

Übungen zu Systemnahe Programmierung in C (SPiC)

Peter Wägemann, Heiko Janker, Moritz Strübe, Rainer Müller
(Lehrstuhl Informatik 4)



Wintersemester 2014/2015



Inhalt

Zustandsmaschine

Darstellung von Zustandsmaschinen
Festlegen von Zuständen
Zustandsabfragen

Aufgabe: Ampel

Aufgabe: trac

Kommandozeilenparameter
Fehlerbehandlung getchar()



Inhalt

Zustandsmaschine

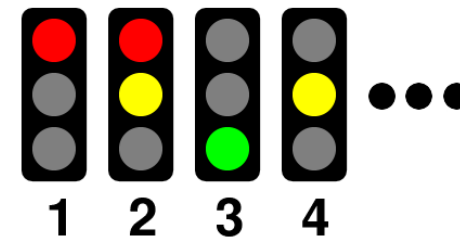
Darstellung von Zustandsmaschinen
Festlegen von Zuständen
Zustandsabfragen

Aufgabe: Ampel

Aufgabe: trac



Ampel als Zustandsmaschine



■ Zustände:

1. Auto: grün, Fußgänger: rot
2. Auto: gelb, Fußgänger: rot
3. ...

■ Zustandswechsel

- Auto: grün, Fußgänger: rot → Auto: gelb, Fußgänger: rot
- ...

■ Auslöser Zustandswechsel

- Direkt: Umschaltung
- Indirekt: Drücken des Tasters



Festlegen von Zuständen

- Festlegung durch Zahlen ist fehleranfällig
 - Schwer zu merken
 - Wertebereich nur bedingt einschränkbar

- Besser enum:

```
1 enum strategy { RANDOM, SEARCH };
2 enum strategy my_strategy = RANDOM;
```

- Mit `typedef` noch lesbarer:

```
1 typedef enum {RANDOM, IMMEDIATE, SEARCH} strategy;
2 strategy my_strategy = IMMEDIATE;
```



switch-case-Anweisung

```
1 switch ( my_strategy ) {
2 case RANDOM:
3     ...
4     break;
5 case SEARCH:
6     ...
7     break;
8     ...
9 default:
10    // maybe invalid state
11    ...
12    break;
13 }
```

- Vermeidung von if-else-Kaskaden
- switch-Ausdruck muss eine Zahl sein (besser ein enum-Typ)
- break-Anweisung nicht vergessen!
- Ideal für die Abarbeitung von Systemen mit verschiedenen Zuständen
⇒ Implementierung von Zustandsmaschinen



Inhalt

Zustandsmaschine

Aufgabe: Ampel

Aufgabe: trac



Ampel

- Implementierung einer Zustandsmaschine
- Kein aktives Warten: kein `sb_timer_delay()!!!`
- „Großer Zustand“
 - Abarbeitung der Ampel-Phase
 - `enum TRAFFIC_LIGHT_STATE {...}`
 - case `CAR_RED_STATE`: ...
- „Kleiner Zustand“
 - `while(1)`-Durchlauf
 - Schlafenlegen wenn möglich
 - Steuerung durch Flags
 - if-Abfragen in den „großen Zuständen“



Zustandsmaschine

Aufgabe: Ampel

Aufgabe: trac

Kommandozeilenparameter

Fehlerbehandlung getchar()



- Ähnlich wie das Kommando `tr` in Unix-artigen Betriebssystemen
- Manual Page: `tr(1)`
- Was wird benötigt?
- Welche Fehlerüberprüfungen sind notwendig?
- Wer liefert die umfangreichste Implementierung?
- Nützliche Funktionen:
 - `fprintf(3)` („man 3 fprintf“ im Terminal eingeben)
 - `strlen(3)`
 - `getchar(3)`
 - `putchar(3)`
 - `ferror(3)`
 - `perror(3)`
 - `exit(3)`
 - ...



Kommandozeilenparameter

```
1 ...
2 int main(int argc, char *argv[]){
3     strcmp(argv[argc - 1], ... )
4     ...
5     return EXIT_SUCCESS;
6 }
```

- Übergabeparameter:
 - `main()` bekommt vom Betriebssystem Argumente
 - `argc`: Anzahl der Argumente
 - `argv`: Vektor aus Strings der Argumente (Indices von 0 bis `argc-1`)
- Rückgabeparameter:
 - Rückgabe eines Wertes an das Betriebssystem
 - Zum Beispiel Fehler des Programms: `return EXIT_FAILURE;`



Fehlerbehandlung getchar()

```
1 int c;
2 while ((c=getchar()) != EOF) {
3     ...
4 }
5
6 /* EOF oder Fehler? */
7 if(ferror(stdin)) {
8     /* Fehler */
9     ...
10 }
```

- „`fgetc()`, `getc()` and `getchar()` return the character read as an unsigned char cast to an int **or EOF on end of file or error.**“
- Wie kann man den Fehlerfall von EOF unterscheiden?
⇒ `ferror(3)`

