Lösungen zur Vorlesung
GRUNDLAGEN DER INFORMATIK I
Studiengang Elektrotechnik
WS 02/03

AG Betriebssysteme FB3
Kirsten Berkenkötter

Lösung Übungszettel 3

1 Aufgabe 1

```
#include <stdio.h>

int main()
{
   //Variable für die Zahl, die ausgegeben werden sol
   int zahl;

   do
   {
      //einlesen der Zahl
      printf("Gib eine Zahl ein: ");
      scanf("%d", &zahl);

      //Art der Ausgabe entscheidet sich in der Switch-Anweisung
      switch (zahl)
      {
         case 0: break;
         case 1: printf("eins\n");//wenn die Zahl 1 ist, gebe eins aus
            break;
         case 2: printf("zwei\n");//wenn die Zahl 2 ist, gebe zwei aus
            break;
         case 3: printf("drei\n");//wenn die Zahl 3 ist, gebe drei aus
            break;
         case 4: //wenn die Zahl 4 oder 5 ist, gebe viel aus
            //deshalb gehe bei 4 direkt in die Behandlung von 5, kein break!
         case 5: printf("viele\n");
            break;
         default: printf("weiss nicht\n");//bei jeder anderen Nummer, gebe
                        //weiss nicht aus
            break;
      }
   }
   while (zahl != 0);
}
```

2 Aufgabe 2

```
#include <stdio.h>

int main()
{
   //variable für Zahl, die ausgeben werden soll
   int zahl;

   do
   {
      //zahl einlesen
      //nach dem scanf geht es automatisch in die neue Zeile
      printf("Gib eine Zahl ein: ");
      scanf(" %d", &zahl);

      //bei 0 nichts -> Programmabbruch
      if (zahl == 0)
         ;
      //wenn die Zahl 1 ist, gib eins aus
      else if (zahl == 1)
         printf("eins\n");
      //wenn die Zahl 2 ist, gib zwei aus
      else if (zahl == 2)
         printf("zwei\n");
      //wenn die Zahl 3 ist, gib drei aus
      else if (zahl == 3)
         printf("drei\n");
      //wenn die Zahl 4 oder 5 ist, gib viel aus
      else if ((zahl == 4) || (zahl == 5))
         printf("viele\n");
      //bei jeder anderen Zahl gib weiss nicht aus
      else
         printf("weiss nicht\n");
   }
   while (zahl != 0);
}
```

3 Aufgabe 3

Voraussetzungen:

epsilon $\in \mathbf{R}_+$	epsilon ist Element der positiven reellen Zahlen
n $\in \mathbf{Z}_+$	n ist Element der positiven ganzen Zahlen
alt, neu, summand $\in \mathbf{R}_+$	alt, neu, summand sind Elemente der positiven reellen Zahlen

Vorbedingungen:

alt == 1	alterWert ist initialisiert mit 1 (nicht zwingend notwendig!)
neu == 1	neuer Wert der Reihe ist initialisiert mit 1 (erstes Glied)
summand == 1	summand ist initialisiert mit 1 ($\frac{1}{2^0}$)
n == 0	n ist initialisiert mit 0 (erstes Glied der Reihe berechnet)

Nachbedingungen:

$neu == h(n) = 1 + \frac{1}{2^0} + \frac{1}{2^1} + ... + \frac{1}{2^{n-1}} + \frac{1}{2^n}$
$alt == h(n-1) = 1 + \frac{1}{2^0} + \frac{1}{2^1} + ... + \frac{1}{2^{n-1}}$
$summand == \frac{1}{2^n}$
$n == min\{x \in \mathbf{Z}_+ | h(x) - h(x-1) < epsilon\}$

4 Aufgabe 4

1. //Typ definieren
   typedef enum {Januar = 1, Februar, Maerz, April, Mai, Juni, Juli, August, September, Oktober, November, Dezember} monate_t;
   //Variable anlegen und initialisieren
   monate_t mein_monat = Januar;

2. //Typ definieren
   typedef struct
   {
   unsigned int tag;
   monat_t monat;
   unsigned int jahr;
   } datum_t;
   //Variable anlegen und initialisieren
   datum_t semsteranfang = {1, September, 2002};

3. //Typ definieren
   typedef struct
   {
   int reell;
   int imag;
   } komplex_t;
   //Variable anlegen und initialisieren
   komplex_t komplexeZahl= {1, -5};

5 Aufgabe 5

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

//definiere Größe des Arrays
#define MAXNUM 10000

//Funktion für die binäre Suche
//suchzahl ist die gesuchte Zahl, *array ein Zeiger auf das zu
//durchsuchende Array
//laenge gibt die Länge dieses Arrays an, *zaehler ist ein Zeiger auf
//einen Zähler, der die Anzahl der benötigten Vergleiche zählt
int binaereSuche(int suchzahl,  int *array, int laenge, long *zaehler)
{
   //Variable für die obere Grenze, die untere Grenze und die Mitte
   int obereGrenze, untereGrenze, mitte;

   //obere Grenze am Anfang auf dem obersten Index des Arrays
```

```
   //untere Grenze zu Anfang auf 0, unterste Grenze des Arrays
   obereGrenze = laenge-1;
   untereGrenze = 0;

   //nun die binäre Suche
   //solange die obere Grenze größer oder gleich der unteren Grenze ist,
   //ist die binäre Suche gültig
   //ist die obere Grenze kleiner als die untere, wurde das Array erfolglos
   //durchsucht
   while(obereGrenze >= untereGrenze)
   {
      //stelle die Mitte des Teilbereichs fest
      //erhöhe den Zähler für die Anzahl der Vergleiche
      mitte = (obereGrenze + untereGrenze)/2;
      (*zaehler)++;

      //wenn die gesuchte Zahl kleiner als die Zahl in der Arraymitte ist,
      //dann liegt sie in der unteren Hälfte des Teilbereichs
      //der neue Suchbereich ist dann die untere Hälfte, d.h. neue
      //obere Grenze ist mitte-1
      if (suchzahl < array[mitte])
         obereGrenze = mitte-1;
      //wenn die gesuchte Zahl größer als die Zahl in der Arraymitte ist,
      //dann liegt sie in der oberen Hälfte des Teilbereichs
      //der neue Suchbereich ist dann die obere Hälfte, d.h. neue
      //untere Grenze ist mitte+1
      else if (suchzahl > array[mitte])
         untereGrenze = mitte+1;
      //wenn die gesuchte Zahl gleich der Zahl in der Arraymitte ist,
      //habe ich sie gefunden und gebe den Index im Array zurück
      else
         return mitte;
   }

   //wenn die Schleife durchlaufen ist und die Funktion immer noch läuft
   //(keine Rückgabe), dann ist die Zahl nicht im Array enthalten,
   //Rückgabewert in diesem Fall ist -1
   return -1;
}

int main()
{
   //eine Variable für das zu durchsuchende Array
   //eine Variable für die zu suchende Zahl
   //eine Variable für das Ergebnis (Index der zu suchenden Zahl)
   //eine Variable als Index für die for-Schleife
   //eine Variable als Zähler für die benötigten Vergleiche beim Suchen
   int array[MAXNUM];
   int suchzahl, ergebnis, i;
   long zaehler = 0;
```

```
    //belege das int-Array in einer for-Schleife
    //damit ist es auch automatisch sortiert
    for (i = 0; i < MAXNUM; i++)
       array[i] = i;

    //bestimme eine Zufallszahl als zu suchende Zahl
    srand(0);
    suchzahl = rand() % MAXNUM;

    //suche die Zahl, Ergebnis ist Index dieser Zahl im Array,
    //wenn sie vorhanden ist, sonst ist das Ergebnis -1
    ergebnis = binaereSuche(suchzahl, array, MAXNUM, &zaehler);

    //gib das Ergebnis und die Anzahl der benötigen Vergleiche
    //oder eine Fehlermeldung aus
    if (ergebnis == -1)
       printf("Nicht gefunden\n");
    else
    {
       printf("Zahl %d ist an Stelle %d\n", suchzahl, ergebnis);
       printf("Vergleiche %d\n", zaehler);
    }
}
```

6 Aufgabe 6

Voraussetzungen:

suchzahl $\in \mathbf{Z}$	suchzahl ist Element der ganzen Zahlen
array = {x $\mid$ x $\in \mathbf{Z}\}\wedge$	array ist ein Menge von Zahlen x für die gilt: x ist Element Z
laenge = #array	und die Anzahl dieser Elemente ist laenge
zaehler $\in \mathbf{Z}_+$	zaehler ist Element der positiven ganzen Zahlen
obereGrenze, untereGrenze, mitte $\in \mathbf{Z}_+$	obereGrenze, untereGrenze und mitte sind Elemente der positiven ganzen Zahlen

Vorbedingungen:

zaehler == 0	zaehler ist mit 0 initialisiert
obereGrenze == laenge -1	obereGrenze ist zu Anfang laenge des Arrays -1 (oberster Index)
untere Grenze == 0	untereGrenze ist zu Anfang 0 (unterster Index)
$mitte = \frac{obereGrenze+untereGrenze}{2}$	mitte liegt in der Mitte zwischen obererGrenze und untererGrenze

Nachbedingungen:

Fall 1:	wenn obereGrenze ist kleiner untereGrenze,
(obereGrenze < untereGrenze) $\Rightarrow$ (suchzahl $\notin$ array)	dann ist die Suchzahl nicht Element des Arrays
Fall 2:	wenn obereGrenze größer od. gleich untereGrenze,
(obereGrenze > untereGrenze) $\Rightarrow$ (suchzahl $\in$ array)	dann ist die Suchzahl Element des Arrays und
$\wedge$ (a[mitte] == suchzahl)	a[mitte] ist die gesuchte Zahl
$\wedge$ obereGrenze $\geq$ mitte $\geq$ untereGrenze	und obereGrenze ist größer oder gleich mitte ist größer oder gleich untereGrenze
In beiden Fällen: zaehler $\leq \log_2 laenge$	zaehler ist in beiden Fällen maximal $\log_2 laenge$