Konzepte des prozeduralen Programmierens

Stand September 2022

- Prof. Dr. Oliver S. Lazar / Christian Frank

7 Variablen und Konstanten

Gültigkeitsbereich

- bislang haben wir Variablen innerhalb von Blöcken { } verwendet oder als Parameter übergeben
- man kann ein und dieselbe Variable jedoch auch aus verschiedenen Blöcken heraus verändern
  - **Globale Variablen**

```
#include<stdio.h>
void zaehlen() {
    counter += 1;
}
int main() {
    int counter = 0;
    zaehlen();
    printf("Zaehler: %d\n", counter);
    return 0;
}
```

```
4: error: 'counter' undeclared (first use in this function)
```

Globale Variablen

- ist eine Variable global, so kann man von jedem Ort aus auf sie zugreifen

```
#include<stdio.h>

// globale Variable
int counter = 0;

void zaehlen() {
    counter += 1;
}

int main() {
    zaehlen();
    printf("Zaehler: %d\n", counter);
    return 0;
}
```

```
Zaehler: 1
```

Statische Variablen

- Normalerweise existieren Variablen nach dem Durchlauf des Blockes nicht mehr
- werden Variablen jedoch mit **`static`** gekennzeichnet, werden diese statisch und behalten Ihre Stellung

```
#include<stdio.h>
int zaehlen() {
    static int counter = 0;
    return ++counter;
}
int main() {
    printf("Zaehler: %d\n", zaehlen());
    printf("Zaehler: %d\n", zaehlen());
    printf("Zaehler: %d\n", zaehlen());
    return 0;
}
```

```
Zaehler: 1
Zaehler: 2
Zaehler: 3
```

Konstanten

- Eine konstante Variable ist eine Variable, dessen Wert nach der Initialisierung nicht mehr geändert werden kann.
- Kennzeichnung mit Schlüsselwort **`const`**

```
#include<stdio.h>

// globale konstante Variable
const float MWSTFAKTOR = 0.19;

int main() {
    float betrag = 350.0;
    float mwst = betrag * MWSTFAKTOR ;
    printf("%.f% von %.2f Euro: %.2f Euro\n",
             MWSTFAKTOR * 100, betrag, mwst );
    return 0;
}
```

```
19% von 350.00 Euro: 66.50 Euro
```

Konstante Zeiger bei Funktions-Parametern

- erhält eine Funktion einen Zeiger auf einen Wert, so kann sie diesen Wert ändern
- um sich gegen Veränderungen abzusichern, kann man die Parameter mit **`const`** deklarieren

```
#include<stdio.h>

void ausgabeMwSt(const float *geld) {
    float mwst = *geld * 0.19;
    printf("MwSt: %.2f Euro\n", mwst);
}

int main() {
    float betrag = 350.0;
    ausgabeMwSt(&betrag);
    return 0;
}
```

```
MwSt: 66.50 Euro
```

Symbolische Konstanten

- man erreicht das Gleiche wie mit einer konstanten Variable
- die Funktionsweise unterscheidet sich jedoch: *Textersetzung*
- Definition erfolgt im Präprozessor durch **`#define NAME WERT`**

```
#include<stdio.h>

// Symbolische Konstante
#define MWSTFAKTOR 0.19

int main() {
    float betrag = 350.0;
    float mwst = betrag * MWSTFAKTOR ;
    printf("%.2f%% von %.2f Euro: %.2f Euro\n",
              MWSTFAKTOR * 100, betrag, mwst );
    return 0;
}
```

```
19% von 350.00 Euro: 66.50 Euro
```

Speicherbelegung der Variablen

- Speicherklasse
  - Wo, wann, wie lange wird Speicher für Variablen belegt?
  - **`auto, static, register, extern`**
- Standard: **`auto`**
  - Speicherplatz wird beim Eintritt in eine Funktion/Block reserviert.
  - Speicherplatz wird beim Verlassen der Funktion/Block aufgegeben.
- **`register`** für **`auto`**-Variablen
  - Ablage in Prozessor-Registern, wenn genügend Registerplätze vorhanden sind.
  - Sie haben KEINE Hauptspeicheradresse → nicht über Zeiger erreichbar
  - nur für solche Variablen verwenden, auf die schnell zugegriffen werden muss, z. B. Zähler
  - spielt bei modernen Compilern keine Rolle mehr, da diese den Code ohnehin optimieren.

Speicherbelegung der Variablen

- `static` für statische Variablen
  - Speicherplatz für gesamte Programmlaufzeit reserviert
  - Die Variablen behalten beim Verlassen und Wiedereintritt in eine Funktion ihren Wert
  - außerhalb von Funktionen / Blöcken definierte Variablen sind immer **`static`** bzgl. dieser Funktionen / Blöcke
  - Auf Datei-Ebene **`static`** deklarierte Variablen sind nur innerhalb dieser Datei bekannt.

- **Hinweis:** Deklarationen innerhalb von Blöcken **{...}** überdecken Deklarationen desselben Bezeichners, die außerhalb dieser Blöcke gemacht werden.

Speicherbelegung der Variablen

❑ Externe Variablen: `extern`

- ❑ **extern** definiert eine globale Variable, die in allen Programm-Modulen sichtbar ist.
- ❑ Eine **extern** Variable kann nicht initialisiert werden, weil sie nur auf eine Variable verweist, die anderswo definiert wird.

```
#include <stdio.h>
#include <stdlib.h>
#include "test.h"

int x=10;

int main(int argc, char *argv[]){

    func();

    return 0;
}
```

```
// test.h
void func(void);

// test.c
void func(void){

    extern int x;
    printf("x = %d\n",x);
}
```

```
x = 10
```

Speicherbelegung der Variablen

- Speicherklassen, Gültigkeitsbereiche und Lebensdauer

Klasse	Gültigkeit	Lebensdauer
auto	Block	Block
register	Block	Block
extern	Programm	Programmlauf
static (blockintern)	Block	Programmlauf
static (außerhalb aller Blöcke)	Quelldatei (Modul)	Programmlauf

Aufg. 07.01 – statische Variablen

Deklariere eine Variable **`static int zaehler=0;`** außerhalb von **`main`**. Schreibe eine Funktion und deklariere dort eine Variable **`static int local1=10;`** und eine Variable **`int local2=10;`**

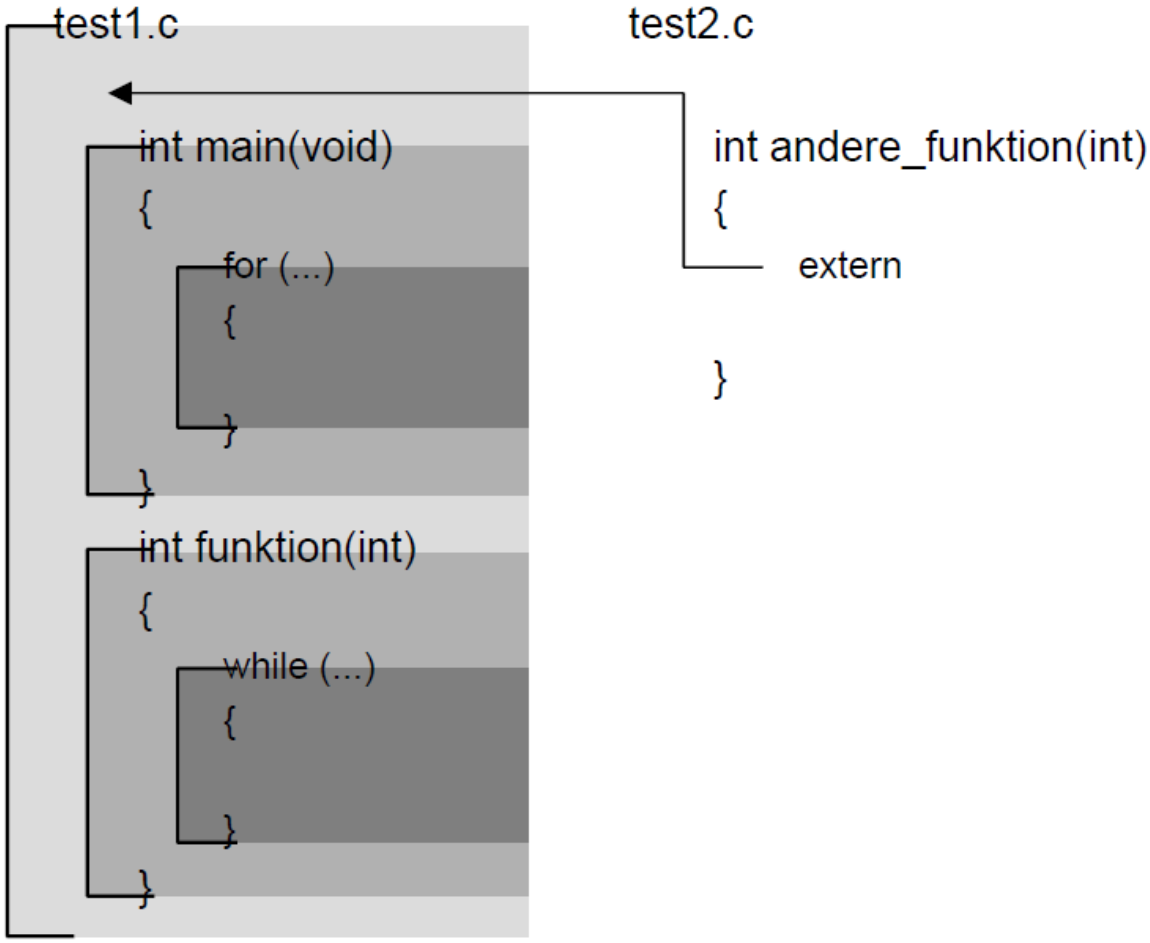
Dekrementiere beide Variablen innerhalb der Funktion und gib noch in der Funktion `zaehler, local1` und `local2` aus. Steuere in `main()` mit Hilfe von `zaehler`, dass die Funktion zehnmal aufgerufen wird.

- Interpretiere das Ergebnis!

Aufg. 07.01b

- Lagere nun die Funktion und ihre Deklaration in ein .h/.c-Dateienpaar aus. Deklariere **`zaehler`** einmal **`static`** und einmal nicht.
  - Code-Element in der Funktion aus der neuen .c-Datei :
    `extern int zaehler;`

Sichtbarkeit von Variablen



Aufg. 07.02 – Gültigkeitsbereich globaler Variablen

- Interpretiere die Ausgaben folgenden Programms:

```
#include <stdio.h>
#include <stdlib.h>


int iZaehlwert;

void myAusgabe(){
    printf("Wert von iZaehlwert (global): %d\n",iZaehlwert);
}

void myCounter(){
    printf("\nAnfangswert von iZaehlwert (global): %d\n",iZaehlwert);
    while (iZaehlwert++<5)
        myAusgabe();
}

int main(){
    int iZaehlwert;
    for (iZaehlwert=0;iZaehlwert>=-3;iZaehlwert--)
        printf("iZaehlwert in main(): %d\n",iZaehlwert);
    myCounter();

    printf("\niZaehlwert in main(): %d\n",iZaehlwert);

    return 0;
}
```

Gültigkeitsbereich - Zusammenfassung

- Das vorhergehende Beispiel belegt, dass Veränderungen an einer der beiden Variablen mit dem gleichen Namen **`iZaehlwert`** keinen Einfluss auf die jeweils andere haben.

- Ferner zeigt das Beispiel, dass der globale **iZaehlwert** einen Anfangswert von **0** hat, obwohl ihr dieser Wert nirgendwo explizit zugewiesen wurde. Das liegt daran, dass globale Variablen bei ihrer Definition generell mit dem Wert 0 initialisiert werden.

- **Hinweis:** Zu einem guten Programmierstiel gehört es ,die Initialisierung einer Variablen mit einem Anfangswert selbst vorzunehmen, d. h. mit der Zeile:

```
int iZaehlwert = 0;
```

  Der/Die Programmierer(in) sollte sich nicht „blind“ auf jeden C-Compiler verlassen – ggf. mal vorher testen!

Einsatz globaler Variablen –Risiken

- Die Verwendung globaler Variablen durchkreuzt das Konzept modularer (prozeduraler) Programmierung, das wesentlich auf die Abschottung von Daten gegenüber unerwünschten Zugriffen zielt.
- Globale Variablen sind für jede Funktion dienstbar und öffnen daher allen möglichen unerwünschten Nebeneffekten Tür und Tor.
- Gerade bei größeren Programmprojekten mit vielen Quelldateien stellen globale Variablen einen Unsicherheitsfaktor dar und sollten daher so weit wie möglich vermieden werden.
- Wenn du schon nicht darauf verzichten kannst (was wirklich selten der Fall ist), dann solltest du mit Hilfe des Schlüsselworts **static** dafür sorgen, dass eine globale Variable nur den Funktionen bekannt ist, die mit ihr zusammen in der gleichen Quelldatei definiert werden.
  - Beispiel dafür auf der nächste Folie

Sinnvolle Verwendung von `static`

```
#include <stdio.h>

static int iZaehlwert;

void func1(void);
void func2(void);

int main(void) {
. . .
```

C-Quelldatei

`iZaehlwert` befindet sich zusammen mit den Funktionen in einer Datei!

Die globale Variable **iZaehlwert** ist wegen **static** nur den Funktionen

`main(), func1(), func2()`

bekannt.
Werden in **anderen** Quelldateien weitere Funktionen definiert, können diese wegen des Schlüsselworts **`static`** nicht auf **`iZaehlwert`** in dieser Datei zugreifen.

Siehe auch Aufgabe 07.01b

Klausur SS19_Nachschreib Aufg. 3