Unterprogramm-Technik
Funktionen

Inhalt

Anwendung von Funktionen

- Funktionen ermöglichen den modularen Aufbau von Programmen, d.h. das zu entwickelnde System wird in Teilsysteme (Teilaufgaben) zerlegt.

  ➔ Folie: "II.1 Prinzipien der Softwareentwicklung"
  Folie: "II.2 Entwurfsrichtungen bei der Strukturierung"

  ➔ Planung: **Top-Down-Prinzip**
  (Bsp.: Top-Down-Entwurf "Packprogramm")

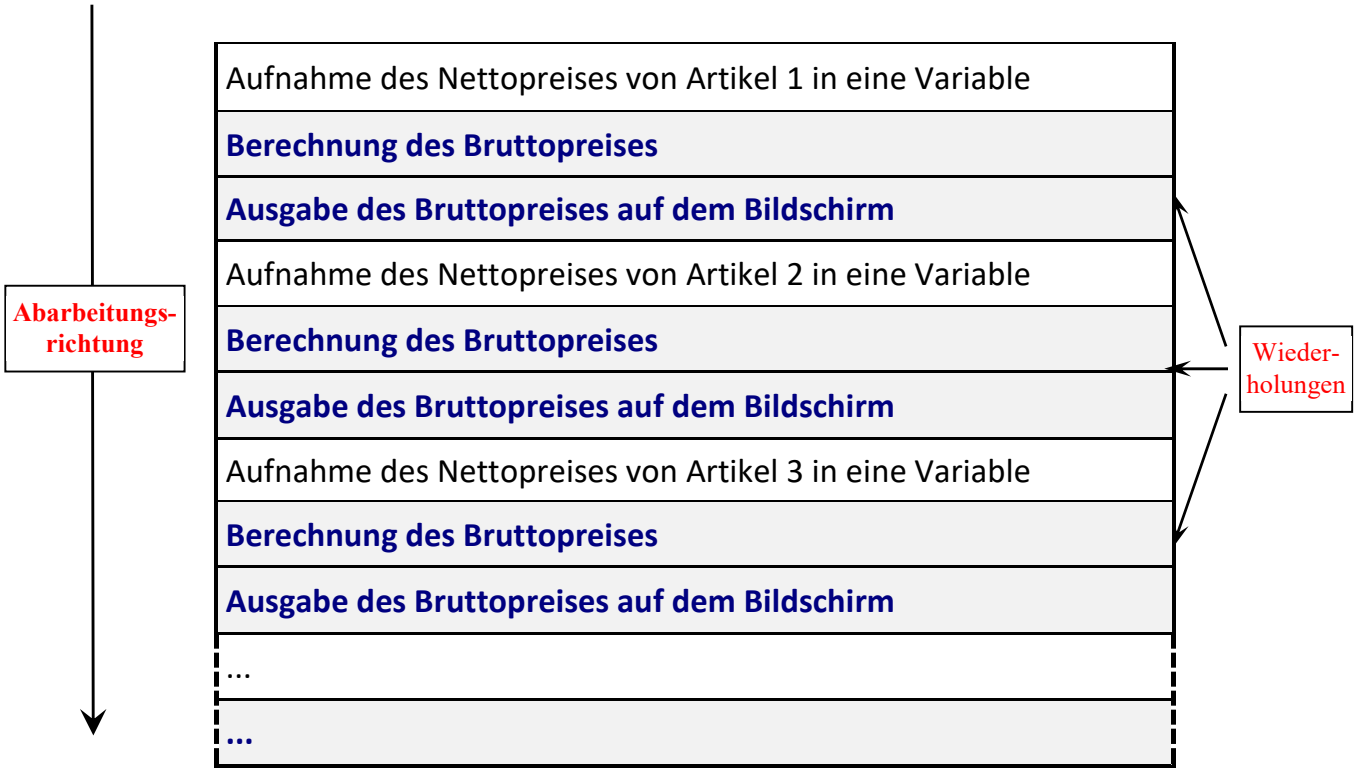
  ➔ Codierung: **Bottom-Up-Prinzip**

  + Erhöhung der Übersichtlichkeit durch Reduzierung der Komplexität
  + leichtere Erstellung und separate Tests der Module
  + Wiederverwendung in anderen Programmen
    ➔ Funktionsbibliothek von **C**

- Vermeidung von redundantem Programmcode, also Anweisungsfolgen, die wiederholt abgearbeitet werden müssen und zudem unbekannt ist, wie oft diese Anweisungsfolgen durchlaufen werden.

  Bsp: Für verschiedene Artikel, deren Netto-Preise (ohne MwSt.) bekannt sind, sollen ähnlich einem Kassenbon deren Brutto-Preise und die Gesamtsumme mit dem Anteil MwSt. auf dem Monitor ausgegeben werden.

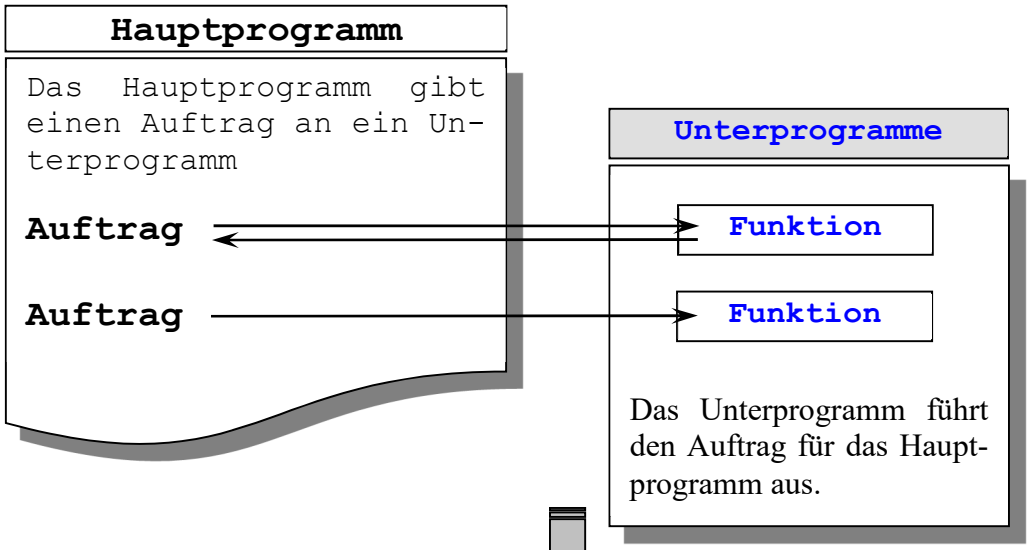
  Struktogramm: Artikelpreise Netto ➔ Brutto



Auswertung des Struktogramms: **Netto ➔ Brutto**

- einzelne Anweisungen (grau unterlegt) wiederholen sich
  - ⇒ Berechnung des Bruttopreises
  - ⇒ Ausgabe des Bruttopreises auf dem Monitor
- Kopien dieser Anweisungen im Quellcode :
  - führen zum Aufbauschen des Quellcodes
  - ermöglichen keinen separaten Testdurchlauf
  - sind nur möglich, wenn die Anzahl der Artikel bekannt ist
- Verwendung einer Wiederholungsstruktur (Schleife) hat den Nachteil, dass diese nicht gezielt aufgerufen werden kann (Schleifen besitzen keinen Namen !)
- eleganteste Lösung ist das Herauslösen der redundanten Anweisungen in ein aufrufbares Unterprogramm ➔ Funktion

Unterprogrammruf als Auftrag des Hauptprogramms



Hauptprogramm

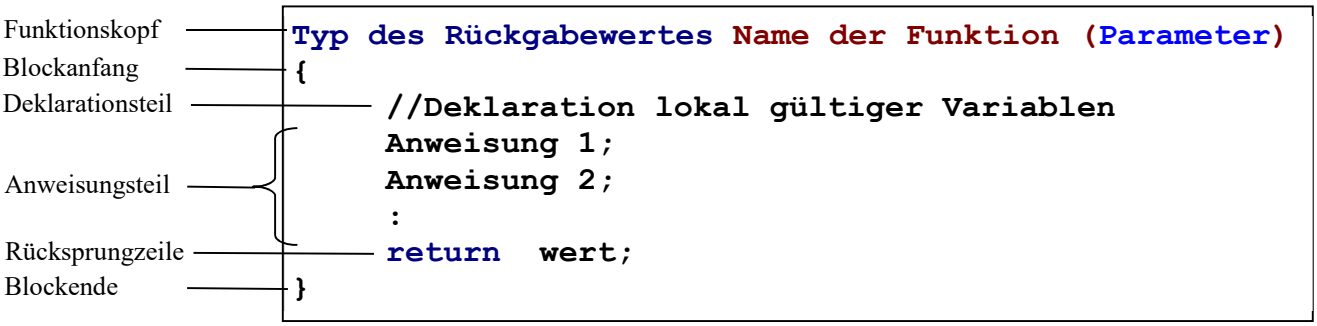
Aufnahme des Nettopreises von Artikel 1 in eine Variable
UP mwst
Aufnahme des Nettopreises von Artikel 2 in eine Variable
UP mwst
Aufnahme des Nettopreises von Artikel 3 in eine Variable
UP mwst
...

Funktion mwst

Berechnung des Bruttopreises
Ausgabe des Bruttopreises auf dem Bildschirm

Merke: Sinn einer Funktion ist es, einen Anweisungsblock (Algorithmus) zur Erledigung einer Teilaufgabe aus dem Hauptprogramm auszulagern und diesen mit einem Namen zu verbinden.
Sie kann bei ihrem Aufruf bestimmte Werte übernehmen, mit diesen innerhalb der Funktion arbeiten, und auch Werte an die Stelle im Programmablauf zurückgeben, von wo ihr Aufruf erfolgte!

Definition einer Funktion



Syntax der Funktionsvereinbarung

- ✓ Der Funktionskopf wird immer über dem Hauptprogramm (main-Fkt.), aber unterhalb der Präcompiler-Anweisungen geschrieben.
  ⇒ Der Funktionskopf informiert den Compiler darüber, nach welcher Syntax die Funktion aus dem Hauptprogramm oder einer anderen Funktion aufgerufen werden muss (s. "Funktionsköpfe" in einer Header-Datei).

  Hinweis:
  Soll der Programmcode einer Funktion unterhalb der main-Fkt. stehen, muss nur die Kopfzeile der Funktion + Semikolon vor die main-Fkt. kopiert werden ("Vorwärtsdeklaration!").

  Die Vorwärtsdeklaration ist sogar zwingend notwendig, wenn sich 2 Funktionen gegenseitig aufrufen !

- ✓ Mit Typ des Rückgabewertes wird der Datentyp desjenigen Wertes festgelegt, welcher von der Funktion an die aufrufende Stelle des Programmablaufs zurückgegeben wird.
  ⇒ Bei Rückgabe eines Wertes ist die Funktion mit dem gewünschten Datentyp zu deklarieren. Die Rückgabe des Wertes erfolgt durch das Schlüsselwort return. Der zurückgegebene Wert muss mit dem Rückgabetyp im Funktionskopf übereinstimmen!
  Hinweis:
  Der Wert selber kann entweder in einer Variable gespeichert oder direkt dem Standard-Ausgabegerät (Monitor) zugeführt werden.
  ⇒ Soll die Funktion keinen Wert zurückgeben, ist die Funktion mit void ("leer") zu deklarieren. Die Rücksprungzeile ist in diesem Fall nicht nötig.

- ✓ Innerhalb der runden Klammern können sog. Parameter deklariert werden, die bestimmte Werte beim Aufruf der Funktion übernehmen (später mehr).
  ⇒ Sind keine Parameter nötig, wird void eingefügt.
  ⇒ Aufgerufen wird die Funktion ohne Angabe von void !

```c
#include <stdio.h>
void add(void)
{
   printf("Summe = %i",10+20);
}
int main(void)
{
   add();
   return 0;
}
```

```c
//Vorwärtsdeklaration
void add(void);
int main(void)
{
   add();
   return 0;
}
void add(void)
{
   printf("Summe = %i",10+20);
}
```

```c
#include <stdio.h>
//gibt Ganzzahl zurück
int add(void)
{
   return 10+20;
}
int main(void)
{
   int summe;
   summe = add();
   printf("Summe = %i", summe);
   return 0;
}
```

```c
#include <stdio.h>

//gibt Ganzzahl zurück
int add(void)
{
   return 10+20;
}
int main(void)
{
   printf("Summe = %i", add());
   return 0;
}
```

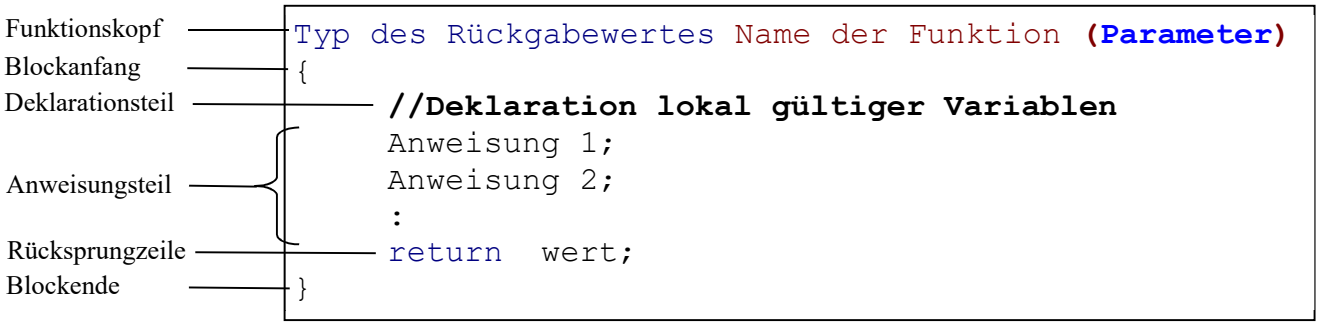
Demo 1

Aufgabe 1: Für verschiedene Artikel, deren Netto-Preise (ohne MwSt.) bekannt sind, sollen ähnlich einem Kassenbon deren Brutto-Preise und die Gesamtsumme mit dem Anteil MwSt. auf dem Monitor ausgegeben werden.

Listing 1: **net2bru1.c**

```
1   /* DEMO 1
2    * Beispiel fuer den Einsatz einer Funktion
3    * ➔ Berechnung der Mehrwertsteuer ohne Rueckgabewert und Parameter
4    */
5
6   #include <stdio.h>
7   #include <conio.h>
8   #define MWST_SATZ 0.19       //MWST_SATZ wird als Konstante vereinbart
9
10  //Deklaration globaler Variablen (nutzbar in allen Funktionen !)
11  float preis;
12
13  //Definition einer Funktion ohne Rueckgabe eines Wertes
14  void mwst(void)
15  {
16          //Deklaration lokaler Variablen
17          float brutto;
18          brutto = preis*(MWST_SATZ+1);
19          printf("\n+\t%.2f Euro\n", brutto);
20  }
21
22  //Beginn des Hauptprogramms
23  int main(void)
24  {
25          //Deklaration lokaler Variablen bzw. lokaler Konstanten
26          const float ARTIKEL_1 = 7.20;
27          const float ARTIKEL_2 = 1.40;
28          const float ARTIKEL_3 = 5.60;
29
30          clrscr();
31          preis = ARTIKEL_1;
32          mwst();
33          preis = ARTIKEL_2;
34          mwst();
35          preis = ARTIKEL_3;
36          mwst();
37
38          printf("-----------------");
39          preis = ARTIKEL_1 + ARTIKEL_2 + ARTIKEL_3;
40          mwst();
41          printf("=================\n\n\n");
42
43          getchar();
44          return 0;
45  }
```

Parameter einer Funktion



Syntax der Funktionsvereinbarung

✓ Über den Funktionskopf können beim Aufruf einer Funktion Werte an sogenannte Parameter übergeben werden. Die Parameter werden innerhalb der runden Klammern wie normale Variablen, durch Komma getrennt, deklariert.

```
#include <stdio.h>
void add(int z1, int z2)
{
   printf("Summe = %i" ,z1+z2);
}
int main(void)
{
   add(10, 20);
   return 0;
}
```

✓ Beim Aufruf einer Funktion mit Parametern müssen entsprechende Werte (sogenannte Argumente) zur Übernahme in die Parameter-Variablen hinzugefügt werden.

Achtung: Anzahl und Datentyp der Argumente müssen mit den Parametern übereinstimmen!

✓ Innerhalb der Funktion können Sie die Parametervariablen wie lokal deklarierte Variablen handhaben. D.h., sie existieren nur innerhalb der Funktion und sie sind temporär.

Erst beim Aufruf der Funktion werden für die Parametervariablen Speicherplätze reserviert, eine Kopie der Argumente dort abgelegt und nach Beendigung der Funktion die Speicherplätze wieder freigegeben.

```
#include <stdio.h>
void add(int z1, int z2)
{
   int summe;
   z2 = 30;
   summe = z1+ z2;
   printf("Summe = %i" ,summe);
}
int main()
{
   add(10, 20);
   return 0;
}
```

✓ **Merke:** In **C-Funktionen** werden die Argumente **nie** nach Namen (*Bezeichner*), sondern nach Wert übergeben **!**

➢ **"call by value"**

D.h., wenn beim Aufruf einer Funktion statt konkreter Werte Variablen, bzw. deren Bezeichner, übergeben werden, wird immer nur der zugehörige Wert als Kopie an die Parameter übergeben.

Die Änderung von Parametern in einer Funktion wirkt sich nicht auf gleichnamige, lokale Variablen einer anderen Funktion aus!

```
#include <stdio.h>
void add(int z1, int z2)
{
   z1 = 50;
   z2 = 100;
   printf("Parameter z1: %i\n" ,z1);
   printf("Parameter z2: %i\n" ,z2);
}
int main()
{
   int z1, z2;  //lokale Variablen
   z1 = 10;
   z2 = 20;

   add(z1, z2);

   printf("Variable z1: %i\n" ,z1);
   printf("Variable z2: %i\n" ,z2);
   return 0;
}
```

Demo 2

Aufgabe 2: Die Funktion zur Berechnung der Mehrwertsteuer von Aufgabe 1 soll so verändert werden, dass auf die globale Variable "preis" verzichtet werden kann.

⇒ Die Funktion "mwst" wird hierzu um den Parameter "preis" ergänzt, welcher die akt. Artikelpreise beim Aufruf der Funktion übernimmt.

Listing 2: **net2bru2.c**

```
1   /* DEMO 2
2    * Berechnung der Mehrwertsteuer
3    * → Funktion mit Parameter
4    */
5
6   #include <stdio.h>
7   #include <conio.h>
8   #define MWST_SATZ 0.16       //MWST_SATZ wird als Konstante vereinbart
9
10  /*Definition einer Funktion
11   * → ohne Rueckgabe eines Wertes
12   * → mit Uebernahme eines Arguments(Artikelpreis)in den Parameter "preis"
13   */
14  void mwst(float preis)
15  {
16        //Deklaration lokaler Variablen
17        float brutto;
18        brutto = preis*(MWST_SATZ+1);
19        printf("\n+\t%.2f DM\n", brutto);
20  }
21
22  //Beginn des Hauptprogramms
23  int main(void)
24  {
25        //Deklaration lokaler Variablen bzw. lokaler Konstanten
26        const float ARTIKEL_1 = 7.20;
27        const float ARTIKEL_2 = 1.40;
28        const float ARTIKEL_3 = 5.60;
29
30        clrscr();
31
32        mwst(ARTIKEL_1);
33        mwst(ARTIKEL_2);
34        mwst(ARTIKEL_3);
35
36        printf("----------------");
37        mwst(ARTIKEL_1 + ARTIKEL_2 + ARTIKEL_3);
38        printf("================\n\n\n");
39
40        getchar();
41        return 0;
42  }
```

Übung 1

Auf einem Rechnungsformular ist zum Eintragen der Beträge für den Warenwert, die Mehrwertsteuer und den Endbetrag ein bestimmter Platz vorgesehen. Zum Schutz vor Fälschungen sollen die nicht für Ziffern bzw. den Dezimalpunkt benötigten Ausgabepositionen durch einen Stern ausgefüllt werden. Beispiel:

Faelschungssichere Rechnung

```
Warenwert:  1845.30

Warenwert:       €*****1845.30***
Mehrwertsteuer:  €******295.25***
Endbetrag:       €*****2140.55***
```

Nach Eingabe des Warenwerts und Berechnung der Mehrwertsteuer sind die drei genannten Beträge mit führenden und abschließenden Sternchen auszugeben.

Schreiben Sie ein Programm, in welchem ein Unterprogramm

- die Anzahl der Sterne so festlegt, dass diese und der jeweilige auszugebende Betrag (*Warenwert, Mehrwertsteuer, Endbetrag*) zusammen gerade den zur Verfügung stehenden Platz ausfüllen.
- Vermeiden Sie die Deklaration globaler Variablen !

Erinnerung: **Beginnen Sie mit dem Struktogramm !**

Struktogramm: Fälschungssichere Rechnung

Hauptprogramm

Warenwert
MwSt berechnen
Ausgabe Warenwert
Ausgabe MwSt
Ausgabe Endebetrag

Ausgabe (Betrag)

Sterne = STELLEN-6 Kopie des Betrags anlegen
Betrag = Betrag / 10 Sterne = Sterne -1
wiederhole solange Betrag >= 1
Für alle führenden Sterne wiederhole
*
Kopie des Betrags und drei Sterne

List

```c
/* ÜBUNG 1
 * Fälschungssichere Rechnung
*/
//Vorwärtsdeklaration von Funktionen
void ausgeben(float fBetrag);

//Präprozesso-Direktiven (Textersetzer)
#include <stdio.h>
#include <conio.h>
#define MWST_SATZ 0.19  //Mehrwertsteuer-Satz
#define STELLEN   15    //Ausgabestellen: Sterne, Ziffern, Dezimalpunkt

/*** Hauptprogramm ***/
int main(void)
{
      float fWarenWert, fMwSt;

      printf("Faelschungssichere Rechnung\n");
      printf("============================");

      printf("\n\nWarenwert: ");
      scanf("%f",& fWarenWert);
      fMwSt = fWarenWert * MWST_SATZ;

      printf("\n\nWarenwert:\tEuro ");
      ausgabe(fWarenWert);

      printf("\nMehrwertsteuer:\t€ ");
      ausgabe(fMwSt);

      printf("\nEndbetrag:\t€ ");
      ausgabe(fWarenWert + fMwSt);

      fflush(stdin);
      getchar();
      return 0;
}

/*** Unterprogramme ***/
void ausgeben(float fBetrag)
{
      //Deklaration lokaler Variablen
      int   i, nSterne;
      float fBetragMem = fBetrag; //Kopie des übergebenen Betrags

      nSterne    = STELLEN - 6;              //max. Anzahl führender Sterne

      //Anzahl führender Sterne je nach übergebenen Betrag bestimmen
      do
      {
          fBetrag /= 10;              //für jede Vorkommastelle...
          nSterne --;                 //...einen Stern weniger
      }
      while (fBetrag >= 1);

      //Ausgabe
      for(i=0; i< nSterne; i++) printf("*");    //führende Sterne ausgeben
      printf("%-.2f***", fBetragMem);                   //Ausgabe des Betrags
}
```

Programmübungen
Funktionen

1. Zeitdifferenz

Schreiben Sie ein Programm, das zwei Uhrzeiten (hh, mm, ss) einliest, die Differenz (hh, mm, ss) zwischen den beiden Zeiten bestimmt und ausgibt. Die Berechnung der Differenz soll in einer Funktion erfolgen.

Hinweis: *Zur leichteren Berechnung bietet sich die Umrechnung der Uhrzeit in reine Sekunden an.*

2. Temperatur-Konverter

Temperaturwerte können in Celsius oder Fahrenheit angegeben werden. Der mathematische Zusammenhang ist aus folgender Tabelle ersichtlich:

Bsp.:

°C	°F
-20	-4
-10	14
0	32
10	50
20	68

Schreiben Sie ein Programm, welches eine Auswahl bzgl. der Richtung der Temperatur-Umrechnung ermöglicht und nach Eingabe eines Temperaturwertes die eigentliche Umrechnung mittels eines Unterprogramms durchführt. Ein- und Ausgaben gehören nicht zu den Aufgaben des jeweiligen Unterprogramms!

3. Normalgewicht

Das “Normalgewicht“ eines Erwachsenen berechnet sich nach der Formel:

Körpergröße (in cm) minus 100

Das “Idealgewicht“ beträgt bei Frauen 85%, bei Männern 90% des Normalgewichts.

Entwickeln Sie ein Unterprogramm **gewichtstest** mit den Parametern ***groesse_in_cm***, ***gewicht_in_kg*** und ***geschlecht***, welches entsprechende Eingaben verarbeitet und eine launige Beurteilung abgibt.

4. CRAPS

"CRAPS" ist ein beliebtes Würfelspiel in Amerika, das nach folgenden Regeln gespielt wird:

1. Es wird mit zwei Würfeln gewürfelt und immer nur die Augensumme betrachtet.
2. Erreicht man beim ersten Wurf eine 7 oder eine 11, so hat man gewonnen.
3. Würfelt man dagegen beim ersten Mal eine 2, 3 oder 12, so hat man verloren.
4. Bei jeder anderen Augensumme ist das Spiel noch offen und es wird weitergewürfelt, bis man entweder eine 7 erzielt (dann hat man verloren!) oder die im ersten Wurf erreichte Augensumme ein zweites Mal würfelt (dann hat man gewonnen!).

Dieses Spiel soll mit Hilfe des Computers gespielt werden. Das Anfangskapital soll 100 DM betragen, als Einsätze sind nur volle DM-Beträge erlaubt. Das Spiel endet, wenn der Spieler das wünscht oder, wenn er sein ganzes Kapital verloren hat.

Der Computer soll dabei 3 Aufgaben übernehmen:

- Er soll das Spielkapital verwalten, d.h. Einsätze entgegennehmen und das Kapital nach jedem Spiel entsprechend erhöhen oder vermindern,
- die Einhaltung der Spielregeln überwachen und
- das Würfeln übernehmen (*Zufallsgenerator*).

Vorgehensweise:

1. Zerlegen Sie das Gesamtproblem in Teilprobleme, die sich durch überschaubare Teilalgorithmen codieren lassen!
   Fertigen Sie hierzu ein **Top-Down-Entwurf** an!
2. Fertigen Sie sowohl für den Hauptalgorithmus (*Hauptprogramm*) als auch zu jedem Teilalgorithmus (*Funktion*) ein **Struktogramm** an!
3. Führen Sie die Codierung nach dem **Bottom-Up-Prinzip** durch!