2.3 Funktionen/Funktionen erstellen

2.3.1 Funktionsdefinition

- Die Funktionsdefinition beschreibt eine Funktion vollständig.
- Sie ist vergleichbar mit dem Innenschaltbild eines ICs.

```
1  void funkname(void) --- Funktionsname, Ein- und Ausgaenge ---Kopf
2  {                --- Anfang                                 ---Rumpf
3      /* Variablen    */
4      ...
5      /* Anweisungen */
6      ...
7      return;        --- Ruecksprung
8  }                --- Ende
```

2.3.2 Funktionsaufruf

- Der Funktionsaufruf ist vergleichbar mit einem Block in einem Gesamtschaltbild.
- Er beginnt mit dem Sprung in den Anweisungsteil der Funktionsdefinition und endet mit dem Rücksprung.

```
1      funkname();
```

2.3.3 Aufbau eines Programmes mit mehreren Funktionen

Jede Funktion muss zuerst definiert (vollständig beschrieben) werden und kann danach beliebig oft und von jeder anderen Funktion benutzt werden (eine Funktion darf sich sogar selbst aufrufen). Wichtig ist nur, dass jede Funktion erst definiert worden sein muss, bevor sie benutzt werden kann. Deshalb muss die `main()`-Funktion, die die oberste Funktion in der Hierarchie darstellt, im Quelltext ganz unten stehen.

```
1  void baust1(void)
2  {
3      ...
4      return;
5  }
6
7  void baust2(void)
8  {
9      ... /* moeglich: baust1(); */
10     return;
11 }
12
13 int main(void)
14 {
15     ... /* moeglich: baust2(); baust1(); */
16     return 0;
17 }
```

2.3.4 Prototypen

Die Festlegung, dass eine Funktion erst definiert (vollständig beschrieben) werden muss, bevor sie benutzt werden kann, ist nur für den Compiler wichtig: Der C-Compiler sieht den Quelltext nur ein einziges Mal durch (ein so genannter 1-Pass-Compiler) und muss daher alle Bausteine, die er einbauen will, vorher im Quelltext kennengelernt haben (1-Pass-Compiler compilieren besonders schnell und sind darum bei Programmierern sehr beliebt).

Andererseits passt es zur Top-Down-Methode eher, wenn die Bausteine, die in der Hierarchie ganz oben liegen (wie etwa die `main()`-Funktion), auch im Quelltext weit oben stehen.

Dazu gibt es eine Lösung: Man teilt dem C-Compiler am Beginn des Quelltextes (oder an anderer, beliebiger Stelle) mit, welche Bausteine man benutzen möchte und wie sie zu benutzen sind. Diese Mitteilung heißt Prototyp oder Deklaration:

```
1 void baust1(void); /* Prototyp fuer baust1() */
2 void baust2(void); /* Prototyp fuer baust2() */
```

Der Prototyp sieht also genauso aus wie der Funktionskopf (die erste Zeile der Funktionsdefinition), jedoch gefolgt von einem Semikolon (anders als bei `if`, `while` und `for` bedeutet hier ein Semikolon etwas ganz anderes als ein Paar geschweifter Klammern!). Das Wort `void` zu Beginn der Zeile gibt an, dass die Funktion keine Daten an die aufrufende Umgebung zurückgibt; das Paar runder Klammern mit dem Wort `void` darin teilt mit, dass die Funktion außerdem keine weiteren Daten zu ihrer eigenen Benutzung benötigt.

Da obige Beispiel kann also umgeschrieben werden:

```
1  void baust1(void); /* Prototyp */
2  void baust2(void); /* Prototyp */
3
4  int main(void) /* Beginn der Definition von main() */
5  {
6      ...
7      return 0;
8  }
9
10 void baust1(void) /* Beginn der Definition von baust1() */
11 {
12     ...
13     return;
14 }
15
16 void baust2(void) /* Beginn der Definition von baust2() */
17 {
18     ...
19     return;
20 }
```

2.3.5 Header-Dateien

Prototypen werden auch Funktions-Header genannt (da sie außer dem Semikolon nur den Funktionskopf enthalten). Oft werden viele solcher Header in so genannten Header-Dateien zusammengefasst (die zugehörigen Dateinamen enden meist auf `.h`). Mit dem `#include`-Befehl kann man den Inhalt dieser Header-Dateien dann in eigene Programme einbinden. Mit der Einbindung von `stdio.h` wird der Compiler z.B. davon informiert, dass es eine Funktion mit dem Namen `printf` gibt. Die zu `printf` gehörende Funktionsdefinition dagegen wird an anderer Stelle eingebunden, nämlich durch den Linker; aber darum braucht sich der Programmierer bei `printf` und anderen Funktionen aus der C-Standard-Bibliothek in der Regel nicht kümmern.

Fehlt der Prototyp, weil man ihn nicht geschrieben oder nicht durch eine Header-Datei eingebunden hat, erhält man eine Compiler-Warnung, z.B. dieser Form:

```
foo.c:3: warning: incompatible implicit declaration of
                  built-in function printf
```

Solch eine Warnung muss man ernst nehmen, weil der Compiler nun nicht mehr feststellen kann, wie die Funktion ins Programm eingebunden wird. Er wählt dann (leider) eine Default-Einbindung, die aber meist nicht passt. Das Programm wird zwar compiliert, funktioniert aber nicht (kann man bei Funktionen wie `sin()` und `cos()` gut ausprobieren).