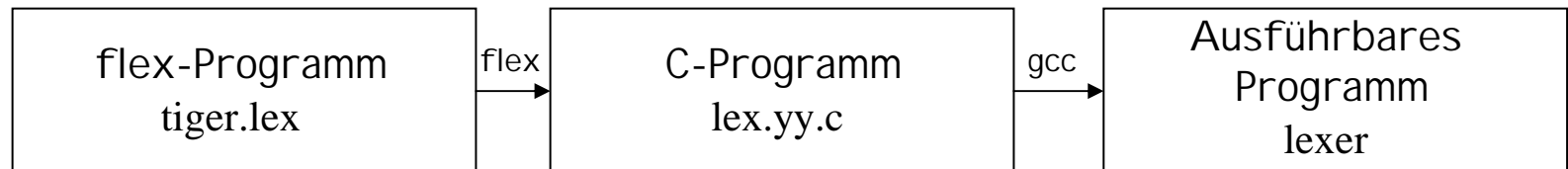


# FLEX - Einleitung

- flex ist ein Programmgenerator, welches ein C oder C++ Programm erzeugt, dass die lexikalische Analyse eines Eingabestroms durchführt
- Vorgehensweise:
  - Spezifikation reg. Ausdrücke und dazugehöriger Aktionen in Zielsprache (C oder C++) → Lexer-Spezifikation
  - Anwenden von flex auf Lexer-Spezifikation
  - Einbinden in andere Programme (z.B. Parser oder eigenständiges Programm)

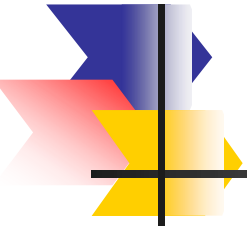




# FLEX-Lexer Spezifikation

---

- Struktur
  - Definitionsteil
  - %%
  - Regelteil
  - %%
  - Benutzerdefinierte Routinen
- Regelteil
  - Liste reg. Ausdrücke  $r_i$  und dazugehörige Aktionen  $a_i$ 
    - $r_1$              $a_1$  ;
    - $r_2$              $a_2$  ;
    - $r_n$              $a_n$  ;
  - eine Aktion besteht aus einer C- oder C++-Anweisung



# FLEX-Lexer Spezifikation (cont.)

- Beispiel 1.lex

```
/* directs flexs to provide default main() function */  
%option main
```

```
/* rule part starts here */  
%%
```

```
a      printf("Vowel %s recognized\n", "a");  
e      printf("Vowel %s recognized\n", "e");  
o      printf("Vowel %s recognized\n", "o");  
u      printf("Vowel %s recognized\n", "u");  
i      printf("Vowel %s recognized\n", "i");  
.      {printf("no vowel\n"); continue;}
```



# Reg. Ausdrücke

---

- Einfache Zeichen und Metazeichen
  - gültiges Alphabet A ist der Zeichensatz des Systems
  - Einige Zeichen haben besondere Bedeutung in regulären Ausdrücken, die sog. Metazeichen
    - \ ^ \$ . [ ] | ( ) \* + ? { } " % < > /
  - Behandlung der Metazeichen als normale Zeichen durch Voranstellen von "\" oder Einbinden in " "

- Beispiel 2.lex

```
%option main
```

```
%%
```

```
\\/\\/      printf("double slash: %s\n", "//");
```

```
"?"      printf("question mark: %s\n", "?");
```

```
\      printf("quotation mark: %s\n", "\"");
```

```
.      {printf("unknown character\n"); continue;}
```



# Reg. Ausdrücke (cont.)

- Escape-Sequenzen und Spezialzeichen
  - Für einige Bestandteile eines Textes gibt es keine Zeichen.
  - Hierfür gibt es Ersatzdarstellungen:

<code>\b</code>	Backspace
<code>\r</code>	Return (Sprung an den Anfang der aktuellen Zeile)
<code>\f</code>	Formfeed (Seitenvorschub)
<code>\n</code>	Newline (Zeilenvorschub)
<code>\t</code>	Tabulatur
<code>\ddd</code>	Zeichen, das den oktalen Wert ddd entspricht
<code>^</code>	Anfang der Zeile
<code>\$</code>	Ende der Zeile

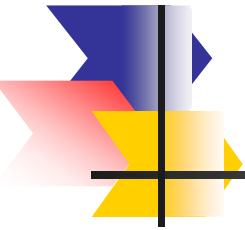
- Beispiel 3.lex

```
%option main
%%
^\\n      printf("empty line\\n");
foo$     printf("foo at the end of the line\\n");
```



# Reg. Ausdrücke (cont.)

- Zeichenklassen
  - Mengen oder Bereich von Zeichen mit „[„ und „]“, z.B.
    - [xyz] - Eines der Zeichen x,y oder z
    - [A-Z] - Ein Grossbuchstabe
  - Beliebige Zeichen mit „.“
    - . - beliebiges Zeichen aus A außer \n
  - Ausschluss von Zeichen mit „^“, z.B.
    - [^0-9] - alle Zeichen außer Zahlen,  $A \setminus \{0, \dots, 9\}$
  - Zusammengesetzte reg. Ausdrücke
    - $r \mid s$  - r oder s
    - $rs$  - r gefolgt von s
    - $r^*$  - r beliebig oft auch keinmal
    - $(r)$  - deckt r ab
    - $r^+$  - r beliebig oft, aber mindestens einmal
    - $r\{n,m\}$  - zwischen n und m Vorkommen von r
    - $r?$  - r ein- oder keinmal
    - $r/s$  - r, aber nur wenn von s gefolgt; Action sieht nur text der durch r gematcht wird



# Aktionen

- Zur Erinnerung

$r_1$	$a_1 i$
$r_2$	$a_2 i$
$\dots$	$\dots i$
$r_n$	$a_n i$

- Eine Aktion besteht aus  $m$  C- oder C++-Anweisungen,  $m \geq 0$ 
  - $m = 0$ : leere Aktion; bzw  $\{\}$
  - $m = 1$ : keine Klammerung nötig
  - $m \geq 2$ : Klammerung mit  $\{\dots\}$
- Passt kein regulärer Ausdruck, wird eine default-Aktion ausgeführt; diese schreibt die gelesenen Zeichen unverändert nach stdout
- Ausführung derselben Aktion für mehrere reguläre Ausdrücke ist möglich durch Aktionszeichen „|“  $\rightarrow$  Aktion des nächsten regulären Ausdrucks wird ausgeführt



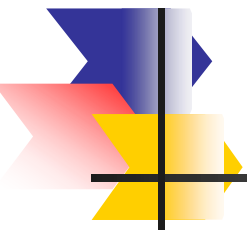
# Aktionen (cont.)

---

- Beispiel 4.lex

```
%option main
/* rule part starts here */
%%
"  "      |
\t        |
\n        {printf("whitespace or newline\n");}
```



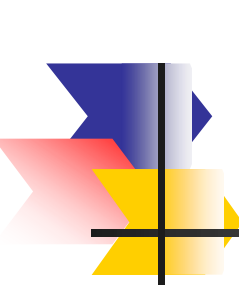


# Flex-interne Variablen, Makros und Funktionen

- `yytext` - char-Pointer oder Array; enthält den aktuell gelesenen Eingabestring
- `yylen` - enthält die Länge des aktuell gelesenen Eingabestrings
- `yylineno` - enthält aktuelle Zeilennummer des Eingabetext
- `ECHO` - Makro, welches den Inhalt von `yytext` enthält

## ■ Beispiel 5.lex

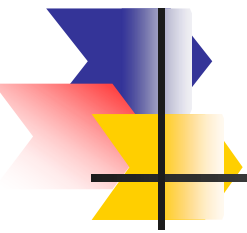
```
%{ #include <string.h> %}  
%option main yylineno  
%%  
foobar {          /* returns the reverse image of foobar*/  
    int i;  
    char *yycopy = strdup (yytext);  
    printf("token in line %d of the input file is  
lexed\n", yylineno);  
    printf("reverse image of "); ECHO; printf(" = ");  
    for(i=yylen-1; i>=0; i--)  
        printf("%c", yycopy[i]);      }
```



# Flex-interne Variablen, Makros und Funktionen (cont.)

- Die folgenden Funktionen bieten Lookahead- und Eingabemanipulations-Möglichkeiten:
- `yylless(n)` - zurückschreiben aller Zeichen in `yytext` mit Ausnahme der ersten `n` Zeichen
- `input()` - liefert nächstes Zeichen des Eingabetextes
- `output(c)` - gibt Zeichen `c` auf `stdout` aus
- `unput(c)` - schiebt Zeichen `c` wieder zurück in den Eingabetext
- Beispiel 6.lex:

```
%option main
%%
"/*"      {      int c;
              for(;;) {
                  while ( ( c = input() ) != '*' && c != EOF )
                      ; /* eat up text of comment */
                  if ( c == '*' ) {
                      while ( ( c = input() ) == '*' )
                          ; /* eat up asterisks */
                      if ( c == '/' )
                          break; /* found the end */
                  }
                  if ( c == EOF ) {
                      error( "EOF in comment" );
                      break;}}}
```



# Flex-interne Variablen, Makros und Funktionen (cont.)

- Die `yywrap()`-Funktion wird beim Erreichen des Dateiendes der Eingabe aufgerufen:
  - Kann überschrieben werden
  - Rückgabewert  $\neq 0$   $\rightarrow$  Programm wird verlassen
  - Rückgabewert  $= 0$   $\rightarrow$  Programm arbeitet auf neuer Eingabe weiter
  - durch die Direktive `%option nnyywrap` wird vom Lexer die Default-Implementierung, die 1 zurückgibt, verwendet
  - N.B. Nach dem Anruf von `yywrap()` ist weiterhin der zuletzt aktive Zustandskontext gültig, d.h. es wird nicht nach `INITIAL` zurückinitialisiert



# Definitionsteil

---

- **Funktionen des Definitionsteils:**
  - Einbinden von C- oder C++-Code
  - Spezifikation sog. regulärer Definitionen zur Vereinfachung regulärer Ausdrücke
  - Definition von Startbedingungen
- Einbinden von C-Code
- Entweder durch

```
%{  
  C-Anweisungen  
%}
```
- oder
  - C-Anweisung\_1;
  
  - C-Anweisung\_N;
- Anweisungen sind hier Präprozessor-Anweisungen und Variablen-Deklarationen
- Hilfsfunktionen werden im Teil „Benutzerdefinierte Routinen“ eingebunden



# Definitionsteil (cont.)

- Reguläre Definitionen

- dienen der Vereinfachung von regulären Ausdrücken
- reguläre Ausdrücke haben die Form:

$\text{name}_1$	$r_1$
$\text{name}_2$	$r_2$
$\text{name}_n$	$r_n$

- wobei  $\text{name}_1, \dots, \text{name}_n$  die Bezeichner der Ausdrücke sind und  $r_i$  ein regulärer Ausdruck über dem Alphabet  $A \cup \{\text{name}_1, \text{name}_2, \dots, \text{name}_n\}$

- Beispiel 7.lex

```
%option main
L      [a-zA-Z_]
D      [0-9]
ID     {L}({D}|{L})*
/* rule part starts here */
%%
{ID}   printf("%s%s", "Identifier: ", yytext);
```



# Definitionsteil (cont.)

## ■ Startbedingungen

- erlauben zustandsabhängige Aktivierung von regulären Ausdrücken
- Definition von Startbedingungen über
  - `%s name1 name2 ... namen`
- Aktivierung eines Zustands im Aktionsteil eines regulären Ausdrucks mit

`Begin namei`

- Der Normalzustand, in dem alle regulären Ausdrücke ohne Startbedingung aktiv sind, ist 0, d.h. mit

`Begin(0);`

oder

`Begin(INITIAL);`

wechselt man in den Normalzustand

- Kennzeichnung von zustandsabhängigen reg. Ausdrücken

`<namei>regAusdruck      Aktion;`

oder

`<namei, namej>regAusdruck      Aktion;`



# Definitionsteil (cont.)

## Beispiel 8.lex

```
%option main
%s      COMMENT
/* rule part starts here */
%%
<COMMENT>" */"    BEGIN( INITIAL );
<COMMENT>.|       |
<COMMENT>\\n      ;
" /* "           BEGIN(COMMENT); /* due to inclusive declaration the
                  rule is always active; equivalent to
                  <COMMENT,INITIAL>" /* " ... */
```



# Benutzerdefinierte Routinen

Funktion des dritten Teils:

- Code wird unverändert nach `lex.yy.c` kopiert
- Einbinden von benutzerdefinierten Funktionen, die in den Aktionen benutzt werden können
- Einbinden des `main`-Programms oder der Funktion `yywrap()`

■ Beispiel 9.lex

```
int lines=0;
%%
.      ;
\n     lines++;
%%
yywrap() {
    printf("Number of source code lines: %d\n", lines);
    return 1;}
main(argc, argv)
int argc;
char **argv; {
    yylex();
    return 0;}
```