Hochschule **Rosenheim**
University of Applied Sciences

Prozedurale Programmierung

Funktionen

Hochschule Rosenheim - University of Applied Sciences

WS 2018/19

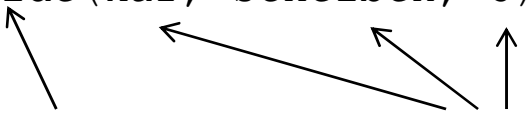
Prof. Dr. FJ. Schmitt

Rinderbeinscheiben in Rotweinsauce

Hochschule **Rosenheim**
University of Applied Sciences

```
herdplatte_schalter(5);        // Schalterstellung am Herd = 5


//definiert als schneide(t_gemuese, t_art, t_breite_mm)
schneide(kar, scheiben, 5);    // schneide Karotten in Scheiben, 5mm breit
```



Funktionsname Parameter unterschiedlichen Typs

⇒ Ziele
- Modularisierung des Programms
- Vermeidung von Codeduplizierung

Anmerkung:
Funktion hat hier keinen expliziten Rückgabewert
die Karotten werden geschnitten in der Variable „kar“ zurückgeliefert

Wozu Funktionen? (1)

- Beispiel:
  - Berechnung von b + sin(x)/x mit unterschiedlichen Werten:

```
a1 = b1 + sin(c1)/c1;
a2 = b2 + sin(c2)/c2;
a3 = b3 + sin(c3)/c3;
```

- Gute Lösung?
  - Codeduplizierung
  - Division durch Null!?

Wozu Funktionen? (2)

- Definition einer Funktion zur Berechnung von b + sin(x)/x

```
double f (double b, double x)
{
  double s;

  if (x == 0.0)
    s = 1;
  else
    s = sin(x)/x;

  return s + b;
}
```

Vorsicht mit Gleichheit bei Gleitpunktzahlen!

```
a1 = b1 + sin(c1)/c1;
a2 = b2 + sin(c2)/c2;
a3 = b3 + sin(c3)/c3;
```

```
a1 = f(b1,c1);
a2 = f(b2,c2);
a3 = f(b3,c3);
```

Wozu Funktionen? (3)

- Funktionen werden benötigt, um
  - immer wieder vorkommende Programmteile einsetzen zu können,
    - Änderungen sind dann nur an **einer einzigen** Stelle nötig
  - in sich abgeschlossene definierte Teilaufgaben zu programmieren,
  - das Programm zu strukturieren und
  - die Funktionalität zu erweitern.

Definition einer Funktion (1)

- Angabe eines eindeutigen Funktionsnamens, eines Rückgabewerts, der Parameter und des kompletten Funktionsrumpfs:

```
Typ Funktionsname (Parameterliste)
{
  Funktionsrumpf
}
```

- Merke: Definition ≠ Deklaration

Definition einer Funktion (2)

- Namen von Funktionen
  - Unterliegen denselben Einschränkungen wie Namen von Variablen und Konstanten
  - Müssen eindeutig sein, d.h. ist ein Name bereits für eine Variable oder Konstante vergeben, kann er nicht mehr benutzt werden
  - Konvention: beginnen mit Großbuchstaben

Definition einer Funktion (3)

- Parameter einer Funktion
  - beliebige Anzahl ist möglich
  - werden durch Komma getrennt in der Parameterliste innerhalb der runden Klammern nach dem Funktionsnamen angegeben
  - für jeden Parameter wird auch sein Typ angegeben

Definition einer Funktion (4)

```
#include <stdio.h>

void Summe(double x, double y)
{
   printf("Die Summe von %g und %g lautet: %g\n", x,y,x+y);
}

int main(void)
{
  double wert1, wert2;
  printf("Erste Zahl: ");
  scanf("%lf", &wert1);
  printf("Zweite Zahl: ");
  scanf("%lf", &wert2);
  Summe(wert1, wert2);
  return 0;
}
```

Definition einer Funktion (5)

Parameter einer Funktion

- Reihenfolge der Parameter ist für die Parameterübergabe (Funktionsaufruf) signifikant
- Hat eine Funktion keine Parameter so bleibt die Parameterliste leer

```
Ausgabe()
{
   printf("Hallo Welt!\n");
}
```

- Es kann (und sollte) hier aber der „Typ“ `void` verwendet werden (bedeutet soviel wie „nichts“)

```
Ausgabe(void)
...
```

Definition einer Funktion (6)

- Rückgabewert
  - Funktionen können einen Rückgabewert haben
  - muss unmittelbar vor dem Funktionsnamen angegeben werden

```
double Summe(double x, double y)
{
    ...
}
```

  - Funktion Summe liefert an den Aufrufer einen Wert vom Typ `double` zurück

Definition einer Funktion (7)

- Rückgabewert
  - Bei Weglassen wird automatisch der Typ `int` (!) angenommen
  - kein Rückgabewert ⇒ `void` angeben
  - man sollte also **immer** einen Rückgabewert angeben

```
Ausgabe()
{
   printf("Hallo Welt!\n");
}
```

```
void Ausgabe(void)
{
   printf("Hallo Welt!\n");
}
```

Definition einer Funktion (8)

- Rückgabewert
  - Der Wert selbst wird mit dem `return`-Befehl an den Aufrufer zurückgegeben
  - Beispiel: nächste Folie

Definition einer Funktion (9)

```
#include <stdio.h>
double Summe(double x, double y)
{
   double ergebnis;
   ergebnis = x+y;

   return ergebnis;
}

int main(void)
{
  double wert1, wert2, sum;
  printf("Erste Zahl: ");
  scanf("%lf", &wert1);
  printf("Zweite Zahl: ");
  scanf("%lf", &wert2);

  sum = Summe(wert1, wert2);
  printf("Die Summe von %g und %g lautet: %g\n", wert1, wert2, sum);
  return 0;
}
```

Typumwandlung

Achtung: Das ist schlechter Programmierstil!

- Stimmen die Typen bei einer Übergabe nicht überein, so wird eine Typkonversion durchgeführt

```
double Summe(long x, long y)   //3. Konvertierung der Argumente
                               //   auf long
{
   long ergebnis;
   ergebnis = x+y;
   return ergebnis;            //4. Ergebnis wird auf double
}                              //   konvertiert

int main(void)
{
  long sum;                    //1. Variable ist vom Typ long
  sum = Summe(1.2, 2.3);       //2. Argumente sind vom Typ double
                               //5. Konvertierung des Rückgabewerts
  return 0;                    //   auf long
}
```

Aufgabe

- Geben Sie für das eben betrachtete Beispiel an
  - welchen Wert die Parameter x und y in Summe() nach dem Aufruf von Summe(1.2, 2.3) haben
  - welchen Wert die Variable sum in main() hat
    - vor dem Aufruf von Summe()
    - nach dem Aufruf von Summe()

```
double Summe(long x, long y)   //3. Konvertierung der Argumente
                               //   auf long
{
   long ergebnis;
   ergebnis = x+y;
   return ergebnis;            //4. Ergebnis wird auf double
}                              //   konvertiert

int main(void)
{
  long sum;                   //1. Variable ist vom Typ long
  sum = Summe(1.2, 2.3);      //2. Argumente sind vom Typ double
                               //5. Konvertierung des Rückgabewerts
  return 0;                    //   auf long
}
```

Rückgabewert der Funktion `main`

- Die Funktion `main` liefert einen Rückgabewert vom Typ `int`

```
int main(void)
{
    //...
    return 0;
}
```

- Hauptprogramm signalisiert dem Aufrufer, ob ein Fehler aufgetreten ist
  - üblicherweise wird der Wert 0 zurückgegeben, wenn das Programm fehlerfrei abgearbeitet werden konnte

Ablauf eines Funktionsaufrufs (1)

- Prinzipiell gilt:
  - Bei einem Funktionsaufruf werden in C die Werte an die Funktionsparameter immer in Form von Kopien (call by value) und nicht im Original (call by reference) übergeben

```
long Erhoehe(long a)
{
   a = a+1;
   return a;
}
int main(void)
{
   long y, x=1;
   y = Erhoehe(x);
   return 0;
}
```

Ablauf eines Funktionsaufrufs (2)

```
long Erhoehe(long a)
{
   a = a+1;
   return a;
}
int main(void)
{
   long y, x=1;
   y = Erhoehe(x);  // x bleibt unverändert!
   return 0;
}
```

- Bei Funktionsaufruf:
  - Variable a wird erschaffen und mit 1 initialisiert
  - a (die Kopie von x) wird erhöht und 2 zurückgegeben
  - x enthält nach wie vor 1

Sichtbarkeit von Variablen (1)

- Variablennamen, die innerhalb von Funktionen definiert wurden, gelten nur innerhalb dieser Funktion

```
#include <stdio.h>
double Summe(double x, double y)
{  double ergebnis;

   ergebnis = x+y;  // x und y des Hauptprogramms sind hier unsichtbar
   return ergebnis;
}
int main(void)
{
  double x, y, ergebnis;   // nur innerhalb von main sichtbar
  printf("Erste Zahl: ");
  scanf("%lf", &x);
  printf("Zweite Zahl: ");
  scanf("%lf", &y);

  ergebnis = Summe(x, y);
  printf("Die Summe von %g und %g lautet: %g\n", x, y, ergebnis);
  return 0;
}
```

Sichtbarkeit von Variablen (2)

- innerhalb der Funktion `main`
  - Nur die lokalen Variablen `x`, `y` und `ergebnis` sind sichtbar
  - Gleichlautende Variablen der Funktion Summe sind nicht sichtbar (existieren noch nicht)

- bei Funktionsaufruf von `Summe`:
  - Gleichlautende Parameter werden erzeugt und die Werte aus den Argumenten des Funktionsaufrufs hineinkopiert

- innerhalb der Funktion `Summe`
  - Variablen der Funktion `main` sind unsichtbar

Sichtbarkeit von Variablen (3)

- Nach Abarbeitung der Funktion Summe
  - Rücksprung zur `main`-Funktion
  - Ergebnis, das in der lokalen Variablen `ergebnis` gespeichert ist, wird in die Variable `ergebnis` der Funktion `main` kopiert
  - alle lokalen Variablen der Funktion `Summe` werden beim Verlassen der Funktion zerstört

Deklaration einer Funktion (1)

- Deklaration = Bekanntmachung einer Funktion in einer C-Datei
  - Anschreiben des Funktionskopfs (Prototyp)

```
void Ausgabe(void);

int main(void)
{
   Ausgabe();
}

void Ausgabe(void)
{
   printf("Hallo Welt!\n");
}
```

Deklaration einer Funktion (2)

- Funktionen müssen dem Compiler bekannt gemacht werden, wenn
  - sie erst später/weiter unten in der C-Datei definiert werden oder
  - sie in einem anderen Modul (C-Datei) definiert werden.

- Compiler wird mitgeteilt:
  - Funktionsname
  - Anzahl und Typen der Argumente
  - Typ des Rückgabewerts

- üblicherweise werden Funktionsdeklarationen in eigenen Header-Dateien abgelegt

Definition versus Deklaration

- Deklaration einer Funktion
  - ist eine Bekanntmachung einer Funktion an den Compiler
  - Funktion wird als Black Box von außen betrachtet
- Definition einer Funktion
  - ist die komplette Beschreibung und enthält auch den Funktionsrumpf
  - Betrachtung von innen durch Beschreibung der Implementierung von Algorithmen

```
void Ausgabe(void);

int main(void)
{
   Ausgabe();
}

void Ausgabe(void)
{
   printf("Hallo Welt!\n");
}
```

Funktionsdeklaration

Variablenbezeichner dürfen in der **Deklaration** weggelassen werden

```
double Summe(double x, double y);

int main(void)
{
   double x, y, e;
   e = Summe(x,y);
}

double Summe(double x, double y)
{
   double ergebnis;
   ergebnis = x + y;
   return ergebnis;
}
```

```
double Summe(double, double);

int main(void)
{
   double x, y, e;
   e = Summe(x,y);
}

double Summe(double x, double y)
{
   double ergebnis;
   ergebnis = x + y;
   return ergebnis;
}
```

Externe Funktionen

- Welche Schritte sind notwendig damit eine Funktion auch in einem anderem Modul (Quelltext-Datei) verwendet werden kann?
  1. Definition der Funktion
     - Alle Funktionen sind automatisch extern und können damit in andere Module exportiert werden
  2. Deklaration der Funktion in dem Modul, in dem sie verwendet werden soll
     - wird erreicht durch:
       Funktionsdeklarationen in einer eigenen Header-Datei zusammenfassen und diese zu Beginn jeder C-Datei inkludieren

Statische Funktionen

- Soll eine Funktion nur in einem Modul sichtbar sein, so muss das Schlüsselwort `static` zu Beginn des Funktionskopfs angeschrieben werden

```
static void StatischeFunktion()
{
   //...
}
```

  - können nicht exportiert werden
  - müssen innerhalb der selben Datei ebenfalls deklariert werden, wenn sie vorher verwendet werden

```
static void StatischeFunktion();
```

Zusammenfassung

- Definition und Deklaration von Funktionen
  - Parameter
  - Rückgabewert
- Ablauf eines Funktionsaufrufs
- Sichtbarkeit von Variablen