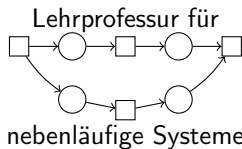


Vorkurs Informatik

Funktionen

Robert Lorenz

Tag 5



Bereits bekannte Funktionen

- Bibliotheksfunktionen: `printf`, `putchar`, `rand`, `srand`, `isdigit`, `toupper`, `sqrt`, `sin`, ...
- `main`-Funktion

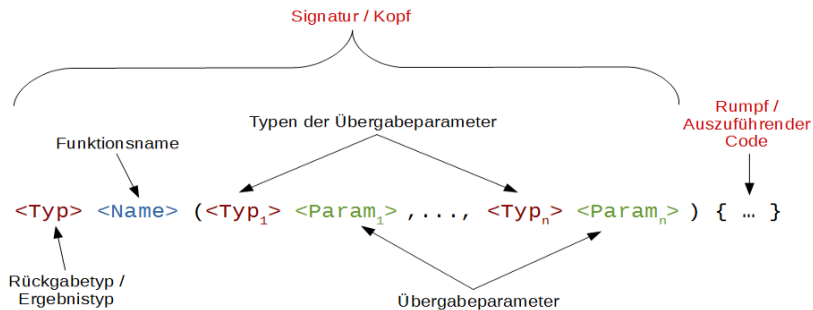
Lernziel

Eigene Funktionen programmieren und nutzen.

- Code, den man unter Umständen **öfter in einem Programm braucht**, kapselt man in eine Funktion ein.
- Dieser Code lässt sich dann über die Funktion aufrufen, **ohne dass man ihn nochmal wiederholt aufschreiben muss**
- Will man diesen Code **ändern, verbessern oder erweitern**, so muss man das **nur an einer Stelle** tun
- *Je konsequenter man sich wiederholende Bestandteile des Codes in Funktionen kapselt, umso besser wartbar und erweiterbar ist also das Programm, und um so weniger fehleranfällig*

Allgemeiner Aufbau einer Funktion

Wiederholung



Es gibt auch Funktionen ohne Parameter ($n=0$), dann schreibt man

`<Typ> <Name> (void) { ... }.`

Und ohne Rückgabewert:

`void <Name> (...) { ... }.`

Den **Funktions-Kopf** nennt man auch **Funktions-Signatur** oder **-Prototyp**.

Rückgabetypen, Parameter-Typen

Jede Programmiersprache stellt eine Reihe von Typen zur Verfügung; wir haben bisher kennengelernt: `int`, `char`, `bool`, `double`

Funktionsname, Parameter-Namen

Namen von Funktionen und Parametern dürfen in C aus lateinischen Buchstaben und dem `_`-Zeichen gebildet werden. Außerdem sollten Konventionen der Namensbildung wie bei Variablen beachtet werden!

isdigit-Funktion

```
int isdigit(int c)
```

int	Typ des Rückgabewerts
isdigit	Funktionsname
c	Eingabeparameter vom Typ int

Liefert einen Wert ungleich Null, wenn c der ASCII-Code einer Ziffer ist, sonst Null.

printf-Funktion

```
int printf(const char * format,...)
```

int	Typ des Rückgabewerts
printf	Funktionsname
format	Eingabeparameter vom Typ const char * (Datentyp für konstante Zeichenketten - siehe Informatik 1)
...	Es kann weitere Eingabeparameter geben

- Gibt format auf Kommandozeile aus, wobei Umwandlungsangaben in format (wir kennen bisher: %i, %c, %f, %e) durch die Werte der danach übergebenen Ausdrücke ersetzt werden.
- Liefert die Anzahl der ausgegebenen Zeichen oder einen negativen Wert bei einem Fehler.

sqrt-Funktion

```
double sqrt(double x)
```

double	Typ des Rückgabewerts
sqrt	Funktionsname
x	Eingabeparameter vom Typ double

Liefert die Quadratwurzel für nicht-negative x.

rand-Funktion

```
int rand()
```

int

Typ des Rückgabewerts

rand

Funktionsname

keine Eingabeparameter

Liefert eine ganze Pseudozufallszahl.

putchar-Funktion

```
int putchar(int c)
```

int	Typ des Rückgabewerts
putchar	Funktionsname
c	Eingabeparameter vom Typ int

- Gibt das ASCII-Zeichen zum ASCII-Code c auf Kommandozeile aus.
- Liefert c oder EOF bei einem Fehler.
- EOF ist eine ganzzahlige symbolische Konstante.

main-Funktion

```
int main(void)
```

int	Typ des Rückgabewerts
main	Funktionsname
void	hat keine Eingabeparameter

- Wird bei Programmstart ausgeführt und enthält die Anweisungen des Programms.
- Liefert 0 im Erfolgsfall und sonst einen Wert ungleich Null.

main-Funktion (Erweiterte Version)

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

int Typ des Rückgabewerts

main Funktionsname

argc Eingabeparameter vom Typ int
(Anzahl der bei Aufruf übergebenen Parameter)

argv Eingabeparameter vom Typ char * []
(Liste der bei Aufruf übergebenen Parameter)

- Wird bei Programmstart ausgeführt und enthält die Anweisungen des Programms.
- Liefert 0 im Erfolgsfall und sonst einen Wert ungleich Null.

- Der Funktions-Rumpf besteht aus Anweisungen, die mittels geschweifter Klammern zu einem Block zusammengefasst werden.
- Die Anweisungen verwenden die Übergabeparameter - diese werden wie Variablen benutzt.
- Mit der `return`-Anweisung kann die Ausführung der Funktion beendet und ein Rückgabewert festgelegt werden.

Rumpf einer Funktion

Beispiel

Wir programmieren die `isdigit`-Funktion nach:

Rumpf der `isdigit`-Funktion

```
1 int my_isdigit(int c) {  
2     return (c >= 48 && c <= 57);  
3 }
```

- `(c >= 48 && c <= 57)`

Dieser Ausdruck hat den Wert 1, falls der Wert von `c` zwischen 48 und 57 liegt (das sind die ASCII-Codes aller Ziffern), und sonst den Wert 0.

- `return (c >= 48 && c <= 57);`

Beendet die Funktion und gibt den Wert des Ausdrucks zurück.

- Bei einer Funktion **mit Rückgabewert vom Typ** T muss jede Ausführung mit einer Anweisung der Form `return <A>;` oder `return(<A>;` beendet werden.
- Hierbei ist $<A>$ ein Ausdruck, dessen Typ dem Rückgabebetyp T entspricht.
- Der Wert von $<A>$ legt den Rückgabewert fest.
- Der Rumpf einer Funktion **ohne Rückgabewert** kann die Anweisung `return;` enthalten. Diese beendet deren Ausführung.

Aufruf einer Funktion

Funktion mit Eingabeparametern

Der Funktionskopf legt fest, wie die Funktion aufgerufen wird. Bei Funktionsaufruf werden die Anweisungen im Rumpf ausgeführt.
Eine Funktion mit dem Kopf

`<T> <Name>(<T_1> <P_1>, ..., <T_n> <P_n>)`

wird in folgender Form aufgerufen:

`<Name>(<Wert_1>, ..., <Wert_n>)`

mit Übergabe-Werten `<Wert_i>` vom jeweiligen Typ `<T_i>` des zugehörigen Parameters `<P_i>`.

Im Funktionsrumpf werden die übergebenen Werten für die Parameter in die Anweisungen eingesetzt.

Aufruf einer Funktion

Funktion ohne Eingabeparameter

Eine Funktion mit dem Kopf

<T> <Name>()

wird in folgender Form aufgerufen:

<Name>()

(die runden Klammern bleiben also leer)

Beispielaufrufe der Funktion isdigit

```
1 int my_isdigit(int c) {  
2     return (c >= 48 && c <= 57);  
3 }
```

Für den Parameter `c` kann bei Aufruf eine ganze Zahl eingesetzt werden:

- `my_isdigit('5')`: Testet ob das Zeichen '5' eine Ziffer ist (Rückgabewert 1)
- `my_isdigit('A')`: Testet ob das Zeichen 'A' eine Ziffer ist (Rückgabewert 0)
- `my_isdigit(50)`: Testet ob 50 der ASCII-Code einer Ziffer ist (Rückgabewert 1)

Aufruf einer Funktion in Anweisungen

Funktion ohne Rückgabewert

Der Aufruf einer Funktion mit dem Kopf

```
void <Name>(<T_1> <P_1>, ..., <T_n> <P_n>)
```

kann nur als Einzelanweisung erfolgen:

```
<Name>(<Wert_1>, ..., <Wert_n>);
```

(der Strichpunkt schließt die Anweisung ab)

Aufruf einer Funktion in Anweisungen

Funktion mit Rückgabewert

Der Aufruf einer Funktion mit dem Kopf

$\langle T \rangle \langle \text{Name} \rangle (\langle T_1 \rangle \langle P_1 \rangle, \dots, \langle T_n \rangle \langle P_n \rangle)$

kann in zwei Formen erfolgen:

- als Einzelanweisung (wie Funktionen ohne Rückgabewert) - in diesem Fall wird der Rückgabewert nicht benutzt
- als Teilausdruck einer Einzelanweisung, z.B.
 - Wertzuweisung des Rückgabewerts an eine Variable von Typ $\langle T \rangle$
 - Vergleich des Rückgabewerts mit einem anderen Wert vom Typ $\langle T \rangle$
 - Übergabe des Rückgabewerts an einen (anderen) Funktionsaufruf für einen Eingabeparameter vom Typ $\langle T \rangle$

Beispiele

- `int n = rand() % 128;`
Rückgabewert wird modulo 128 gerechnet und das Ergebnis in `n` gespeichert
- `putchar(toUpper('a'));`
Rückgabewert von `toUpper` wird an `putchar` übergeben und dann von `putchar` auf Kommandozeile ausgegeben (`toUpper` wird vor `putchar` ausgeführt).
- `if (putchar('a') == EOF) {...}`
Rückgabewert von `putchar` wird mit `EOF` verglichen.
- `printf("Hallo");`
Funktionsaufruf als Einzelanweisung ohne Verwendung des Rückgabewerts

Ausführung einer Funktion

Durch Aufruf einer Funktion wird diese ausgeführt.

Funktions-Ausführung

- Die übergebenen Werte werden im Rumpf für die jeweils zugehörigen Parameter eingesetzt
- Dann werden die Anweisungen mit den übergebenen Werten in der angegebenen Reihenfolge ausgeführt.
- Der Aufruf erhält als Wert den dabei berechneten Rückgabewert der Funktion.

Ausführung der Funktion `isdigit`

Der Aufruf `my_isdigit('1')` hat den Wert 1

Eine Funktion kann über einen Rückgabewert

- das Ergebnis einer Berechnung zurückgeben
- anzeigen, ob ein Fehler aufgetreten ist

Hierbei müssen Rückgabewerte, die Fehler anzeigen, **unterscheidbar** sein von Rückgabewerten, die Ergebnisse einer erfolgreichen Berechnung darstellen

Verwendung von Rückgabewerten

Rückgabewerte können also insbesondere als **Fehlerwerte** dienen. Bei Aufruf kann abhängig vom Rückgabewert eine geeignete Fehlerbehandlung (Abbruch, Wiederholung, Fehlermeldung) erfolgen.

Beispiel

- `my_isdigit` gibt das Ergebnis einer Berechnung zurück
- `putchar` gibt im Erfolgsfall das ausgegebene Zeichen, und im Fehlerfall EOF zurück

```
1 if (putchar('q') == EOF) {  
2     printf("Fehler bei der Ausgabe");  
3 }
```

Auch als Teil des Ausdrucks `putchar('q') == EOF` gibt `putchar('q')` das Zeichen `q` aus

Wo steht der Code einer Funktion?

Eine Funktion muss **vor ihrem ersten Aufruf** (am besten vor der `main`-Funktion) **deklariert** werden.

Die **Deklaration** besteht **nur aus dem Prototyp** (dem Kopf, der **Signatur**) der Funktion (der Rumpf wird also hier weggelassen), abgeschlossen mit einem Semikolon.

Die **Definiton** der Funktion mit Prototyp **inklusive Rumpf** erfolgt dann **an einer beliebigen Stelle** (in der Regel nach der `main`-Funktion)

Beispielprogramm

```
1 #include <stdio.h>
2 #include <time.h>
3 #include <stdlib.h>
4
5 int my_isdigit(int c); /*Deklaration*/
6
7 int main(void)
8 {
9     char c;
10    int b;
11    srand(time(NULL));
12    c = rand() % 128;
13    b = my_isdigit(c); /*Aufruf*/
14    printf("%i", b);
15    return 0;
16 }
17
18 int my_isdigit(int c) /*Definition*/
19 {
20     return (c >= 48 && c <= 57);
21 }
```

Eine eigene Funktion entwerfen

Programmiere eine Funktion, mit der man ein frei wählbares ASCII-Zeichen eine gewünschte Anzahl oft ausgeben kann.

- Sprechender Funktionsname: `print_symbol`
- Eingabeparameter für das gewählte Zeichen: `int c`
- Eingabeparameter für die gewünschte Anzahl: `int n`
- Rückgabebetyp: `void` (einfache Version ohne Unterscheidung zwischen Fehler- und Erfolgsfall)

Eine eigene Funktion entwerfen

Programmiere eine Funktion, mit der man ein frei wählbares ASCII-Zeichen eine gewünschte Anzahl oft ausgeben kann.

```
1 void print_symbol(int c, int n) {  
2     int i;  
3     for (i = 0; i < n; ++i) {  
4         putchar(c);  
5     }  
6 }
```

- Wir benötigen keine `return`-Anweisung: Die **Ausführung endet automatisch** nach Abarbeitung der `for`-Schleife bei der abschließenden `}`-Klammer des Funktionsrumpfs.
- Wir überprüfen in der Regel nicht die **Gültigkeit der Eingabeparameter**, also zum Beispiel, ob für `n` eine nicht-negative Zahl übergeben wurde (so sind auch die Bibliotheksfunktionen programmiert - deren korrekte Benutzung liegt in der Verantwortung des Verwenders).

Eine eigene Funktion entwerfen

Übung

Erweitere die Funktion `print_symbol` um eine Unterscheidung zwischen Fehler- und Erfolgsfall: Gib `EOF` zurück, wenn es bei einem Aufruf von `putchar` zu einem Fehler kommt, und sonst `n`.

```
1 int print_symbol(int c, int n) {  
2     int i;  
3     for (i = 0; i < n; ++i) {  
4         if (putchar(c) == EOF) {  
5             return EOF;  
6         }  
7     }  
8     return n;  
9 }
```