

# Programmierung in C: Vermischtes (Teil 2)

Inhalt:

- **Fehlersuche in Programmen**
- Organisation des Quellcodes

# Fehlersuche in Programmen

Möglichkeiten:

- **printf()-Debugging**

Ausgaben mit printf(), die schrittweise den Programmfortschritt und die Variablenwerte dokumentieren

- **Nutzung s.g. Assertions**

Formulierung von Ausdrücken, die zur Laufzeit des Programm erfüllt sein müssen, d.h. logisch wahr ergeben. Diese Annahmen können per assert()-Funktion in das Programm eingebaut werden. Das Programm wird abgebrochen, wenn eine Annahme verletzt wird.

- **Ausführung mit einem Debugger**

Ausführung des Programms unter Beobachtung der Anweisungsreihenfolge und der Variablenwerte

# Fehlersuche in Programmen

Beispiel einer FiFO-Warteschlange mit Fehler:

```
typedef struct fifo { int value;
                     struct fifo *next;} fifo_t;
fifo_t *fliste;
int main()
{ fliste=NULL;
  int v; char s;
  // Liste Anlegen
  fifo_input(1);
  fifo_input(2); // ... weitere

  // Liste Ausgeben
  do{ v=fifo_output(&s);
      if (s==0) break;
      printf("value: %d\n",v);
    } while (1);
  return 0;
}
```


```
int fifo_output(char *success){
    int value;
    fifo_t *oe;
    value= fliste->value;
    oe = fliste;
    fliste = fliste->next;
    free(oe);
    *success=1;
    return value;
}
```

Der Fall, wenn FIFO-Puffer  
leer ist, wurde in der Funktion  
vergessen

# Fehlersuche in Programmen

Beispiel für printf()-Debugging:

```
int fifo_output(char *success){  
    int value;  
    fifo_t *oe;  
    printf("Entry fifo_output fliste =%x \n", fliste);  
    value= fliste->value;  
    oe = fliste;  
    fliste = fliste->next;  
    free(oe);  
    *success=1;  
    printf("Exit fifo_output\n");  
    return value;  
}
```



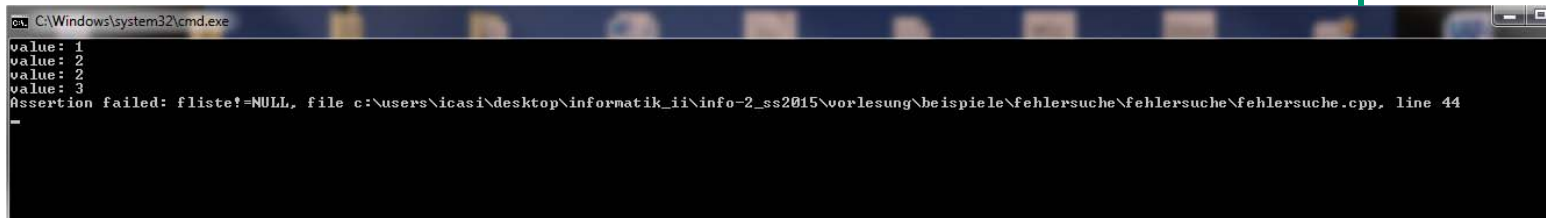
Hier erkennt man, dass fliste den Wert NULL annimmt und das Programm danach abstürzt.

# Fehlersuche in Programmen

Beispiel für Assertion:

```
#include <assert.h>
int fifo_output(char *success)
{   int value;
    fifo_t *oe;
    assert (fliste!=NULL);
    value= fliste->value;
    oe = fliste;
    fliste = fliste->next;
    assert (oe!=NULL);
    free(oe);
    *success=1;
    return value;
}
```

Annahme fliste!=NULL  
wird verletzt.



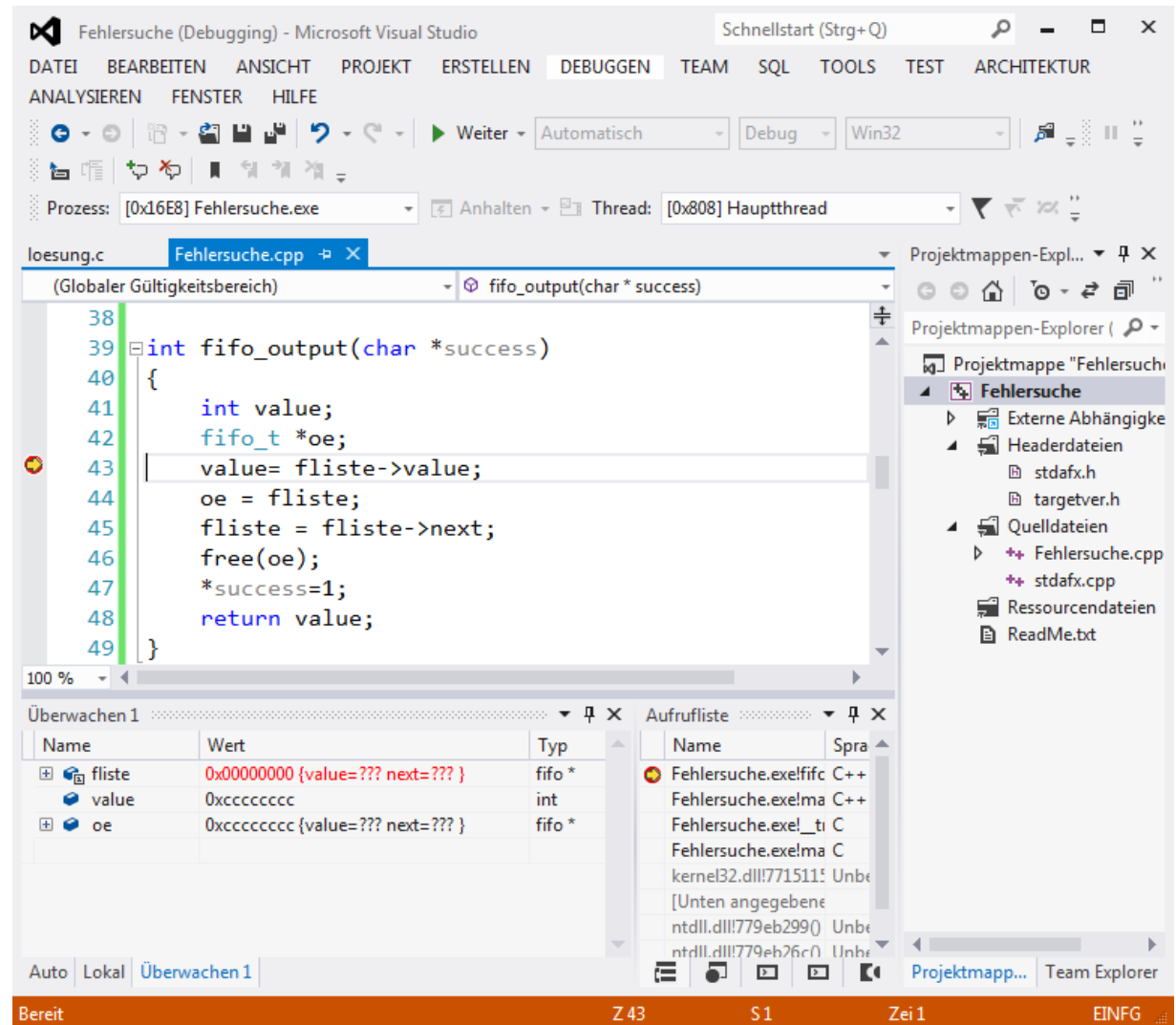
The screenshot shows a Windows command prompt window with the title "C:\Windows\system32\cmd.exe". The output of the program is as follows:

```
value: 1
value: 2
value: 2
value: 3
Assertion failed: fliste!=NULL, file c:\users\icasi\desktop\informatik_ii\info-2_ss2015\vorlesung\beispiele\fehlersuche\fehlersuche\fehlersuche.cpp, line 44
```

# Fehlersuche in Programmen

Beispiel für  
Debugging:

- Ausführen innerh. Der Entwicklungs-umgebung
- Vorheriges Setzen von Haltepunkten (engl. Breakpoint)
- Hinzufügen von Überwachungen für ausgewählte Variable



# Programmierung in C: Vermischtes (Teil 2)

Inhalt:

- Fehlersuche in Programmen
- **Organisation des Quellcodes**

# Organisation des Quellcodes

- Projekte können aus mehreren C-Quelldateien bestehen.
- Oft werden inhaltlich verwandte Funktionen in einer C-Quelldatei zusammen implementiert, z.B. `stack_push()`, `stack_pop()` in `stack.c`.
- Eine C-Quelldatei kann Funktionen aus anderen Quelldateien aufrufen.
- Alle Nicht-Hauptquelldateien erhalten i.d.R. eine s.g. Headerdatei, die Typdeklarationen und Funktionsköpfe enthält. Diese Headerdateien werden von anderen eingebunden (`#include`).
- Nur eine C-Quelltextdatei kann die `main()`-Funktion enthalten, das ist die Hauptquelldatei.
- Zusammengehörige C-Quelltextdateien (`*.c/*.cpp`) und Headerdateien (`*.h`) werden oft als s.g. Modul bezeichnet.



# Organisation des Quellcodes

Beispiel: Ein Hauptprogramm raketenstart.c nutzt Countdown-Funktionen, die in countdown.c programmiert sind.

```
// Hauptprogramm raketenstart.c
#include "countdown.h"
#define STEPS 10
int main()
{
    cntdwn_t *c;           // Typ aus countdown.h
    int raketenstart=0;
    c= countdown_init(STEPS);

    printf("Nach %d Zeiteinheiten wird die Rakete gestartet ...\n");

    while (!countdown_over(c))
    {
        countdown_tick(c);
        printf("Eine Zeiteinheit verstrichen ...\n");
    }
    printf("Countdown jetzt abgelaufen, Rakete gestartet, Guten Flug!\n");
    raketenstart=1;
    countdown_finish(c);
    return 0;
}
```

# Organisation des Quellcodes

countdown.c enthält alle Funktionen zur Handhabung des Countdown

```
#include "countdown.h"
#include <stdlib.h>

cntdwn_t* countdown_init(unsigned int value)
{ cntdwn_t *c = (cntdwn_t*) malloc(sizeof(cntdwn_t));
  c->start_value = value;
  c->curr_value = value;
  return c;
}

int countdown_tick(cntdwn_t *cd)
{
  if (cd->curr_value>0)
  { (cd->curr_value)--;
    return 1;
  }
  else
    return 0;
}

...
```

# Organisation des Quellcodes

countdown.h ist die zu countdown.c gehörende Headerdatei

```
typedef struct {  
    unsigned int start_value;  
    unsigned int curr_value;  
} cntdwn_t;  
  
cntdwn_t* countdown_init(unsigned int value);  
  
int countdown_tick(cntdwn_t* cd);  
  
int countdown_over(cntdwn_t* cd);  
  
int countdown_reset(cntdwn_t* cd);  
  
void countdown_finish(cntdwn_t* cd);
```