Kapitel 6
Funktionen

Kapitel 6 – Funktionen

6.1. Einleitung

Funktionen werden verwendet, um

- Teilaufgaben zu bearbeiten, die häufig vorkommen,
- große Programme in kleine, übersichtliche Teile zu unterteilen.

C-Programme bestehen mindestens aus einer einzelnen Funktion, der Funktion **main**. Mit dieser Funktion beginnt die Ausführung des Programms.

Beispiel: Berechne $y = x^n$ für vorgegebene x und n

```
y = 1.0
for (i = 1; i <= n; i++)
{
    y = y * x;
}
```

Sollte diese Potenz öfters berechnet werden, müsste dieser Programmteil ständig wiederholt werden!

Auslagerung der Teilaufgabe „Potenz berechnen“ in eine Funktion:

- Eingabe: Werte von Basis (x) und Exponent (n)
- Ausgabe: Wert der n-ten Potenz

```c
#include <stdio.h>
float power(float x, int k);

int main(void)
{
   float x1, x2, z, t;
   int n = 4;
   x1 = 3.0;
   z = power(x1, n);
   x2 = 3.75;
   t = power(x2, 5);
   return 0;
}
```

```c
float power(float x, int k)
{
   int i;
   float y;
   y = 1.0;
   for (i = 1; i <= k; i++)
   {
      y = y * x;
   }
   return y;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Datenaustausch zwischen Hauptprogramm und Funktion:

main :

```
x1 = 3.0;
n = 4;
z = power(x1, n);

...

t = power(3.75, 5);
```

Input 3.0 4

Output : 81.0

3.75 5

741.58

power :

„Black Box"

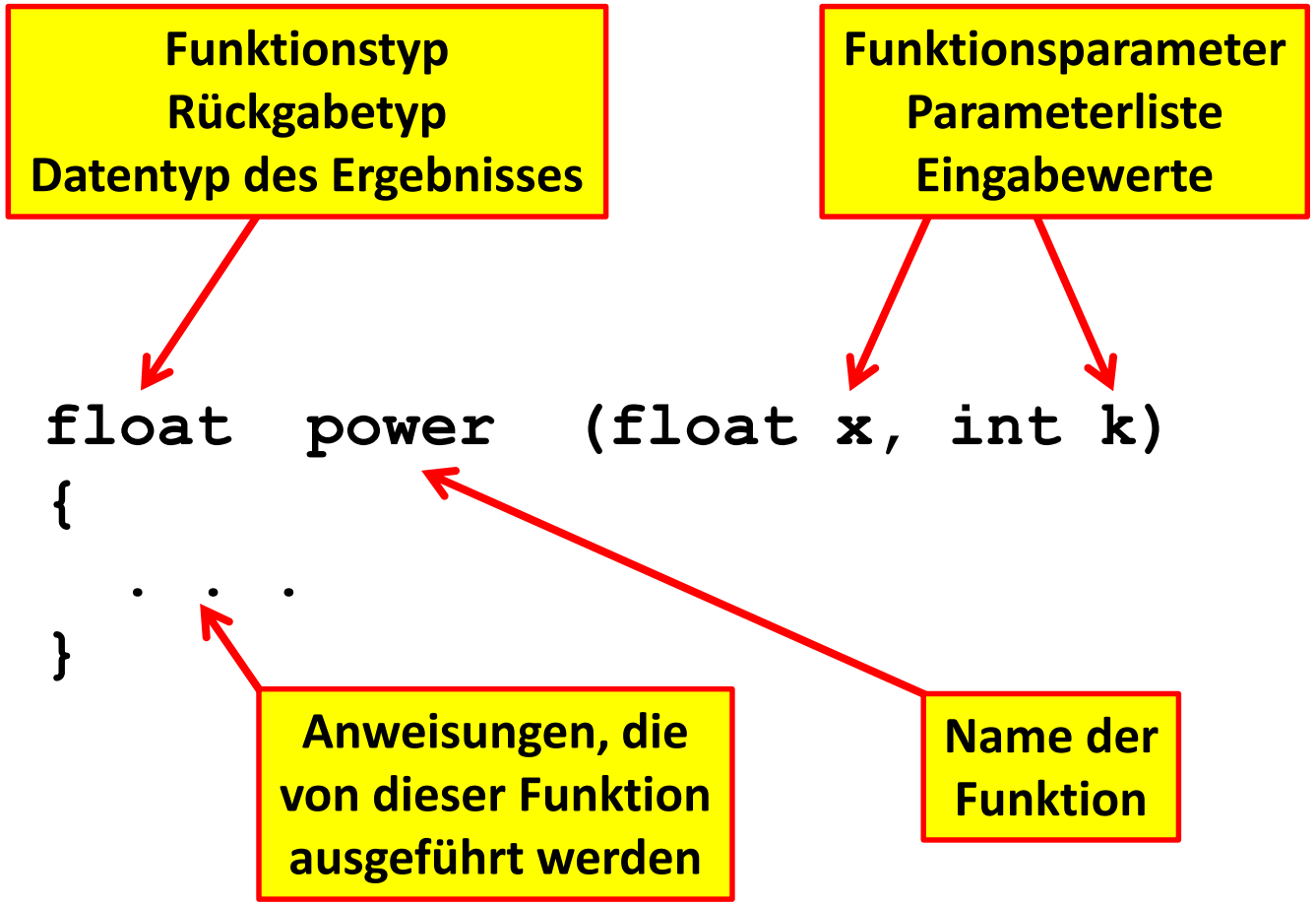
Kapitel 6 – Funktionen

Beim Start des Programms passiert Folgendes:

- Es wird immer zuerst die Funktion main aufgerufen.
- Die Zeilen in main werden der Reihe nach abgearbeitet.
- In der Zeile `z = power(x1,n);` wird die Funktion main verlassen. Es wird Speicherplatz für die **lokalen Variablen** x und k der Funktion power bereitgestellt; dann werden die **Werte kopiert** von x1 nach x bzw. von n nach k.
- Die Funktion power wird abgearbeitet.
- Bei der Anweisung `return y;` wird die Funktion power verlassen und der Wert von y wird der linken Seite beim Funktionsaufruf (hier z) zugewiesen.
- Schließlich wird die Funktion main fortgesetzt.

Kapitel 6 – Funktionen

Aufbau einer Funktionsdefinition:



6.3. Aufbau einer Funktion

Funktionsname:

- Es gelten die gleichen Regeln wie für Variablennamen.
- *Der Name einer Funktion sollte etwas mit der Aufgabe zu tun haben, die von der Funktion erledigt wird!*

Funktionsparameter:

- Funktionsparameter sind diejenigen Werte, die die Funktion von außen (vom aufrufenden Programm) benötigt.
- Datentypen, Namen, Reihenfolge festlegen – **formale Parameter.**
- `z = power(x1,n);` – x1 und n heißen **aktuelle Parameter.**
- Erst beim Funktionsaufruf (während der Laufzeit des Programms) wird Speicherplatz für die Parameter reserviert.
- Parameter dürfen nur innerhalb der Funktion verwendet werden (sog. **lokale Variablen**).
- **void** („nichts, leer“), falls keine Aufrufparameter vorhanden sind.

6.3. Aufbau einer Funktion

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Funktionstyp, Rückgabewert der Funktion:

- Funktionstyp und Datentyp des Rückgabewerts, der in der **return-Anweisung** zurückgegeben wird, müssen übereinstimmen.
- Alle Datentypen sind erlaubt.
- Funktionstyp **void** wird für Funktionen ohne Rückgabewert verwendet.

Rücksprung, return-Anweisung:

- Mit der return-Anweisung wird die Funktion verlassen – es erfolgt ein Rücksprung zum aufrufenden Programm.
- Beispiele: `return y;` `return y*y + 3.16;`
- Mehrere return-Anweisungen in einer Funktion möglich.
- Rücksprung erfolgt ebenfalls, wenn das Funktionsende erreicht wird (nur sinnvoll, falls Funktionstyp void verwendet wird).
- In der Funktion main bewirkt die return-Anweisung das Programmende; der Wert kann vom Betriebssystem ausgewertet werden.

6.3. Aufbau einer Funktion

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Deklaration:

- Jede Funktion muss vor ihrer Verwendung **deklariert** werden.
- Der sog. **Funktionsprototyp** legt die Schnittstelle zum Aufruf der Funktion fest (Name der Funktion; Datentypen, Anzahl und Reihenfolge der Übergabeparameter).
- Die Funktionsdeklaration (bzw. der Funktionsprototyp) sieht aus wie die erste Zeile der Funktionsdefinition – sie wird allerdings mit einem Semikolon abgeschlossen.
- Beispiel: `float power(float x, int k);`

Funktion main:

- Die Funktion main wird beim Start des Programms automatisch aufgerufen.
- Gültige Varianten (lt. C-Standard):

```
int main (void)
int main (int argc, char *argv[])
```

Kapitel 6 – Funktionen

Beispiel 1: Minimum von 2 ganzen Zahlen bestimmen

- Aufrufparameter: zwei Integerzahlen
- Rückgabewert: Integerzahl
- Funktionsprototyp: `int imin(int a, int b);`

```
#include <stdio.h>
int imin(int a, int b);
int main(void)
{
    int i, j, k;
    i = 3;
    j = 7;
    k = imin(i, j);
    printf("Min = %d\n", k);
    k = imin(i, 234);
    printf("Min = %d\n", k);
    return 0;
}
```

Deklaration von imin (Funktionsprototyp)

Aufruf der Funktion imin

Beispiel 2: Minimum von zwei Fließkommazahlen

- Aufrufparameter: zwei Fließkommazahlen (float oder double)
- Rückgabewert: Fließkommazahl
- Funktionsprototyp: `float imin(float a, float b);`
  oder: `double imin(double a, double b);`

Beispiel 3: Minimum von 2 ganzen Zahlen, kein Rückgabewert

- Die Funktion soll das Minimum ermitteln und auch ausgeben
- Aufrufparameter: zwei Integerzahlen
- Rückgabewert: void
- Funktionsprototyp: `void imin(int a, int b);`

Kapitel 6 – Funktionen

6.5. Taschenrechner

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:
Es soll ein einfacher Taschenrechner mit zwei Funktionen erstellt werden. Der Taschenrechner kann zwei Integerzahlen multiplizieren und eine Zahl, die dezimal eingegeben wurde, in hexadezimaler Form wieder ausgeben. Schreiben Sie ein C-Programm, das folgende Aufgaben erfüllt: Der Anwender wird aufgefordert eine Taschenrechnerfunktion auszuwählen. Nach der Ausführung dieser Funktion kann eine weitere Funktion ausgewählt oder das Programm beendet werden.

```
Einfacher Taschenrechner
(0) Beenden
(1) Multiplikation
(2) Hexadezimalzahl
Bitte wählen Sie: _
```

```
Multiplikation
Erste Zahl: _
Zweite Zahl: _
Produkt _ * _ = _
```

```
Hexadezimalzahl
Geben Sie eine Dezimalzahl ein: _
Dezimal _ entspricht _ Hexadezimal
```

Strukturierte Programmierung

- Schrittweise Verfeinerung
- Größeres Problem in kleinere Teilprobleme zerlegen
- Teilprobleme gegebenenfalls wieder zerlegen
- Funktionen helfen, größere Probleme zu zerlegen; jede Funktion bearbeitet ein definiertes, einfaches Teilproblem
- Beschränkung auf 3 einfache Kontrollstrukturen:

1)	**Folge**	`anweisung1;` `anweisung2;`
2)	**Alternative**	`if else` `switch`
3)	**Wiederholung**	`while` `do while` `for`

Teilaufgaben

1) Eingabeaufforderung anzeigen, Funktion auswählen (1 oder 2)
   Eingegebene Zahl prüfen, gegebenenfalls Eingabe wiederholen
   → Funktion: **auswahl**
2) Zwei Zahlen einlesen
   Multiplikation durchführen, Produkt ausgeben
   → Funktion: **multiplikation**
3) Aufforderung zur Eingabe einer Zahl
   Dezimalzahl einlesen, hexadezimal ausgeben
   → Funktion: **hexdarstellung**
4) Hauptprogramm, Ablaufsteuerung
   Funktion auswahl aufrufen, gibt Integerwert zurück
   Gewählte Funktion ausführen, ggf. Programm beenden
   auswahl erneut aufrufen
   → Funktion: **main**

Hauptprogramm des Taschenrechners:

```
#include <stdio.h>

int auswahl(void);   /* Funktionsdeklarationen */
void hexdarstellung(void);
void multiplikation(void);

int main(void)   /* Hauptprogramm */
{
     int wahl;
     do
     {
          wahl = auswahl();
          switch(wahl)
          {
          case 1: multiplikation(); break;
          case 2: hexdarstellung(); break;
          }
     }
     while(wahl != 0);   /* Beenden durch Eingabe von 0 */
     return 0;
}
```

Kapitel 6 – Funktionen

6.6. Standardfunktionen

Beim Programmieren gibt es eine Reihe von Standardaufgaben, die immer wieder auftreten:

- Berechnung der Quadratwurzel
- n-te Potenz einer Zahl berechnen
- trigonometrische Funktionen
- Ein- und Ausgabe

Hierfür existieren bereits fertige **Standardfunktionen** – normiert im ANSI-Standard. Sie sind in praktisch allen Programmierumgebungen vorhanden.

Auf den folgenden Folien werden verschiedene Gruppen von Standardfunktionen vorgestellt (viele weitere Beispiele finden sich im Internet oder in der Online-Hilfe).

Mathematische Standardfunktionen: `#include<math.h>`

Quadratwurzel $y = \sqrt{x}$	`double sqrt(double x)`
Potenzfunktion x^y	`double pow(double x,double y)`
Exponentialfunktion	`double exp(double x)`
Natürlicher Logarithmus	`double log(double x)`
Logarithmus zur Basis 10	`double log10(double x)`
Absoluter Betrag	`double fabs(double x)`
Sinus	`double sin(double x)`
Kosinus	`double cos(double x)`
Tangens	`double tan(double x)`
Arcus Sinus	`double asin(double x)`
Arcus Kosinus	`double acos(double x)`
Arcus Tangens	`double atan(double x)`
Sinus Hyperbolicus	`double sinh(double x)`
Kosinus Hyperbolicus	`double cosh(double x)`
Tangens Hyperbolicus	`double tanh(double x)`

Ausschnitt aus der Header-Datei math.h:

```
/* math.h -- Definitions for the
   math floating point package.  */
double cos(double x);
double sin(double x);
double tan(double x);
double tanh(double x);
double fabs(double x);
double acos(double x);
double asin(double x);
double cosh(double x);
double sinh(double x);
double exp(double x);
double pow(double base, double power);
double sqrt(double x);
#define M_E   2.7182818284590452354
#define M_PI  3.14159265358979323846
. . .
```

Verarbeitung von Zeichen: `#include<ctype.h>`

`int isalnum(int c);`	Ist c ein Buchstabe oder eine Ziffer
`int isalpha(int c);`	Ist c ein Buchstabe?
`int iscntrl(int c);`	Ist c ein Steuerzeichen?
`int isdigit(int c);`	Ist c eine Ziffer?
`int isgraph(int c);`	Ist c ein druckbares Zeichen (außer Leerzeichen)?
`int islower(int c);`	Ist c ein Kleinbuchstabe?
`int tolower(int c);`	Umwandlung von c in einen Kleinbuchstaben
`int toupper(int c);`	Umwandlung von c in einen Großbuchstaben
`int isprint(int c);`	Ist c ein druckbares Zeichen?
`int ispunct(int c);`	Ist c ein Interpunktionszeichen?
`int isspace(int c);`	Ist c ein Leerzeichen/Tabulatur/Zeilenumbruch?
`int isupper(int c);`	Ist c ein Großbuchstabe?
`int isxdigit(int c);`	Ist c eine Hexadezimalziffer?

Ein- und Ausgabe: `#include<stdio.h>`

- `printf, scanf` – Ausgabe auf Bildschirm
- `fprintf, fscanf` – Ein-/Ausgabe mit Dateien
- `getchar` – Einzelnes Zeichen einlesen

„Standard-Library“, Verschiedenes: `#include<stdlib.h>`

- `rand` – Integer-Zufallszahl zwischen 0 und RAND_MAX
- `abs, labs` – Absoluter Betrag von Integer- und Long-Variablen (vergl. `fabs` aus math.h für Double-Variablen)

Verarbeitung von Datum, Uhrzeit: `#include<time.h>`

Arbeit mit Zeichenketten: `#include<string.h>`

Kapitel 6 – Funktionen

6.7. Lokale und globale Variablen

Was passiert, wenn wir den Funktionsparametern und der Variablen in der Funktion **imin** die Namen i, k und j geben (Achtung: In der Funktion **main** gibt es ebenfalls Variablen i, j und k…)?

Ist das erlaubt? (→ **Ja!**) Belegen die Variablen die gleichen Speicherzellen, kommt es zu „Überschneidungen“? (→ **Nein!**)

```c
#include <stdio.h>
int imin(int i, int k);
int main(void)
{
   int i, j, k;
   i = 3; j = 7;
   k = imin(i, j);
   printf("Min = %d\n", k);
   k = imin(i, 234);
   printf("Min = %d\n", k);
   return 0;
}
```

```c
int imin(int i, int k)
{
   int j;
   if (i < k)
      j = i;
   else
      j = k;
   return j;
}
```

Lokale Variablen:

- Lokale Variablen werden im Kopf einer Funktion (wie die Variablen i und k bei der Funktion imin) oder innerhalb einer Funktion definiert (wie die Variable j in der Funktion imin oder die Variablen i, j, k und n in der Funktion main).
- Sie dürfen bzw. können nicht von anderen Funktionen verwendet werden.
- Nur während eine Funktion bearbeitet wird, ist Speicherplatz für die lokalen Variablen der Funktion reserviert; nach Beendigung der Funktion gibt es diese Variablen nicht mehr.
- Lokale Variablen, die den gleichen Namen besitzen, aber in unterschiedlichen Funktionen definiert sind, haben nichts miteinander zu tun (wie die zwei Variablen j in den Funktionen main und imin); sie haben nur zufällig den gleichen Namen.

6.7. Lokale und globale Variablen


```
#include <stdio.h>
void imin(int a, int b);
int k;
int main(void)
{
   int i = 3, j = 7;
   imin(i, j);
   printf("Min = %d\n", k);
   k = 17; imin(k, 234);
   printf("Min = %d\n", k);
   return 0;
}
void imin(int a, int b)
{
   if (a < b)
      k = a;
   else
      k = b;
}
```

Beispiel für eine globale Variable: k ist sowohl in der Funktion main als auch in imin sichtbar!

Welche Ausgabe liefert dieses Programm?

Globale Variablen:

- Globale Variablen werden außerhalb von Funktionen, meist am Programmanfang definiert.
- Sie existieren während der gesamten Programmlaufzeit, d. h. es sind während der gesamten Programmlaufzeit Speicherzellen für die globalen Variablen reserviert.
- Sie dürfen von allen Funktionen verwendet werden, außer es gibt innerhalb einer Funktion eine lokale Variable mit dem gleichen Namen – lokale Variable hat Vorrang; in diesem Fall kann die Funktion nicht auf die globale Variable zugreifen (die globale Variable ist „unsichtbar“).
- Typische Anwendung globaler Variablen: Informationsaustausch zwischen Funktionen; speziell zur **gleichzeitigen Rückgabe mehrerer Werte** aus einer Funktion.

Kapitel 6 – Funktionen

Aufgabe:

Erstellen Sie ein Programm zur Lösung quadratischer Gleichungen mit der pq-Formel. Lösen Sie die quadratische Gleichung in einer Funktion **`int qsolve(double p, double q)`** und geben Sie beide Nullstellen in den globalen Variablen x1 und x2 zurück.

Die Funktion qsolve gibt in ihrem Integer-Rückgabewert eine 1 zurück, falls die Gleichung gelöst werden konnte, andernfalls eine 0.

$$x^2 + px + q = 0$$

$$\Rightarrow \quad x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

Hauptprogramm mit Aufruf der Funktion qsolve:

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

double x1, x2;
int qsolve(double p, double q);

int main(void)
{
    double p, q;
    printf("   2                    \n");
    printf("  x  + px + q = 0\n\n");
    printf("p eingeben: "); scanf("%lf", &p);
    printf("q eingeben: "); scanf("%lf", &q);

    if(qsolve(p, q) != 0)
        printf("Nullstellen: %f, %f\n", x1, x2);
    else
        printf("Keine Nullstellen!\n");
    return 0;
}
```