2.5.V Funktionen/Parameter – Versuch

2.5.V.1 Ziele

Hier soll an einem Beispiel untersucht werden, wie man einen Programmabschnitt in eine Funktion verwandeln kann.

2.5.V.2 Fragen

- Gibt es ein Schema? Welche Schritte sind sinnvoll?
- Was muss man beachten? Gibt es Prinzipien/Regeln?

2.5.V.3 Schritt 1: Festlegen des Funktionskopfes

Im folgenden Programm werden zwei Geldbeträge addiert und ausgegeben.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int wert1;
6      int wert2;
7      int summe;
8      int anteil_eur;
9      int anteil_ct;
10                                                  /* Eingabe */
11     wert1=350; /* 3,50 EUR */
12     wert2=620; /* 6,20 EUR */
13                                                  /* Verarbeitung */
14     summe=wert1+wert2;
15                                                  /* Ausgabe */
16     anteil_eur=summe/100;
17     anteil_ct =summe%100;
18     printf("%i,%02i EUR", anteil_eur, anteil_ct);
19     printf("\n");
20     return 0;
21 }
```

Die Eingabe und die Berechnung (Zeile 12) erfolgen in Cent. Die Ausgabe erfolgt getrennt nach Euro (Zeile 16) und Cent (Zeile 17). Nun soll der Ausgabeteil (Zeilen 16-18) in eine eigene Funktion ausgegliedert werden. Hier ist zu beachten:

- Die Funktion braucht einen Namen. Der Name soll das wiederspiegeln, was die Funktion tun soll. Wir nennen sie:
  `printgeldbetrag()`
- Die Funktion soll so universal wie möglich sein. Es könnte ja sein, dass man die Funktion später noch einmal für einen anderen Zweck braucht. Deshalb wird Zeile 19 (Zeilenende) nicht mit in die Funktion einbezogen. Es könnte zum Beispiel sein, dass man mehrere Geldbeträge nebeneinander in eine Zeile schreiben möchte.
- Die Funktion braucht einen Parameter: Welcher Betrag soll ausgegeben werden?
- Meist versucht man, so wenige Parameter wie möglich zu benutzen. Die einzelnen Programmteile (Funktionen) sollen möglichst unabhängig voneinander sein. Man nennt dies *das Prinzip der schmalen Datenkopplung*. Hier reicht offenbar dieser eine Parameter.

- Der Parameter ist vom Typ `int`, weil wir Beträge, die ganzzahlig in Cent vorliegen, ausgeben wollen.

Der Funktionskopf sieht dann so aus:

```
1 void printgeldbetrag(int betrag)
```

Damit kann man den Prototyp und eine leere Funktionsdefinition erstellen:

```
1 void printgeldbetrag(int betrag); /* Prototyp    */
2 void printgeldbetrag(int betrag)  /* Definition */
3 {                                 /* Definition */
4     printf("Dummy-Inhalt\n");     /* Definition */
5 }                                 /* Definition */
```

Diese Zeilen könnte man übrigens jetzt schon in das Programm einfügen (den Prototypen vor `main` und die Definition hinter `main`). Denn solange eine Funktion nicht aufgerufen wird, passiert nichts.

2.5.V.4 Schritt 2: Auftrennen

Nun muss alles, was in die Funktion gehören soll, vom Rest getrennt werden. Dazu eignen sich Blockklammern.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int wert1;
6      int wert2;
7      int summe;
8                                                   /* Eingabe */
9      wert1=350; /* 3,50 EUR */
10     wert2=620; /* 6,20 EUR */
11                                                  /* Verarbeitung */
12     summe=wert1+wert2;
13                                                  /* Ausgabe */
14     {
15         int anteil_eur;
16         int anteil_ct;
17
18         anteil_eur=summe/100;
19         anteil_ct =summe%100;
20         printf("%i,%02i EUR", anteil_eur, anteil_ct);
21     }
22     printf("\n");
23     return 0;
24 }
```

- Zeile 14+18: Blockklammern. Sie fassen Anweisungen optisch zusammen, und sie bilden einen Bereich, in dem man Variablen neu vereinbaren kann.
- Zeile 15+16: Die beiden Variablen `antein_eur` und `antein_ct` werden nur für die Ausgabe benutzt. Darum ist es sinnvoll, sie in diesem Block zu vereinbaren.
- Zeile 18+19: Die Variable `summe` wird außerhalb und innerhalb des Blocks benutzt. Deshalb muss sie später durch einen Parameter ersetzt werden.

2.5.V.5 Schritt 3: Auslagern

Jetzt kann der Block in die bisher leere Funktionsdefinition ausgelagert werden.

```
1  #include <stdio.h>
2  void printgeldbetrag(int betrag);
3
4  int main(void)
5  {
6      int wert1;
7      int wert2;
8      int summe;
9                                                    /* Eingabe */
10     wert1=350; /* 3,50 EUR */
11     wert2=620; /* 6,20 EUR */
12                                                   /* Verarbeitung */
13     summe=wert1+wert2;
14                                                   /* Ausgabe */
15     printgeldbetrag(summe);
16     printf("\n");
17     return 0;
18 }
19
20 void printgeldbetrag(int betrag)
21 {
22     int anteil_eur;
23     int anteil_ct;
24
25     anteil_eur=betrag/100;
26     anteil_ct =betrag%100;
27     printf("%i,%02i EUR", anteil_eur, anteil_ct);
28 }
```

- Zeile 15: Hier ist der Funktionsaufruf. Er steht an der Stelle, an der vorher der Block stand.
- Zeilen 25+26: Der Parameter heißt hier `betrag` statt `summe`, also ein allgemeiner anstelle eines speziellen Namens. Das kann und sollte man so machen, es ist aber nicht notwendig.

2.5.V.6 Schritt 4: Testen

Jede Funktion *muss* getestet werden. Nach allen Erfahrungen gilt:
Was man nicht getestet hat, funktioniert nicht.
Bisher hatten wir nur ein Beispielprogramm vorhanden, jetzt brauchen wir ein Testprogramm. So kann es aussehen:

```
1  #include <stdio.h>
2  void printgeldbetrag(int betrag);
3
4  int main(void)
5  {
6      int betrag;
7      printf("Eingabe in ct: ");
8      scanf("%i", &betrag);
9      printgeldbetrag(betrag);
10     return 0;
```

```
11 }
12 void printgeldbetrag(int betrag)
13 { ... }
```

Ein Testprogramm erlaubt es dem Benutzer, beliebige Werte einzugeben und gibt ihm dann die Ergebnisse aus. Diese Werte (oder Wertekombinationen, wenn man mehrere Parameter hat), heißen *Testfälle*. Die Testfälle gehören immer in die Dokumentation eines Programmierprojekts.

Man kann so ein Testprogramm auch automatisieren, indem man schon vorher alle Testfälle festlegt:

```
1  #include <stdio.h>
2  #include <limits.h> /* fuer INT_MIN und INT_MAX */
3  void printgeldbetrag(int betrag);
4
5  int main(void)
6  {
7      printgeldbetrag(0);          printf("\n");
8      printgeldbetrag(1);          printf("\n");
9      printgeldbetrag(99);         printf("\n");
10     printgeldbetrag(801);        printf("\n");
11     printgeldbetrag(899);        printf("\n");
12     printgeldbetrag(-1);         printf("\n");
13     printgeldbetrag(-99);        printf("\n");
14     printgeldbetrag(-801);       printf("\n");
15     printgeldbetrag(-899);       printf("\n");
16     printgeldbetrag(INT_MIN);    printf("\n");
17     printgeldbetrag(INT_MAX);    printf("\n");
18     return 0;
19 }
20 void printgeldbetrag(int betrag)
21 { ... }
```

Meistens ist es sinnvoll, Rand- oder Extremwerte in die Testfälle einzubeziehen.

2.5.V.7 Auswertung

a) Vorgehensweise/Schritte:

b) Prinzip(ien):

c) Ergebnisse der Tests: