Systematische Scanner-Implementierung

grober Aufbau des lex-input-files (<name>.l)
Erster Teil: optional, besteht aus:
Vorspann wird im Programm von lex unverändert eingesetzt:
meist #includes, #defines, C-Code
Abkürzungen
Abkürzende Bezeichnungen für reguläre Ausdrücke -> nur für Lex intern benutzt
%%
Zweiter Teil: obligatorisch
Definition der **regulären Ausdrücke** für die Tokens
%%
Dritter Teil: optional
Nachspann, meist C Prozeduren, die 1:1 übertragen werden.

Systematische Scanner-Implementierung

Aufbau im Detail:
Erster Teil:
Steuerzeichen innerhalb des ersten Teils:
%{ für Beginn Vorspann
%} für Ende Vorspann

Danach folgen Definitionen für abkürzende Schreibweise in den regulären Ausdrücken des zweiten Teils in folgender Form:
Bezeichner regulärer Ausdruck
z.B.: Ziffer [0-9]
Bezeichner werden mittels { Bezeichner } referenziert.
Zahl {Ziffer}+

uebung8_L

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
%}
ziffer [0-9]
letter [a-zA-Z]
sign [+-]
zeichen [_]
%%
"+" {printf("Operator + gefunden!\n"); }
"if" {printf("Schluesselwort if gefunden!\n");}
"while" {printf("Schluesselwort while gefunden!\n");}
{sign}?[0-9]+ {printf("Ganzzahl gefunden! Zahlenwert=%i \n",atoi(yytext));}
{letter}({letter}|{ziffer}|{zeichen})* {printf("Bezeichner gefunden! Name=%s \n",yytext); }
%%
yywrap()
{
 return 0;
};

void main()
{
 do { yylex (); }
 while (! feof(stdin));
}
```

Vorspann

Teil 1

Teil 2

Teil 3

Systematische Scanner-Implementierung

Aufbau im Detail:

Zweiter Teil:

Folge von Zeilen der Form:

regulärer Ausdruck {Aktionen }

{Aktionen} sind C++-statements, die ausgeführt werden, wenn regulärer Ausdruck erkannt wird

Die Aktionen werden unverändert in lex.yy.c übernommen.

Unix-Notation für reguläre Ausdrücke

Bereichsangabe: [0-9] - 1 Ziffer 0-9

Aufzählung: [abcx] - 1 Zeichen, das a, b, c oder x ist

Aufzählung von Bereichen: [a-zA-Z] - 1 Zeichen, das klein- oder großbuchstabe ist

*, \, +, . sind Sonderzeichen

falls benötigt: "*" oder *

Beispiel Gleitkommazahl:

```
{sign}?(({ziffer}+"."{ziffer}*)|({ziffer}*"."{ziffer}+))
```

akzeptiert auch .23, 2.

Vorzeichen mit/ohne

Umgang und Arbeitsweise mit flex:

