Fachhochschule Jena
University of Applied Sciences Jena

Informatik I

Kapitel 3: Funktionen und Variablengültigkeit

Burkart Voss

- **Kleine _____, mit denen Teilprobleme einer größeren Aufgabe gelöst werden können.**
  - **Mit Funktionen lässt sich der Quellcode besser lesen.**
  - **Der Code kann durch Erstellen einer Funktionsbibliothek wieder verwertet werden.**
  - **Ständig sich wiederholende Routinen können in eine Funktion gepackt und müssen nicht immer wieder neu geschrieben werden.**
  - **Fehler und Veränderungen lassen sich daher auch schneller finden bzw. ausbessern, da der Code nur an einer Stelle bearbeitet werden muss.**

Definition von Funktionen

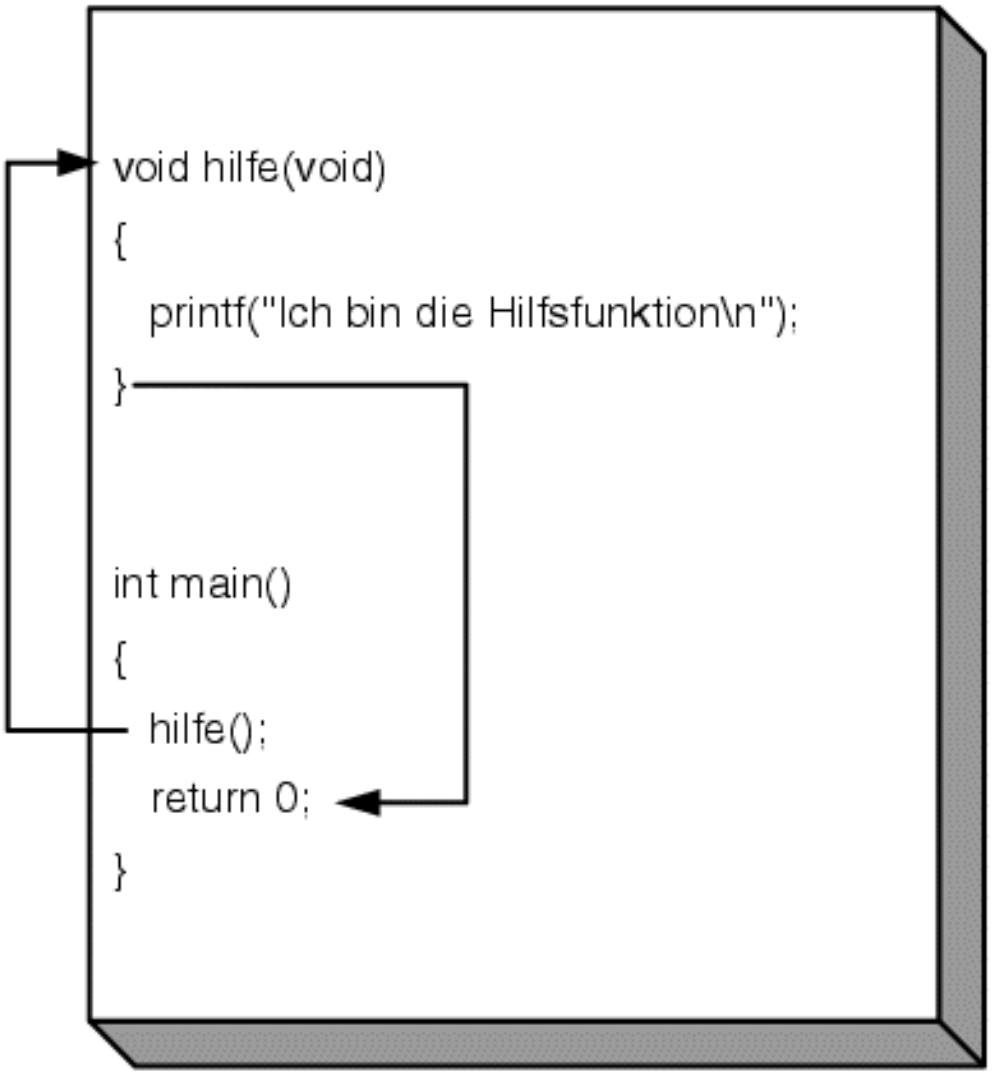

```
Rückgabetyp Funktionsname(Parameter)
{
    /* Anweisungsblock mit Anweisungen */
}
```

- **Rückgabetyp**
  - Datentyp des _____.
  - Alle bisher kennen gelernten Datentypen können verwendet werden.
  - Eine Funktion ohne Rückgabewert wird als void deklariert.
  - Wird kein Rückgabetyp angegeben, so wird automatisch eine Funktion mit Rückgabewert vom Datentyp int erzeugt.
- **Funktionsname**
  - Eindeutiger Name, mit dem die Funktion von einer anderen Stelle aus im Programmcode aufgerufen werden kann.
- **Parameter**
  - optional.
  - Werden durch _____ spezifiziert und durch ein Komma getrennt.
  - Wird kein Parameter verwendet, kann zwischen die Klammern entweder void oder gar nichts geschrieben werden.

```
#include <stdio.h>

void hilfe(void)
{
   printf("Ich bin die\
   Hilfsfunktion\n");
}

int main()
{
   hilfe();
   return 0;
}
```



- **Funktion muss vor ihrem _____ deklariert werden!**

```
#include <stdio.h>

void hilfe(void);

int main()
{
   hilfe();
   return 0;
}

void hilfe(void)
{
   printf("Ich bin die Hilfsfunktion\n");
}
```

Gültigkeit von Variablen


```
int x; /* x – global */

int f()
{
  int x, y;   /* x-lokal zu f */
              /* globales x wird
              verdeckt! */
  x = 1; y = 1;
  if (x == y) {
    int x;    /* x-lokal zu {} */
     x = 2;
  }           /* 1.lokales x wird
              bis '}' verdeckt! */
}
```

- Frage: Wo ist welches x gemeint???
- Jede neue Definition einer Variablen (in einer Funktion / in einem Block) erzeugt auch eine neue Speicheradresse, unter der ein Wert abgelegt werden kann
- Im gleichen Sichtbarkeitsbereich darf eine Variable jedoch nur einmal definiert werden!
- Dies gilt selbst in rekursiven Aufrufen (so sind z.B. alle 'n' bei der Fakultätsberechnung unterschiedlich!)

- Verwaltet den _____ für Funktionsaufrufe
  - Ist dynamisch (wächst bei Bedarf automatisch an bzw. schrumpft wieder)
  - In ihn werden alle Daten abgelegt, die zur Verwaltung von Funktionsaufrufen benötigt werden.
- Bei Funktionsaufruf wird Stack um einen Datenblock (auch Stack-Frame) erweitert. In ihn werden abgelegt:
  - _____
  - _____
  - _____ zur aufrufenden Funktion
- Datenblock bleibt so lange bestehen, bis die Funktion endet.
  - Wird in der Funktion eine weitere Funktion aufgerufen, wird ein weiterer Datenblock auf den aktuellen gepackt.
  - Ganz unten steht Block start-up Funktion, der main()-Funktion aufruft.
  - Der oberste Datenblock ist immer der aktuelle.

- **Lokale Variablen**
  - **Sind beschränkt _____ gültig**
- **Globale Variablen**
  - **Sind für alle Funktionen_____ gültig.**
- **Statische Variablen**
  - **Verlieren bei Beendigung ihres Bezugsrahmens _____**

  _____

Lokale Variablen

- Bei gleichnamigen Variablen ist immer die _____ gültig.
- Die lokalste Variable ist die, die dem _____ am nächsten steht.
- Lokale Variablen, die in einem Anweisungsblock definiert wurden, sind außerhalb dieses _____
- Funktionen sind wie Anweisungsblöcke
- Variablen so lokal wie möglich und so global wie nötig anlegen!!

- Statische Variablen verlieren bei Beendigung ihres Bezugsrahmens nicht ihren Wert.
- Achtung: Statische Variablen müssen schon bei ihrer Deklaration initialisiert werden!
- Schlüsselwort: `static`

Lebensdauer:

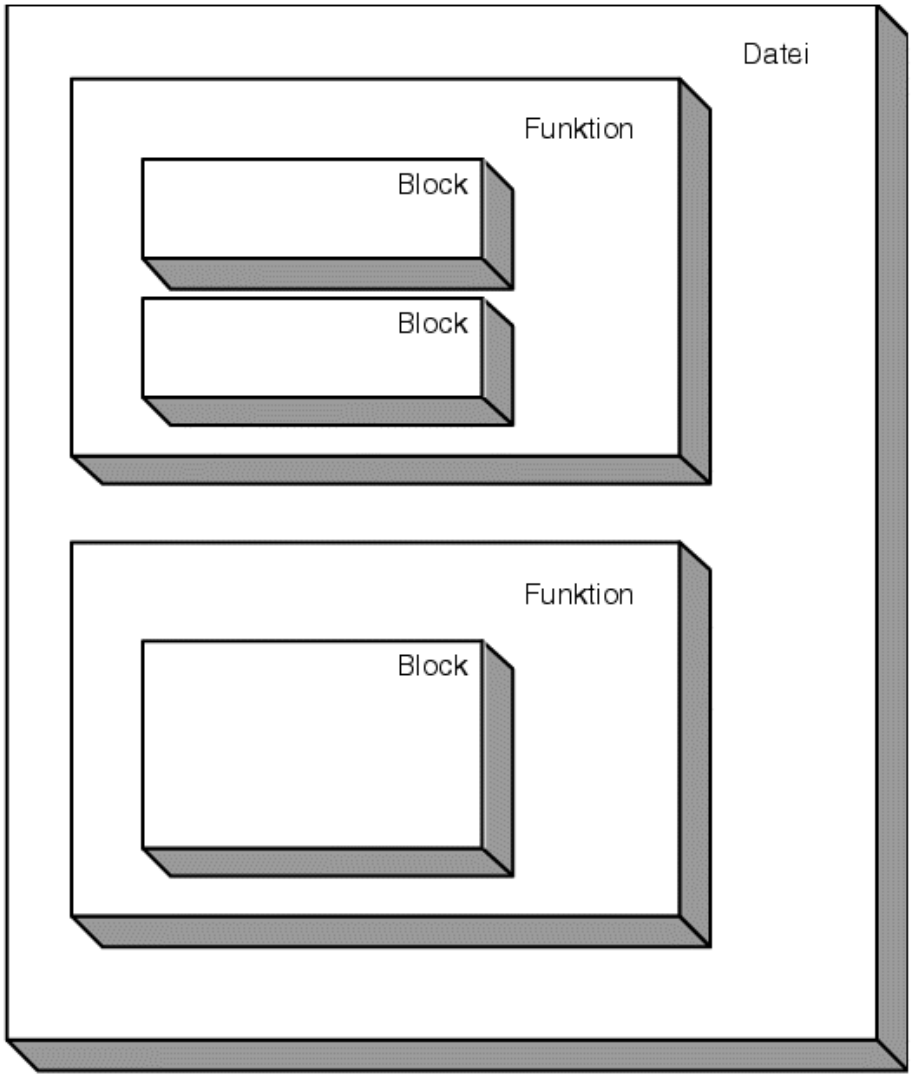
- `static`: Variable wird zu Programmbeginn einmal deklariert und existiert während der gesamten Programmausführung
- `auto`: Variable wird beim Eintritt in den Anweisungsblock, in dem sie definiert ist, neu erzeugt und beim Verlassen des Anweisungsblocks wieder gelöscht – Variable ist lokal.

Schlüsselworte für Variablen

- _____: Variable wird automatisch angelegt und wieder gelöscht
  - `int zahl=5;` ist identisch zu `auto int zahl=5;`
  - `auto` ist überflüssig.
- _____: Variable in anderen Dateien bekommen dieses Schlüsselwort.
- _____: Schlüsselwort für immer währende Variablen
- _____: Wert dieser Variablen wird vor jedem Zugriff neu aus dem Hauptspeicher eingelesen (wenn er sich von extern ändert)
- _____: Damit wird eine Konstante definiert:

```
const int wert=5;
wert +=5; /*Fehler*/
```

- **Geltungsbereich und Lebensdauer hängen von 2 Punkten ab:**
  - **Von Position in Deklaration der Variablen.**
    - **Block**
    - **Funktion**
    - **Datei**
  - **Von Speicherklassen-Spezifizierer, der vor der Variablen steht.**



Position	Speicherklasse	Lebensdauer	Geltungs-bereich
In einer Funktion	keine, auto, register	automatisch	Block
In einer Funktion	extern, static	statisch	Block
Außerhalb Funktion	keine, extern, static	statisch	Datei

Bestandteile einer Variablen

- Typ → _____
- Name → darüber wird auf den Wert der Variablen _____
- Wert → kann meistens variabel sein

- _____

- _____ der Speicherplatzzuordung (`static` oder `auto`)l
- Gültigkeitsdauer des Speicherplatzes der Variablen (bei auto Abhängig von Postion der Variablendeklaration)
- Welcher Programmteil kann auf Variable zugreifen (Abhängig von Position der Variablendeklaration und ob es gleichnamige Variable gibt).
- Wann wird Variable gespeichert/gelesen (`volatile`)

- **Globale Variablen (eine gemeinsame Variable für alle Funktionen)**
  - **Unübersichtlich bei größeren Programmen**
  - **Variablen so lokal wie möglich halten!**

- **Funktion mit Wertübergabe (Parameter)**
- **Funktion mit Wertrückgabe**

→ **Funktion kann als _____ wirken.**

Funktion mit Wertübergabe (call–by-value)

- **Parameter ist_____, mit dessen Argument die Funktion aufgerufen wurde.**
- **Schritte:**
  1. **Bei der Funktionsdefinition wird die Parameterliste festgelegt (formale Parameterliste).**
  2. **Die Funktion wird von einer anderen Funktion mit dem Argument aufgerufen (muss mit dem Typ des formalen Parameters übereinstimmen).**
  3. **Für die Funktion wird ein dynamischer Speicherbereich (im Stack) angelegt.**
  4. **Jetzt kann die Funktion mit den Parametern arbeiten.**

- **Call-by-value ermöglicht Ausdrücke als Aktualparameter zu verwenden**
  - **Dieser Ausdruck darf selbst wieder Funktionsaufrufe enthalten**
  - **Mechanismus: Ausdruck auswerten -Wert bestimmen -Formalparameter ersetzen**

```
void f(int a){
      printf("a = %d\n", a);
      /* ... */
}
int main(void){
      int x = 1;
      f(2*x+1); /*  Ausgabe: a = 3 */
      return 0;}
```

- **Wie können Daten von der Funktion an den Aufrufer zurückgegeben werden?**
  - Funktion hat _____
  - Wert wird mit `return wert;` zurückgegeben (mit return wird auch zu der aufrufenden Stelle zurückgesprungen!)
  - Typ von `wert` muß gleich sein zu oder konvertierbar sein in Typ der Funktion
  - `return` ohne `wert` nur erlaubt bei Funktionen ohne Rückgabewerte, d.h. vom Typ void.

```
int bignum(int a, int b)

{

   if(a > b)

      return a;

   else if(a < b)

      return b;

   else

      return 0; /* beide
Zahlen gleich groß */

}
```

- Eine Funktion in einem Programm muss den Namen
  `int main()`
  besitzen.
- Diese Funktion wird als erste beim Programmstart ausgeführt.
- Rückgabewert (return 0) dient dazu, daß der Startup-Code dem Betriebssystem mitteilt, ob das Programm ordnungsgemäß beendet wurde oder nicht.
- Bedeutung des Rückgabewertes ist Plattformabhängig
  - In <stdlib.h> ist Konstante EXIT_SUCCESS definiert, die genommen werden sollte.

- Sind jedem Compiler beigefügt
- Liegen als Object-Code vor (sind also schon kompiliert)
- Müssen dem Compliler bekannt gemacht werden
  - Z.B. über #include <stdio.h>
- Werden durch den dem Compiler nachgeschalteten Linker in das Programm eingebunden.

- **Wenn größere Programme in nur einer Datei sind:**
  - **Dauert das Compilieren länger**
  - **Wird der Quelltext unübersichtlicher**
  - **Wird Teamarbeit erschwert.**
- **Deshalb Funktionen in unterschiedliche Dateien**

_____!

Beispiel

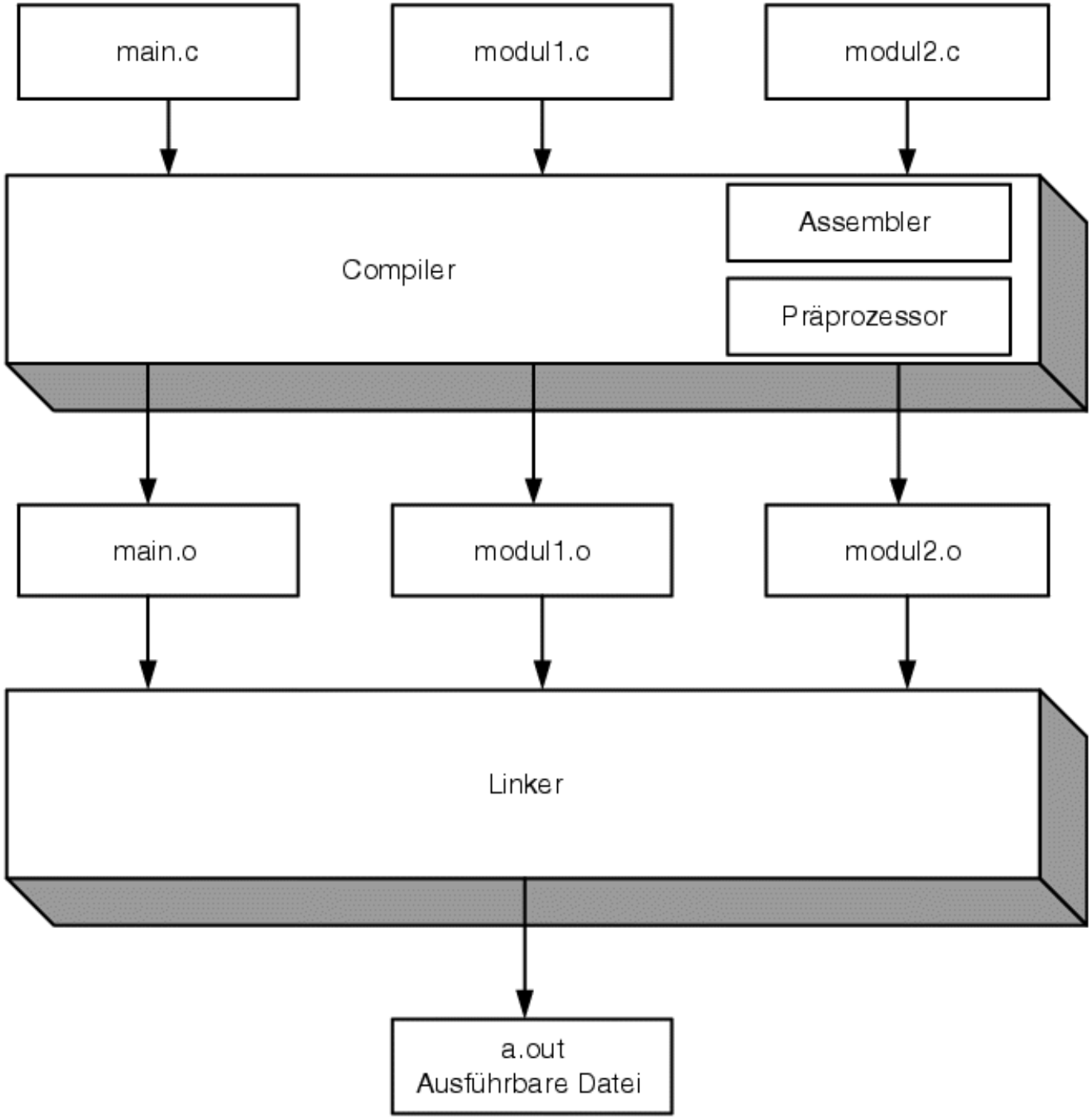

```
/*main.c*/
#include <stdio.h>
#include <stdlib.h>


extern void modul1(void);
extern void modul2(void);

int main()
{
    modul1();
    modul2();
    return EXIT_SUCCESS;
}
```

```
/*modul1.c*/
void modul1(void)
{
    printf("Ich bin das
   Modul 1\n");
}
```

```
/*modul2.c*/
void modul2(void)
{
    printf("Ich bin Modul
   2\n");
}
```


main.c
modul1.c
modul2.c
Compiler
Assembler
Präprozessor
main.o
modul1.o
modul2.o
Linker
a.out
Ausführbare Datei

Rekursive Funktionen

Rekursion ist eine Funktion, die sich selber aufruft.

- **Abbruchbedingung ist notwendig, damit sich Rekursion nicht unendlich oft selbst aufruft.**
- **Rücksprungadresse und Wert der Variablen (Datenblock/Stack-Frame) werden für jeden Aufruf im Stack gespeichert**
  → Stack kann „überlaufen“.

- **Bekanntes Beispiel: Fakultätsfunktion: Fakultät(1) := 1, Fakultät(n) := n * Fakultät(n-1)**

```
int fakultaet(int n)
{
if (n == 1)
return 1;
else
return n * fakultaet(n-1);
}
int main(void)
{
printf("Fakultaet(5) = %d\n", fakultaet(5));
return 0;
}
```