezno)

Pogledati Antlr.

Kako izgledaju formalne specifikacije leksike nekih popularnih jezika?

Linkovi Flex:

<https://github.com/westes/flex/>

<https://westes.github.io/flex/manual/>

Pogledati zbirke na acs sajtu.