# Programovací jazyky a překladače

**Přednáška 2,** poznámky

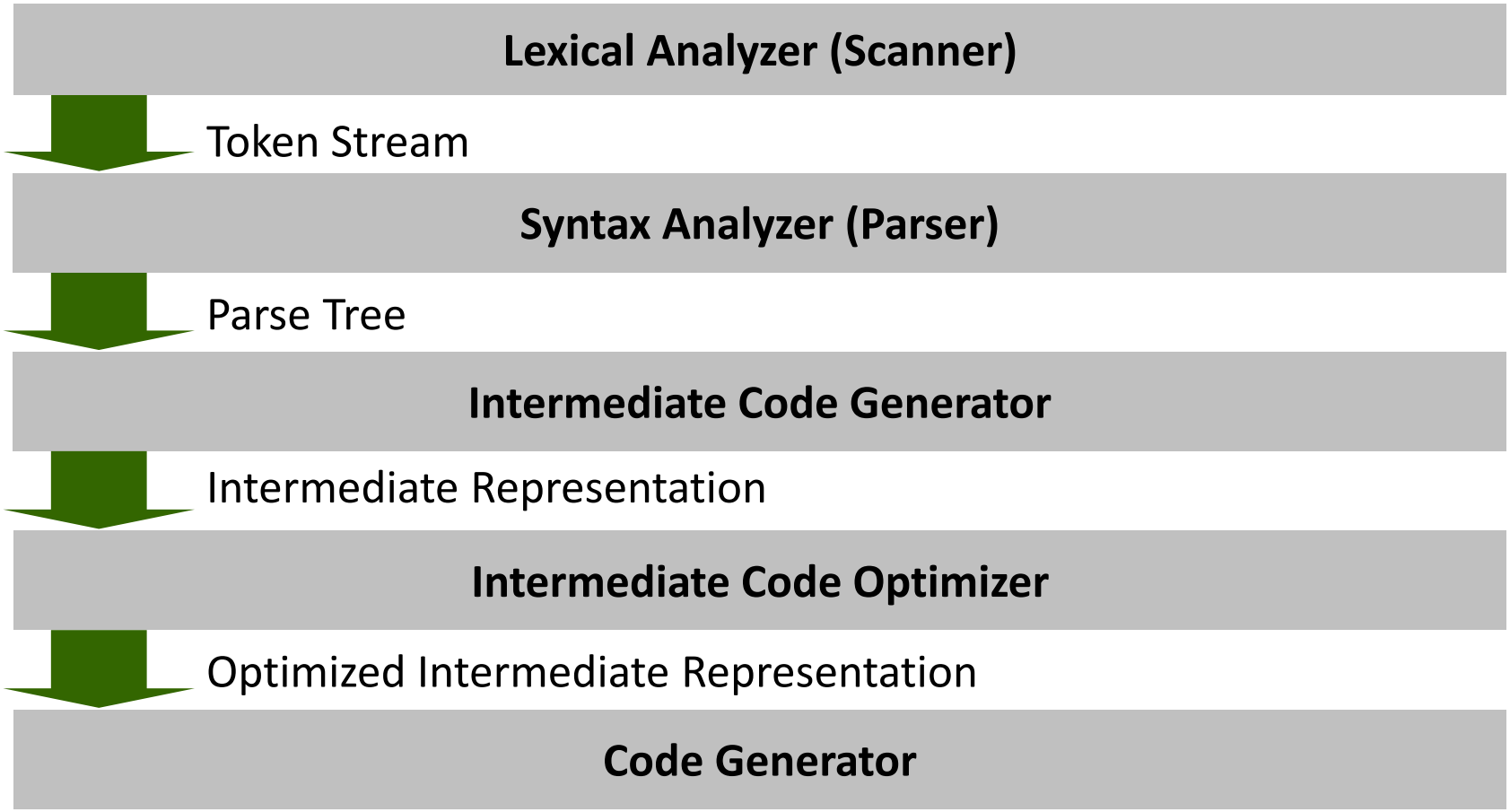
**Jan Voráček**
VŠPJ Jihlava, voracek@vspj.cz

# Dnešní témata

- Prezentace
  - Daniel Dobrovolný: Princip práce překladače
  - Jiří Šimek: Rozdíl mezi překladačem a interpretem
  - Vojtěch Bouchner: LEX
- Rámcový princip práce překladače
- LEX: generátor lexikálních analyzátorů
- Klasifikace formálních gramatik a jazyků

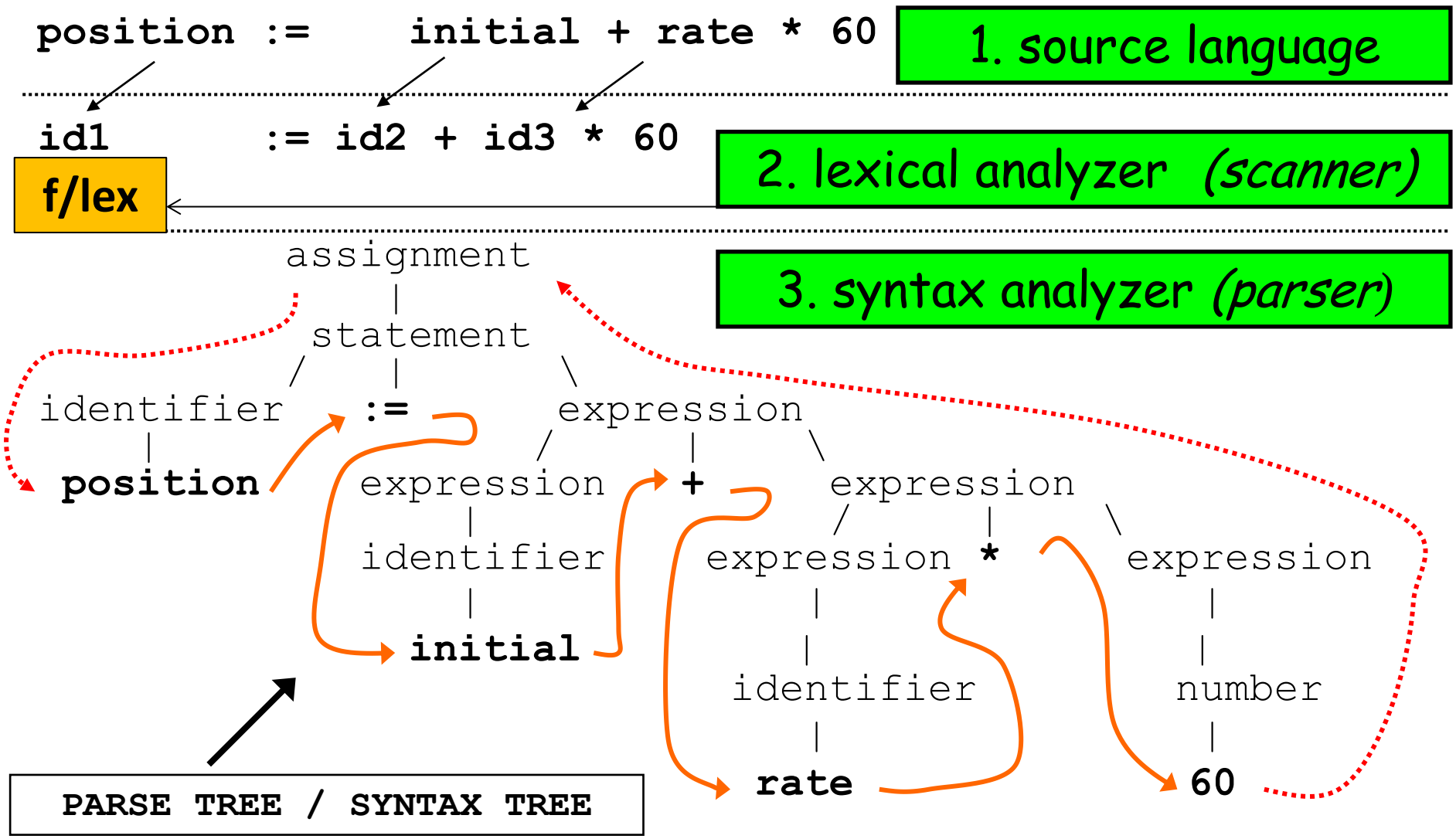
# Anatomy of compiler

**Program (character stream)**

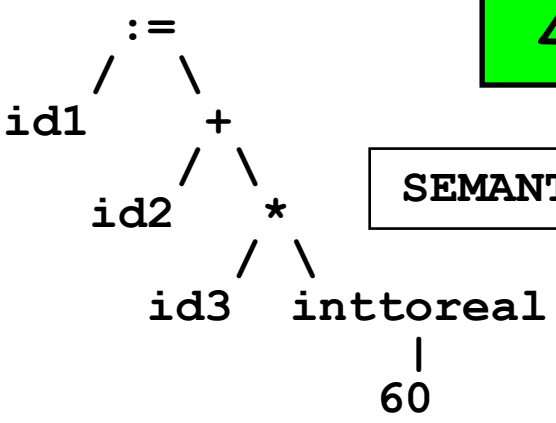



**Assembly code**

# ASU example 1/2



# ASU example 2/2




## 4. semantic analyzer

SEMANTIC TREE, DAG

SYMBOL TABLE

| | | |
|---|---|---|
| 1 | position | .... |
| 2 | initial | .... |
| 3 | rate | .... |

## 5. intermediate code generator

```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1   := temp3
```

## 6. code optimizer

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

## 7. code generator

```
MOVF id3,   R2
MULF #60.0, R2
MOVF id2,   R1
ADDF R2,    R1
MOVF R1,    id1
```

source destination

# LEX

V tomto kontextu se nejedná o zákon, ale o

program pro generování

**lexikálních analyzátorů**

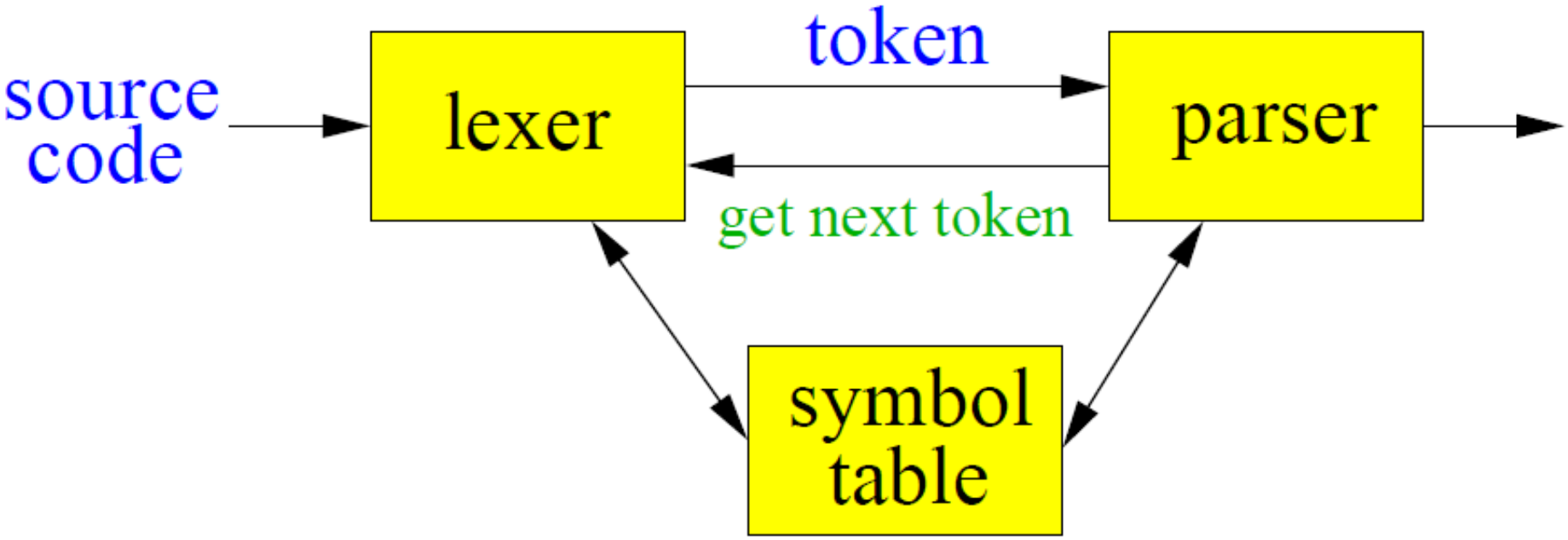
Lexikální analýza se realizuje **konečným automatem**. **Gramatiky**, generující lexikální elementy se nazývají **regulární** nebo **lineární**. **Regulární výrazy** reprezentují další (technickou) možnost zápisu generující struktury.

# Příklady regulárních výrazů

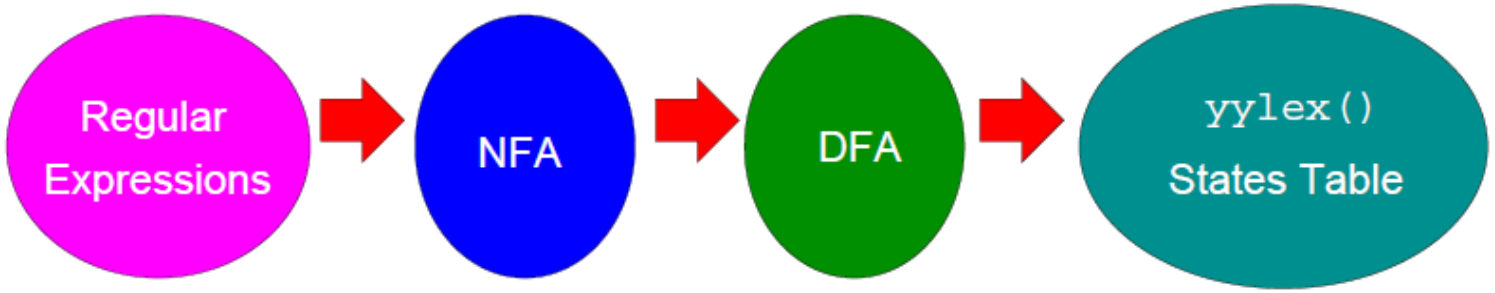
http://epaperpress.com/lexandyacc/prl.html

| Expression | Matches |
|---|---|
| abc | abc |
| abc* | ab abc abcc abccc ... |
| abc+ | abc, abcc, abccc, abcccc, ... |
| a(bc)+ | abc, abcbc, abcbcbc, ... |
| a(bc)? | a, abc |
| [abc] | one of: a, b, c |
| [a-z] | any letter, a through z |
| [a\-z] | one of: a, -, z |
| [-az] | one of: - a z |
| [A-Za-z0-9]+ | one or more alphanumeric characters |
| [ \t\n]+ | whitespace |
| [^ab] | anything except: a, b |
| [a^b] | a, ^, b |
| [a\|b] | a, \|, b |
| a\|b | a, b |

# *lex* v kontextu dalších základních struktur



# *lex*, generátor lexikálních analyzátorů




lex.yy.c

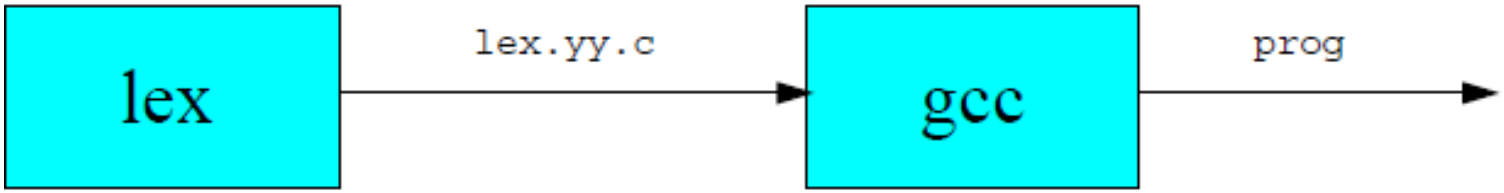
prog.l

```
Declarations
%%
Transition rules
%%
C procedures
main() {
    yylex();
}
```

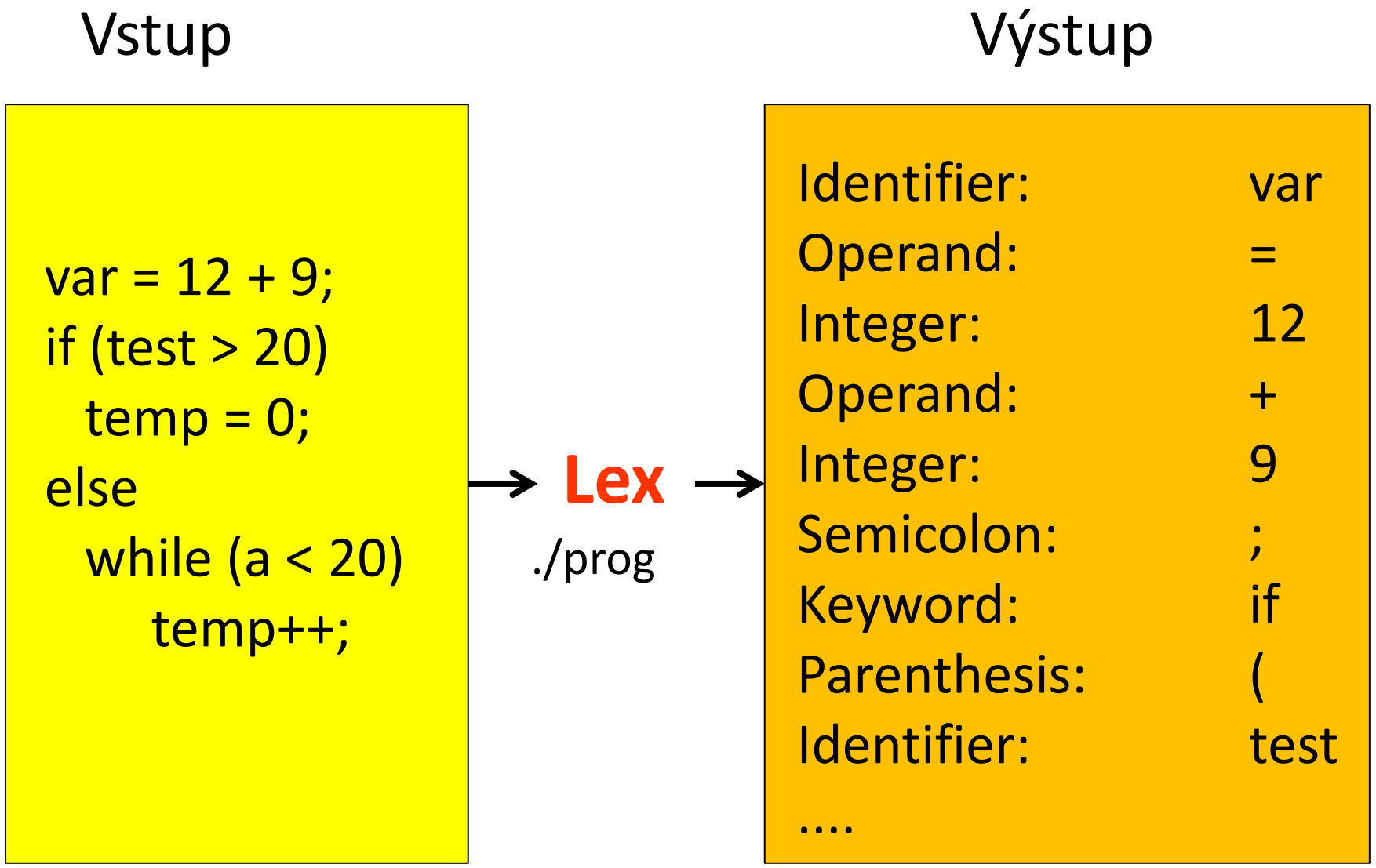
%{
C deklarace
%}
Lex definice
**%%**
**Pravidla a akce: jediná povinná část kódu**
%%
C kód

lex prog.l

gcc –o prog lex.yy.c -ll



# *lex*, ukázka činnosti

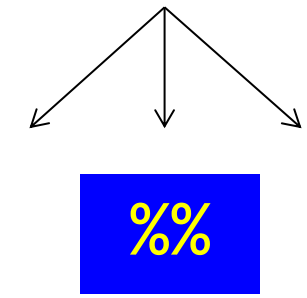


# *lex,* tři jednoduché příklady

```
%%
.
\n
%%

int yywrap(void) {
    return 1;
}

int main(void) {
  yylex();
return 0;
}
```




DEFAULT

```
%%
.     ECHO;
\n    ECHO;
%%

int yywrap(void) {
    return 1;
}

int main(void) {
  yylex();
return 0;
  }
```

```
#define ECHO fwrite(yytext, yyleng, 1, yyout)
```

# *lex*, příklady speciálních výrazů

| Metacharacter | Matches |
|---|---|
| `.` | any character except newline |
| `\n` | newline |
| `*` | zero or more copies of the preceding expression |
| `+` | one or more copies of the preceding expression |
| `?` | zero or one copy of the preceding expression |
| `^` | beginning of line / complement |
| `$` | end of line |
| `a\|b` | `a` or `b` |
| `(ab)+` | one or more copies of `ab` (grouping) |
| `[abc]` | `a` or `b` or `c` (character class) |
| `a{3}` | 3 instances of `a` |
| `"a+b"` | literal "`a+b`" (C escapes still work) |

Zdroj: http://epaperpress.com/lexandyacc/prl.html

# *lex*, složitější příklad

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
dgt    [0-9]
%%
{dgt}+   return atoi(yytext);
%%
void main()
{
  int val, total = 0, n = 0;
  while ( (val = yylex()) > 0 ) {
    total += val;
    n++;
  }
  if (n > 0) printf("ave = %d\n", total/n);
}
```

← C/C++ deklarace

← *lex*, definice

← **Pravidla**

← Hlavní program

# Řešení konfliktů v *lex*_u

Pokud aktuálnímu vstupu vyhovuje více předdefinovaných vzorů, postupuje *lex* následujícím způsobem:

- Vrátí **nejdelší** shodný token ( např. <= )
- V případě shodných vzorů téže délky vrátí ten, který je ve specifikaci uveden **dříve** (např. klíčová slova se definují před identifikátory)
- Nenajde-li shodu, **kopíruje** následující znak na *stdout*

| Definice | Vstup | Token | Důvod |
|---|---|---|---|
| **if** | **if** | **if** | **je dříve** |
| **ifthen** | **ifthen** | **ifthen** | **je delší** |

Problémy s potenciálně dlouhými tokeny, jak se scanneru mohou při nedostatečné definici jevit např. komentáře v C, mohou být řešeny **zařazením dodatečných stavů** (viz cvičení).

# *lex*, příklady spec. funkcí

http://epaperpress.com/lexandyacc/prl.html

| Name | Function |
|---|---|
| `char *yytext` | pointer to matched string |
| `int yyleng` | length of matched string |
| `int yylval` | value associated with token |
| `FILE *yyin` | input stream pointer |
| `FILE *yyout` | output stream pointer |
| `int yylex(void)` | call to invoke lexer, returns token |
| `char* yymore(void)` | return the next token |
| `int yyless(int n)` | retain the first n characters in yytext |
| `int yywrap(void)` | wrapup, return 1 if done, 0 if not done |
| `ECHO` | write matched string |
| `REJECT` | go to the next alternative rule |
| `INITAL` | initial start condition |
| `BEGIN` | condition switch start condition |

# Příklady komentářů v ANSI C

```
/* jednoduchý komentář */
/* také /* komentář */
/* komentář s * hvězdičkou uprostřed */
/* víceřádkový
     komentář
*/
/* dva komentáře */ ..... /* na řádku */
* komentář s hvězdičkou na konci **/
```

Podobné problémy mohou vznikat i při práci s:

“textovými literály”.

# RE pro komentáře v C

Takto? `"/*".*"*/ "`

Nebo takto? `"/*"(.|\n)*"*/"`

Problém dlouhých vzorů:


Možný RE pro zpracování komentářů v C :

```
"/*" ([^*]|\*+[^*/])* \*+"/"
```

# (f)lex jako stavový automat

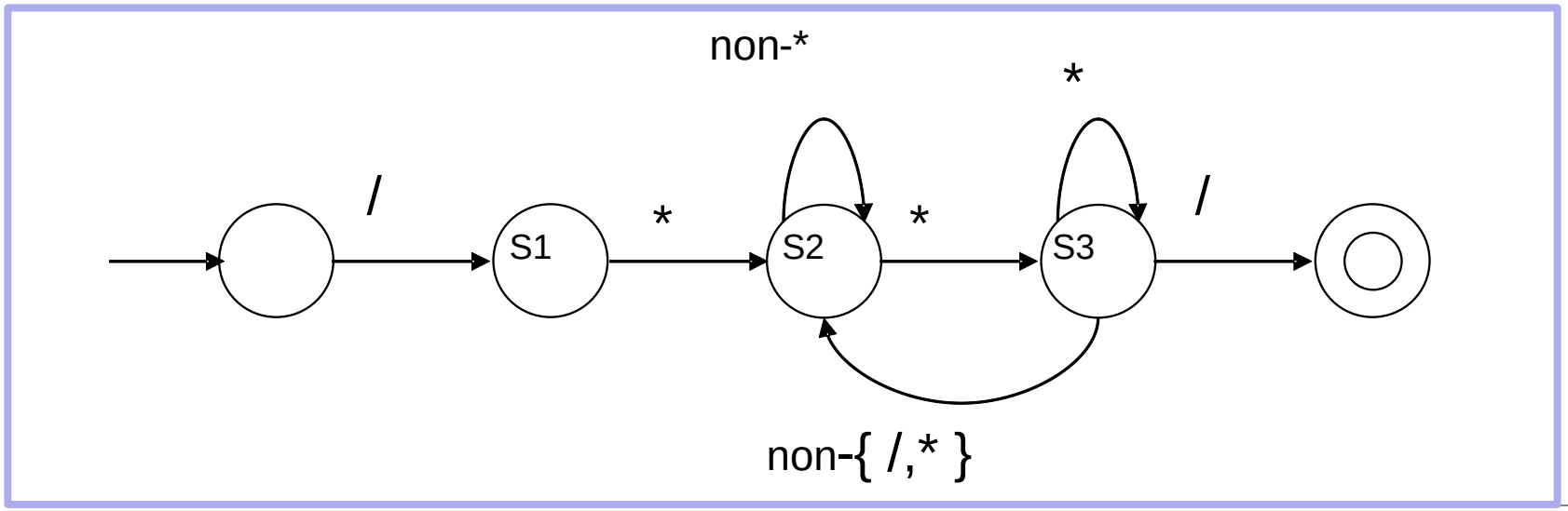
- %s <STAV> : zahrnuté stavy (všechny),
- %x <STAV> : vyloučené stavy (pouze specifické)

```
%x comment
%%
        int line_num = 1;

"/*"          BEGIN(comment);
<comment>[^*\n]*          /* ignoruj cokoli mimo '*',       */
<comment>"*"+[^*/\n]*     /* ignoruj '*', nenásledované '/' */
<comment>\n               ++line_num;
<comment>"*"+"/"          BEGIN(INITIAL);
```

# Příklad

```
%x S1, S2, S3
%%
"/"             BEGIN(S1);
<S1>"*"         BEGIN(S2);
<S2>[^*]        ;     /* zůstaň v S2 */
<S2>"*"         BEGIN(S3);
<S3>"*"         ;     /* zůstaň v S3 */
<S3>[^*/]       BEGIN(S2);
<S3>"/"         BEGIN(INITIAL);
```



# Poznámky doc. Beneše, část 8b

## Konstruktory překladačů

Miroslav Beneš

Dušan Kolář

Zdroj: http://www.cs.vsb.cz/benes/vyuka/pjp/prednasky/08b-lexyacc.pdf

# Úkoly na příští týden

## Studium

- Sledovat rozpis prezentací
- Prostudovat doporučené zdroje
  - `lex/flex`

## Cvičení

- Práce s generátorem lexikálních analyzátorů `lex/flex`