Inhaltsverzeichnis

1 Variablen in Java und in C

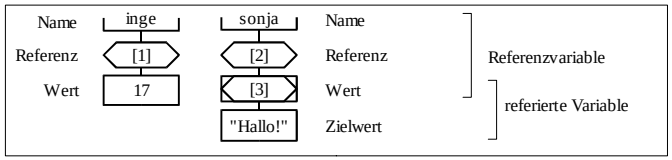
1.1 Primitive Variablen und Referenzvariablen in Java

In Java unterscheidet man **primitive Typen** (`int`, `float`, ...) und **Referenztypen** (Klassentypen wie `String` und `Vector` etc., Reihungstypen wie `int[]` und `String[]` etc.). Variablen dieser Typen bezeichnet man entsprechend als **primitive Variablen** bzw. **Referenzvariablen**.

Beispiel: Variablen in Java

```
1   int    inge  = 17;                   // Eine primitive Variablen
2   String sonja = new String("Hallo!"); // Eine Referenzvariable
```

Als Bojen dargestellt:



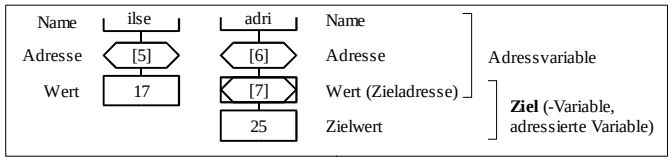
1.2 Adresstypen in C (ähneln den Referenztypen in Java)

In C gibt es zu jedem Typ `T` einen **Adresstyp** `T*` (Lies: Adresse von T), z.B. `int*` (Adresse von int), `float*` (Adresse von float), `int**`(Adresse von Adresse von int), ... etc.

Beispiel: Variablen in C

```
3   int   ilse = 17;                          // Eine "normale" Variable
4   int * adri = (int *) malloc(sizeof(int)); // Eine Adressvariable
5       * adri = 25; // Die Zielvariable *adri von adri initialisieren
```

Als Bojen dargestellt:



1.3 Der Adressoperator &

Darf auf beliebige Variablen angewendet werden. Liefert die **Adresse** der Variablen.

Beispiele: `&ilse` ("die Adresse der Variablen `ilse`") ist gleich `[5]`, `&adri` ist gleich `[6]`.

1.4 Der Zieloperator *

Darf nur auf Adressvariablen angewendet werden. Liefert das **Ziel** der Variablen.

Beispiel: `*adri` ("das Ziel der Adressvariablen `adri`") ist die Variable mit der Adresse `[7]` und dem Wert `25`. `&*adri` ("die Adresse der Variablen `*adri`") ist gleich [7].

2 C-Variablen als Bojen darstellen

2.1 Variablen der Typen int, float, int* und float* als Bojen darstellen

Zur Erinnerung: In Java besteht eine **Variable** aus einer **Referenz** und einem **Wert**. Zusätzlich kann eine Variable einen **Namen** und/oder einen **Zielwert** besitzen. Man unterscheider **primitive Typen** (z.B. `int`, `float` etc.) und **Referenztypen** (z.B. Klassentypen wie `String` oder `Vector`, Reihungstypen wie `int[]` oder `String[]`). Der Wert einer primitiven Variablen ist ein **primitiver Wert**. Der Wert einer Referenzvariablen ist eine **Referenz**.

In C besteht eine Variable aus einer **Adresse** und einem **Wert**. Zusätzlich kann eine Variable einen **Namen** und/oder ein **Ziel** haben. In C unterscheidet man **"normale" Typen** (z.B. die vordefinierten Typen `int` und `char` und die vom Programmierer definierten Typen, etwa den Verbundtyp `Person` oder den Aufzählungstyp `Farbe` etc.) und **Adresstypen** (deren Namen mit einem Stern '`*`' enden, z.B. `int *`, `char *`, `Person *`, `Farbe *` etc.). Der Wert einer "normalen" Variablen ist ein **normaler Wert** (ein `int`-Wert oder ein `char`-Wert oder ein `Person`-Wert oder ein `Farbe`-Wert etc.), der Wert einer Adressvariablen ist eine **Adresse**.

Wenn man den **Adressoperator** & auf eine beliebige Variable anwendet, erhält man die **Adresse** dieser Variablen. Wenn man den **Zieloperator** * auf eine Adressvariable anwendet, erhält man das **Ziel** der Variablen. Das Ziel einer Adressvariablen ist eine Variable.

In Java ist es **nicht** möglich, die Referenz einer Variablen in einen Wert eines Ganzzahltyps umzuwandeln und zu bearbeiten (z.B. auszudrucken). Dagegen kann man in C **alle** Teile einer Variablen (die Adresse, den Wert und das Ziel, falls vorhanden) bearbeiten und ausgeben, wie das folgende Programm verdeutlichen soll:

```
6   // Datei Bojen01.c
7   /* ------------------------------------------------------
8    * Gibt Variablen der Typen int, float, int*, und float*
9    * als "querliegende Bojen" aus.
10  * ------------------------------------------------------ */
11  #include    <stdio.h> // printf, scanf, sprintf
12  //-------------------------------------------------------------
```

```
13 void bojeInt  (int   * adr, char * name) {
14    // Gibt die int-Variabl *adr als querliegende Boje aus.
15    printf("|%s|--<%X>--[%+d]\n", name, adr, *adr);
16 } // bojeInt
17 //-----------------------------------------------------------------
18 void bojeFloat(float * adr, char * name) {
19    // Gibt die float-Variabl *adr als querliegende Boje aus.
20    printf("|%s|--<%X>--[%+3.1f]\n", name, adr, *adr);
21 } // bojeFloat
22 //-----------------------------------------------------------------
23 void bojeIntAdr  (int   * * adr, char * name) {
24    // Gibt die int*-Variable *adr als querliegende Boje aus.
25    printf("|%s|--<%X>--[<%X>]--[%+d]\n",    name, adr, *adr, **adr);
26 } // bojeIntAdr
27 //-----------------------------------------------------------------
28 void bojeFloatAdr(float * * adr, char * name) {
29    // Gibt die float*-Variable *adr als querliegende Boje aus.
30    printf("|%s|--<%X>--[<%X>]--[%+3.1f]\n", name, adr, *adr, **adr);
31 } // bojeFloatAdr
32 //-----------------------------------------------------------------
33 int main() {
34    int     anna = 123;
35    int     bert = -45;
36
37    float   carl = 1.5;
38    float   dora = 2.3;
39
40    int   * emma = &anna;
41    int   * fred = &bert;
42
43    float * gert = &carl;
44    float * hana = &dora;
45
46    printf("Bojen01: Jetzt geht es los!\n");
47    printf("-----------------------------------------------------\n");
48    printf("A int-Variablen    als querliegende Bojen:\n");
49    printf("B "); bojeInt      (&anna, "anna");
50    printf("C "); bojeInt      (&bert, "bert");
51    printf("-----------------------------------------------------\n");
52    printf("D float-Variablen  als querliegende Bojen:\n");
53    printf("E "); bojeFloat    (&carl, "carl");
54    printf("F "); bojeFloat    (&dora, "dora");
55    printf("-----------------------------------------------------\n");
56    printf("G int*-Variablen   als querliegende Bojen:\n");
57    printf("H "); bojeIntAdr   (&emma, "emma");
58    printf("I "); bojeIntAdr   (&fred, "fred");
59    printf("-----------------------------------------------------\n");
60    printf("J float*-Variablen als querliegende Bojen:\n");
61    printf("K "); bojeFloatAdr (&gert, "gert");
62    printf("L "); bojeFloatAdr (&hana, "hana");
63    printf("-----------------------------------------------------\n");
64    printf("Bojen01: Das war's erstmal!\n");
65    return 0;
66 } // main
67 /* ------------------------------------------------------
68 Ausgabe des Programms Bojen01:
69
70 Bojen01: Jetzt geht es los!
71 -----------------------------------------------------
72 A int-Variablen    als querliegende Bojen:
73 B |anna|--<22FEEC>--[+123]
74 C |bert|--<22FEE8>--[-45]
```



```
75 -----------------------------------------------------
76 D float-Variablen  als querliegende Bojen:
77 E |carl|--<22FEE4>--[+1.5]
78 F |dora|--<22FEE0>--[+2.3]
79 -----------------------------------------------------
80 G int*-Variablen   als querliegende Bojen:
81 H |emma|--<22FEDC>--[<22FEEC>]--[+123]
82 I |fred|--<22FED8>--[<22FEE8>]--[-45]
83 -----------------------------------------------------
84 J float*-Variablen als querliegende Bojen:
85 K |gert|--<22FED4>--[<22FEE4>]--[+1.5]
86 L |hana|--<22FED0>--[<22FEE0>]--[+2.3]
87 -----------------------------------------------------
88 Bojen01: Das war's erstmal!
89 ------------------------------------------------------ */
```

2.2 In C gibt es jeden Typ als "primitven Typ" und als "Referenztyp"

In C gibt es zu jedem Typ `T` einen Adresstyp `T *` (lies: "Adresse von T"). Diese **Adresstypen** in C entsprechen weitgehend den **Referenztypen** in Java, wie das folgende Programm verdeutlichen soll.

```
1  // Datei Bojen02.c
2  /* ------------------------------------------------------------
3   * In Java unterscheidet man zwischen primitven Typen und
4   * Referenztypen. In C gibt es sozusagen "von jedem Typ eine
5   * primitive Version und eine Referenz-Version", etwa so:
6   *
7   * "primitve Version"   "Referenz-Version"
8   *  int                  int *
9   *  Person               Person *
10  * ------------------------------------------------------------ */
11 #include    <stdio.h> // printf, scanf, sprintf
12 #include   <stdlib.h> // atof, atoi, atol, rand, malloc
13 //-----------------------------------------------------------------
14 typedef struct {
15    unsigned short jahre;
16    unsigned short zentim;
17    float          kilo;
18 } Person;
19 //-----------------------------------------------------------------
20 void bojeInt(int * adr, char * name) {
21    // Gibt die int-Variable *adr als Boje aus.
22    printf("|%s|--<%08X>--|%d|\n", name, adr, *adr);
23 } // bojeInt
24 //-----------------------------------------------------------------
25 void bojeIntAdr(int * * adr, char * name) {
26    // Gibt die int*-Variable *adr als Boje aus.
27    printf("|%s|--<%08X>--|<%08X>|--|%d|\n",
28          name, adr, *adr, **adr);
29 } // bojeIntAdr
30 //-----------------------------------------------------------------
31 void bojePerson(Person * adr, char * name) {
32    // Gibt die Person-Variable *adr als Boje aus.
33    printf("|%s|-----<%08X>\n", name, adr);
34    printf("    |jahre |-<%08X>-|%3d|\n", &(adr->jahre),  adr->jahre);
35    printf("    |zentim|-<%08X>-|%3d|\n", &(adr->zentim), adr->zentim);
36    printf("    |kilo  |-<%08X>-|%3d|\n", &(adr->kilo),   adr->kilo);
37 } // bojePerson
38 //-----------------------------------------------------------------
39 #define B13 "             " // 13 Blanks
40 void bojePersonAdr(Person * * adr, char * name) {
```

```
41    // Gibt die Person*-Variable *adr als Boje aus:
42    Person * adr1 = *adr;
43    printf("|%s|--<%08X>--|<%08X>|\n", name, adr, *adr);
44    printf(B13 "|jahre |-<%08X>-|%3d|\n", &(adr1->jahre),  adr1->jahre);
45    printf(B13 "|zentim|-<%08X>-|%3d|\n", &(adr1->zentim), adr1->zentim);
46    printf(B13 "|kilo  |-<%08X>-|%3d|\n", &(adr1->kilo),   adr1->kilo);
47 } // bojePersonAdr
48 //------------------------------------------------------------
49 int main() {
50    int      anna  = 17;
51    int    * bert  = (int *) malloc(sizeof(int));
52    *bert          =  25;
53
54    Person   carl  = {19, 175, 72};
55    Person * dora  = (Person *) malloc(sizeof(Person));
56    (*dora).jahre  =  23;   // oder: dora->jahre  =  23;
57    (*dora).zentim = 178;   // oder: dora->zentim = 178;
58    (*dora).kilo   =  75;   // oder: dora->kilo   =  75;
59
60    printf("Bojen02: Jetzt geht es los!\n");
61    printf("----------------------------------------\n");
62    printf("A "); bojeInt      (&anna, " anna");
63    printf("B "); bojeInt      ( bert, "*bert");
64    printf("C "); bojeIntAdr   (&bert, " bert");
65    printf("----------------------------------------\n");
66    printf("D "); bojePerson   (&carl, " carl");
67    printf("E "); bojePerson   ( dora, "*dora");
68    printf("F "); bojePersonAdr(&dora, " dora");
69    printf("----------------------------------------\n");
70    printf("Bojen02: Das war's erstmal!\n");
71    return 0;
72 } // main
73 /* ------------------------------------------------------------
74 Ausgabe des Programms Bojen02:
75
76 Bojen02: Jetzt geht es los!
77 ----------------------------------------
78 A | anna|--<0022FEEC>--|17|
79 B |*bert|--<0A040428>--|25|
80 C | bert|--<0022FEE8>--|<0A040428>|--|25|
81 ----------------------------------------
82 D | carl|-----<0022FEE0>
83      |jahre |-<0022FEE0>-| 19|
84      |zentim|-<0022FEE2>-|175|
85      |kilo  |-<0022FEE4>-|  0|
86 E |*dora|-----<0A040438>
87      |jahre |-<0A040438>-| 23|
88      |zentim|-<0A04043A>-|178|
89      |kilo  |-<0A04043C>-|  0|
90 F | dora|--<0022FEDC>--|<0A040438>|
91                |jahre |-<0A040438>-| 23|
92                |zentim|-<0A04043A>-|178|
93                |kilo  |-<0A04043C>-|  0|
94 ----------------------------------------
95 Bojen02: Das war's erstmal!
96 ------------------------------------------------------------ */
```

2.3 Eine Reihung ist eine konstante Adresse ihrer ersten Komponenten

In Java ist eine Reihung ein Objekt, welches (unter anderem) die **Komponenten** der Reihung enthält und zusätzlich ein Attribut namens `length` (und evtl. weitere, nicht-öffentliche Elemente).

In C besteht eine Reihung nur aus ihren Komponenten. Eine Reihung von **drei** `int`-Komponenten belegt genau dreimal soviel Speicherplatz wie **eine** `int`-Variable. Der Name einer Reihung (z.B. `ir`) steht für die Adresse der ersten Reihungskomponenten (die man auch durch `&ir[0]` bezeichnen kann).

```
1  // Datei Bojen03.c
2  /* ------------------------------------------------------------
3   * Reihungen haben grosse Aehnlichkeit mit konstanten Adressen.
4   * ------------------------------------------------------------ */
5  #include    <stdio.h> // printf, scanf, sprintf
6  #include   <stdlib.h> // atof, atoi, atol, rand, malloc
7
8  #define LEN1 3
9  #define LEN2 5
10 // ----------------------------------------------------------------------
11 void drucke01(int ir[], int len) {
12    // Verlaesst sich darauf, dass ir eine int-Reihung der Laenge len
13    // bezeichnet. Gibt diese Reihung in lesbarer Form aus.
14    int i;
15    printf("drucke01: Eine Reihung mit %d Komponenten: ", len);
16    for (i=0; i<len; i++) {
17       printf("%d ", ir[i]);
18    }
19    printf("\n");
20 } // drucke01
21 // ----------------------------------------------------------------------
22 void drucke02(int * const ia, int len) {
23    // Verlaesst sich darauf, dass ia die Adresse einer int-Reihung der
24    // Laenge len ist. Gibt diese Reihung in lesbarer Form aus.
25    int i;
26    printf("drucke02: Eine Reihung mit %d Komponenten: ", len);
27    for (i=0; i<len; i++) {
28       printf("%d ", ia[i]);
29    }
30    printf("\n");
31 } // drucke02
32 // ----------------------------------------------------------------------
33 void drucke03(int ir[LEN1]) {
34    // Verlaesst sich darauf, dass ir eine int-Reihung der Laenge len
35    // bezeichnet. Gibt diese Reihung in lesbarer Form aus.
36    int i;
37    printf("drucke03: Eine Reihung mit 3 Komponenten: ");
38    for (i=0; i<LEN1; i++) {
39       printf("%d ", ir[i]);
40    }
41    printf("\n");
42 } // drucke03
43 // ----------------------------------------------------------------------
44 void drucke04(int * ia, int len) {
45    // Verlaesst sich darauf, dass ia die Adresse einer int-Reihung der
46    // Laenge len ist. Gibt diese Reihung in lesbarer Form aus.
47    int i;
48    printf("drucke04: Eine Reihung mit %d Komponenten: ", len);
49    for (i=0; i<len; i++) {
50       printf("%d ", *ia++);
51    }
52    printf("\n");
53 } // drucke04
54 // ----------------------------------------------------------------------
55 int main() {
56    // Zwei int-Reihungen der Laenge LEN1:
57    int         ir01[LEN1] = {11, 12, 13};
58    int * const ai01       = (int *) calloc(LEN1, sizeof(int));
```

```
59
60      // Eine int-Reihung der Laenge LEN2:
61      // (ir02 und ai02 bezeichnen beide dieselbe int-Reihung)
62      int         ir02[LEN2] = {31, 32, 33, 34, 35};
63      int * const ai02       = ir02;
64
65      // calloc hat ai01 nur mit Nullen initialisiert:
66      ai01[0] = 21;
67      ai01[1] = 22;
68      ai01[2] = 23;
69
70      printf("Bojen03: Jetzt geht es los!\n");
71      printf("-------------------------------------------------\n");
72      printf("A "); drucke01(ir01, LEN1);
73      printf("B "); drucke02(ir01, LEN1);
74      printf("C "); drucke03(ir01);
75      printf("D "); drucke04(ir01, LEN1);
76      printf("-------------------------------------------------\n");
77      printf("E "); drucke01(ai01, LEN1);
78      printf("F "); drucke02(ai01, LEN1);
79      printf("G "); drucke03(ai01);
80      printf("H "); drucke04(ai01, LEN1);
81      printf("-------------------------------------------------\n");
82      printf("I "); drucke03(ir02);
83      printf("J "); drucke03(ai02);
84      printf("-------------------------------------------------\n");
85      printf("Bojen03: Das war's erstmal!\n");
86      return 0;
87  } // main
88  /* ---------------------------------------------------------------------------
89  Ausgabe des Programms Bojen03:
90
91  Bojen03: Jetzt geht es los!
92  -------------------------------------------------
93  A drucke01: Eine Reihung mit 3 Komponenten: 11 12 13
94  B drucke02: Eine Reihung mit 3 Komponenten: 11 12 13
95  C drucke03: Eine Reihung mit 3 Komponenten: 11 12 13
96  D drucke04: Eine Reihung mit 3 Komponenten: 11 12 13
97  -------------------------------------------------
98  E drucke01: Eine Reihung mit 3 Komponenten: 21 22 23
99  F drucke02: Eine Reihung mit 3 Komponenten: 21 22 23
100 G drucke03: Eine Reihung mit 3 Komponenten: 21 22 23
101 H drucke04: Eine Reihung mit 3 Komponenten: 21 22 23
102 -------------------------------------------------
103 I drucke03: Eine Reihung mit 3 Komponenten: 31 32 33
104 J drucke03: Eine Reihung mit 3 Komponenten: 31 32 33
105 -------------------------------------------------
106 Bojen03: Das war's erstmal!
107 ------------------------------------------------------------------------ */
```

2.4 Reihungen und der Adressoperator &

Als Besispiel wird hier eine float-Reihung namens fr betrachtet:

```
1   float fr[3] = {1.5, 3.7, 2.3};
```

In C ist `fr` eigentlich der Name einer **Konstanten** vom Typ **Adresse von float**, die die erste Komponente der Reihung adressiert (im Beispiel ist das die Komponente `fr[0]` mit dem Wert `1.5`). Da Konstanten keine Adressen haben (d.h. nicht unbedingt an **einer** bestimmten, **adressierbaren** Stelle des Speichers stehen), ist es eigentlich nicht sinnvoll, den Adressoperator & auf den Reihungsnamen

wie `fr` anzuwenden. In C ist ein socher "Missbrauch" des Adressoperators aber trotzdem erlaubt, möglicherweise aus folgendem Grund:

Will man mit der Funktion `scanf` z.B. einen `int`-Wert einlesen, muss man als Parameter die **Adresse** einer `int`-Variablen angeben, etwa so:

```
2   int eingabe;
3   scanf("%d", &eingabe);
```

Ein immer wieder gern gemachter Fehler besteht darin, den Adressoperator & vor dem Variablennamen `eingabe` zu vergessen. In diesem Fall erkennt der Ausführer zwar (leider!) keinen formalen Fehler, aber die Funktion `scanf` funktioniert dann nicht.

Will man mit der Funktion `scanf` einen String einlesen, muss man als Parameter die Adresse eines geeigneten Speicherbereichs angeben, etwa so:

```
4   char puffer[100];
5   scanf("%s", puffer);
```

Ein immer wieder gern gemachter Fehler bestand früher darin, vor dem Reihungsnamen `puffer` einen Adressoperator & anzugeben, obwohl der Name `puffer` (als Name einer **Adress**konstanten) bereits eine Adresse bezeichnet. Vermutlich haben sich viele Programmierer angewöhnt, in Aufrufen der Funktion `scanf` **jeden** Parameter (ausser den ersten) mit einem Adressoperator & zu versehen. Möglicherweise ist das der Grund dafür, dass es seit dem C89-Standard offiziell erlaubt ist, in Zeile 5 wahlweise `puffer` oder `&puffer` anzugeben. In beiden Fällen muss die `scanf`-Funktion denselben Effekt haben, um dem Standard zu entsprechen.

Allgemein gelten für den **Adressoperator** & und **Reihungen** wie `fr` bestimmte Regeln, die durch das folgende Programm `Bojen04` illustriert werden sollen (und einigen Programmiereren ziemlich abstrus vorkommen). Glücklicherweise kann man viele C-Programme schreiben, ohne sich mit diesen Regeln zu befassen.

```
1   // Datei Bojen04.c
2   /* ---------------------------------------------------------------------------
3   Dieses Programm solle zeigen: Wenn fr eine Reihung ist (z.B. mit float-
4   Komponenten), dann haben die Ausdruecke fr und &fr gleiche Werte (z.B.
5   0x22FED8), gehoeren aber zu verschidenen Typen:
6    fr gehoert zum Typ Reihung von float (bzw. Adr. von float)
7   &fr gehoert zum Typ Adr. von Reihung von float
8   ------------------------------------------------------------------------ */
9   #include    <stdio.h> // printf, scanf, sprintf
10  #include   <stdlib.h> // atof, atoi, atol, rand, malloc
11
12  int main() {
13
14     // Umwandlungsbuchstaben etc. fuer printf:
15     #define HEX "%X"
16     #define FLT "%6.4f"
17
18     // Eine Reihung namens fr von float-Variablen:
19     float fr[3]    = {1.5, 3.7, 2.3};
20
21     // Zu welchen Typen gehoeren die Ausdruecke fr und &fr?
22     // Aus den Warnungen der Compiler folgt: fr gehoert zum Typ
23     // Adr. von float und &fr gehoert zum Typ Adr. von Reihung von float:
24     float  * a1    =  fr; // a1 ist vom Typ Adr.             von float
25     float  * a2    = &fr; // a2 ist vom Typ Adr.             von float
26     float (* a3)[] =  fr; // a3 ist vom Typ Adr. von Reihung von float
```

```
27    float (* a4)[] = &fr; // a4 ist vom Typ Adr. von Reihung von float
28
29    printf("Bojen04: Jetzt geht es los!\n");
30    printf("-------------------------------------------------+\n");
31    if (fr == &fr) {
32    printf("A|fr und &fr bezeichnen gleiche Werte!            |\n");
33    }
34    printf("-+-----+------+---------------------------------+\n");
35    printf(" | Aus | Wert | Typ des Aus (-drucks):           |\n");
36    printf("-+-----+------+---------------------------------+\n");
37    printf("B| &fr:|" HEX"|         Adr. von Reihung von float|\n", &fr);
38    printf("C|  fr:|" HEX"|                 Reihung von float|\n",  fr);
39    printf(" |     |      |    bzw. Adr.             von float|\n");
40    printf("D| *fr:|" FLT"|                          float|\n", *fr);
41    printf("-+-----+------+---------------------------------+\n");
42    printf(" | Aus | Wert | Typ des Aus (-drucks):           |\n");
43    printf("-+-----+------+---------------------------------+\n");
44    printf("E| &a4:|" HEX"|Adr. von Adr. von Reihung von float|\n",  &a4);
45    printf("F|  a4:|" HEX"|         Adr. von Reihung von float|\n",   a4);
46    printf("G| *a4:|" HEX"|                 Reihung von float|\n",  *a4);
47    printf(" |     |      |    bzw. Adr.             von float|\n");
48    printf("H|**a4:|" FLT"|                          float|\n", **a4);
49    printf("-+-----+------+---------------------------------+\n");
50    printf(" |fr als Konstante (ohne Adresse &fr) dargestellt: \n");
51    printf("I|fr|--------------------[<"HEX">]-["FLT"]\n", fr, *fr);
52    printf(" |fr als Variable  (mit  Adresse &fr) dargestellt: \n");
53    printf("J|fr|-----------<" HEX">--[<"HEX">]-["FLT"]\n", &fr,   fr, *  fr);
54    printf("K|fr|-----------<" HEX">--[<"HEX">]-["FLT"]\n", &fr, *&fr, **&fr);
55    printf(" |a4 als Variable  (mit  Adresse &a4) dargestellt: \n");
56    printf("L|a4|-<" HEX">-[<" HEX">]-[<"HEX">]-["FLT"]\n", &a4, a4, *a4, **a4);
57    printf("-+-------------------------------------------------\n");
58    printf("Bojen04: Das war's erstmal!\n");
59    return 0;
60 } // main
61 /* ------------------------------------------------------------
62 Meldungen des C-Compilers gcc von Gnu:
63
64 Bojen04.c:25: warning: initialization from incompatible pointer type
65 Bojen04.c:26: warning: initialization from incompatible pointer type
66 Bojen04.c:32: warning: comparison of distinct pointer types lacks a cast
67 ------------------------------------------------------------
68 Meldungen des C-Compilers bcc32 von Borland:
69
70 Warning W8075 Bojen04.c 25: Suspicious pointer conversion in function main
71 Warning W8075 Bojen04.c 26: Suspicious pointer conversion in function main
72 Warning W8011 Bojen04.c 32: Nonportable pointer comparison in function main
73 ------------------------------------------------------------
74 Meldungen des C-Compilers lcc-win32:
75
76 Warning bojen04.c: 25  assignment of pointer to float to
77                                  pointer to array 3 of float
78 Warning bojen04.c: 26  assignment of pointer to incomplete array of float to
79                                  pointer to float
80 Error bojen04.c: 32  operands of == have illegal types 'pointer to float'
81 and 'pointer to array 3 of float'
82 ------------------------------------------------------------
83
```



```
84  Ausgabe des Programms Bojen04 (bei allen drei
85  Compilern fast gleich, nur unterschiedliche Adressen):
86
87  Bojen04: Jetzt geht es los!
88  -------------------------------------------------+
89  A|fr und &fr bezeichnen gleiche Werte!            |
90  -+-----+------+---------------------------------+
91   | Aus | Wert | Typ des Aus (-drucks):           |
92  -+-----+------+---------------------------------+
93  B| &fr:|22FED8|         Adr. von Reihung von float|
94  C|  fr:|22FED8|                 Reihung von float|
95   |     |      |    bzw. Adr.             von float|
96  D| *fr:|1.5000|                          float|
97  -+-----+------+---------------------------------+
98   | Aus | Wert | Typ des Aus (-drucks):           |
99  -+-----+------+---------------------------------+
100 E| &a4:|22FEC8|Adr. von Adr. von Reihung von float|
101 F|  a4:|22FED8|         Adr. von Reihung von float|
102 G| *a4:|22FED8|                 Reihung von float|
103  |     |      |    bzw. Adr.             von float|
104 H|**a4:|1.5000|                          float|
105 -+-----+------+---------------------------------+
106  |fr als Konstante (ohne Adresse &fr) dargestellt:
107 I|fr|--------------------[<22FED8>]-[1.5000]
108  |fr als Variable  (mit  Adresse &fr) dargestellt:
109 J|fr|-----------<22FED8>--[<22FED8>]-[1.5000]
110 K|fr|-----------<22FED8>--[<22FED8>]-[1.5000]
111  |a4 als Variable  (mit  Adresse &a4) dargestellt:
112 L|a4|-<22FEC8>-[<22FED8>]-[<22FED8>]-[1.5000]
113 -+-------------------------------------------------
114 Bojen04: Das war's erstmal!
115 ------------------------------------------------------------ */
```

3 Eine sinnvolle Anwendung von Adressen von Adressen von Adressen

Ein Programmmodul für die Verwaltung von Meldungen enthält jeden Meldungstext in verschiedenen Sprachen (z.B. auf Deutsch und auf Englisch). Die anderen Module des Programms greifen nur über eine Adressvariable namens `m_as` ("Meldungen in der aktuellen Sprache") auf die einzelnen Meldungen zu. Dadurch ist es möglich, die Sprache der Meldungen (Deutsch bzw. Englisch) leicht zu verändern (auch "dynamisch", während einer Programmausführung), indem man nur den Wert der Variablen `m_as` ändert. Das Unterprogramm `gibMeldungenInDerAktuellenSpracheAus` soll einen typischen Benutzer dieses Moduls repräsentieren (und das Unterprogramm `gibMeldungenInAllenSprachenAus` einen untypischen Benutzer).

Einige Unterprogramme (`gibEinPaarAdressVariablenAus`, `gibEinPaarAdressAusdrueckeAus` und weitere) sollen das Verstehen von Adressvariablen erleichtern, sind aber kein wichtiger Bestandteil des Moduls zur Verwaltung von Meldungen.

3.1 Das Programm Bojen06

```
1   // Datei Bojen06.c
2   //------------------------------------------------------------
3   #include    <stdio.h> // printf, scanf, sprintf
4   #include   <stdlib.h> // atof, atoi, atol, rand, malloc
5   //------------------------------------------------------------
6   #define ANZ_MELDUNGEN 3
7   #define ANZ_SPRACHEN  2
8   #define DE 0 // Sprache Deutsch
9   #define EN 1 // Sprache Englisch
```

```
10 #define M1 0 // Meldung 1
11 #define M2 1 // Meldung 2
12 #define M3 2 // Meldung 3
13 //------------------------------------------------------------
14 // Die einzelnen Meldungen auf Deutsch:
15 char * m_de_01 = "Bitte geben Sie eine Ganzzahl ein:";
16 char * m_de_02 = "Ihre Eingabe war nicht zulaessig!";
17 char * m_de_03 = "Hier ist das Dreifache Ihrer Eingabe:";
18
19 // Die einzelnen Meldungen auf Englisch:
20 char * m_en_01 = "Please enter an integer:";
21 char * m_en_02 = "Your input was not correct!";
22 char * m_en_03 = "Here comes your input times three:";
23
24 char * * m_de; // Alle Meldungen auf Deutsch
25 char * * m_en; // Alle Meldungen auf Englisch
26 char * * m_as; // Alle Meldungen in der aktuellen Sprache
27
28 char * * * m;  // Alle Meldungen in allen Sprachen
29 //------------------------------------------------------------
30 void init(void) {
31    // Initialisiert alle Adressvariablen.
32    m          = (char * * *) malloc(sizeof(char * *) * ANZ_SPRACHEN);
33    m_de       = (char * *  ) malloc(sizeof(char *  ) * ANZ_MELDUNGEN);
34    m_en       = (char * *  ) malloc(sizeof(char *  ) * ANZ_MELDUNGEN);
35
36    *(m_de+M1) = m_de_01;
37    *(m_de+M2) = m_de_02;
38    *(m_de+M3) = m_de_03;
39
40    *(m_en+M1) = m_en_01;
41    *(m_en+M2) = m_en_02;
42    *(m_en+M3) = m_en_03;
43
44    *(m+DE)    = m_de;
45    *(m+EN)    = m_en;
46 } // init
47 //------------------------------------------------------------
48 void gibMeldungenInDerAktuellenSpracheAus(void) {
49    printf("A %s\n", *(m_as+M1));
50    printf("B %s\n", *(m_as+M2));
51    printf("C %s\n", *(m_as+M1));
52    printf("D %s\n", *(m_as+M3));
53 } // gibMeldungenInDerAktuellenSpracheAus
54 //------------------------------------------------------------
55 void gibMeldungenInAllenSprachenAus(void) {
56    printf("E %s\n", *(*(m+DE)+M1));
57    printf("F %s\n", *(*(m+EN)+M1));
58    printf("G %s\n", *(*(m+DE)+M3));
59    printf("H %s\n", *(*(m+EN)+M3));
60 } // gibMeldungenInAllenSprachenAus
61 //------------------------------------------------------------
62 void gibAdrVar3Aus(char **** adr, char * name) {
63    // Erwartet die Adresse einer char***-Variablen (d.h.
64    // einen char****-Wert) und den Namen der Variablen als
65    // Parameter. Gibt die betreffende char***-Variable als
66    // Boje aus.
67    printf("|%-7s|--<%X>--|<%X>|--|<%X>|--|<%X>|--|%c|\n",
68           name,   adr,  *adr,  **adr, ***adr, ****adr);
69 } // gibAdrVar3Aus
70 //------------------------------------------------------------
71 void gibAdrVar2Aus(char *** adr, char * name) {
```



```
72    // Ebenso wie gibAdrVar3Aus, aber fuer char**-Variablen.
73    printf("|%-7s|--<%X>--|<%X>|--|<%X>|--|%c|\n",
74           name,   adr,  *adr,  **adr, ***adr);
75 } // gibAdrVar2Aus
76 //------------------------------------------------------------
77 void gibAdrVar1Aus(char ** adr, char * name) {
78    // Ebenso wie gibAdrVar3Aus, aber fuer char*-Variablen.
79    printf("|%-7s|--<%X>--|<%X>|--|%c|\n",
80           name,   adr,  *adr,  **adr);
81 } // gibAdrVar1Aus
82 //------------------------------------------------------------
83 void gibAdr3Aus(char *** adr, char * ausdruck) {
84    // Erwartet den char***-Wert eines Ausdrucks und den
85    // Ausdruck selbst als Parameter. Gibt den Ausdruck,
86    // seinen Wert und "alles was daran haengt" aus.
87    printf("%-14s: <%X>--|<%X>|--|<%X>|--|%c|\n",
88         ausdruck, adr,   *adr,  **adr, ***adr);
89 } // gibAdr3Aus
90 //------------------------------------------------------------
91 void gibAdr2Aus(char ** adr, char * ausdruck) {
92    // Ebenso wie gibAdr3Aus, aber fuer char**-Ausdruecke.
93    printf("%-14s: <%X>--|<%X>|--|%c|\n",
94         ausdruck, adr,   *adr, **adr);
95 } // gibAdr2Aus
96 //------------------------------------------------------------
97 void gibAdr1Aus(char * adr, char * ausdruck) {
98    // Ebenso wie gibAdr3Aus, aber fuer char*-Ausdruecke.
99    printf("%-14s: <%X>--|%c|\n",
100        ausdruck, adr,  *adr);
101 } // gibAdr2Aus
102 //------------------------------------------------------------
103 void gibEinPaarAdressVariablenAus(void) {
104    printf("Ein paar Adressvariablen als Bojen dargestellt:\n");
105    gibAdrVar3Aus(&m,       "m");
106    gibAdrVar2Aus(&m_de,    "m_de");
107    gibAdrVar2Aus(&m_en,    "m_en");
108    printf("-------------------------------------------------\n");
109    gibAdrVar1Aus(&m_de_01, "m_de_01");
110    gibAdrVar1Aus(&m_de_02, "m_de_02");
111    gibAdrVar1Aus(&m_de_03, "m_de_03");
112    printf("-------------------------------------------------\n");
113    gibAdrVar1Aus(&m_en_01, "m_en_01");
114    gibAdrVar1Aus(&m_en_02, "m_en_02");
115    gibAdrVar1Aus(&m_en_03, "m_en_03");
116 } // gibEinPaarAdressVariablenAus
117 //------------------------------------------------------------
118 void gibEinPaarAdressAusdrueckeAus(void) {
119    printf("Ein paar Adressausdruecke und ihre Werte etc.:\n");
120    gibAdr3Aus(m+DE,          "    m+DE");
121    gibAdr3Aus(m+EN,          "    m+EN");
122    printf("-------------------------------------------------\n");
123    gibAdr2Aus(*(m+DE)+M1,    "  *(m+DE)+M1");
124    gibAdr2Aus(*(m+DE)+M2,    "  *(m+DE)+M2");
125    gibAdr2Aus(*(m+DE)+M3,    "  *(m+DE)+M3");
126    printf("-------------------------------------------------\n");
127    gibAdr2Aus(*(m+EN)+M1,    "  *(m+EN)+M1");
128    gibAdr2Aus(*(m+EN)+M2,    "  *(m+EN)+M2");
129    gibAdr2Aus(*(m+EN)+M3,    "  *(m+EN)+M3");
130    printf("-------------------------------------------------\n");
131    gibAdr1Aus(*(*(m+DE)+M1), "*(*(m+DE)+M1)");
132    gibAdr1Aus(*(*(m+DE)+M2), "*(*(m+DE)+M2)");
133    gibAdr1Aus(*(*(m+DE)+M3), "*(*(m+DE)+M3)");
```

```
134    printf("-----------------------------------------------\n");
135    gibAdr1Aus(*(*(m+EN)+M1),  "*(*(m+EN)+M1)");
136    gibAdr1Aus(*(*(m+EN)+M2),  "*(*(m+EN)+M2)");
137    gibAdr1Aus(*(*(m+EN)+M3),  "*(*(m+EN)+M3)");
138 } // gibEinPaarAdressAusdrueckeAus
139 //------------------------------------------------------------
140 int main(void) {
141    printf("Bojen06: Jetzt geht es los!\n");
142    init();    // Alle Adressvariablen initialisieren
143    printf("------------------------------------------------------------\n");
144    m_as=m_en; // Englisch zur aktuellen Sprache machen
145    gibMeldungenInDerAktuellenSpracheAus();
146    printf("------------------------------------------------------------\n");
147    m_as=m_de; // Deutsch  zur aktuellen Sprache machen
148    gibMeldungenInDerAktuellenSpracheAus();
149    printf("------------------------------------------------------------\n");
150    gibMeldungenInAllenSprachenAus();
151    printf("------------------------------------------------------------\n");
152    gibEinPaarAdressVariablenAus();
153    printf("------------------------------------------------------------\n");
154    gibEinPaarAdressAusdrueckeAus();
155    printf("------------------------------------------------------------\n");
156    printf("Bojen06: Das war's erstmal!\n");
157    return 0;
158 } // main
159 /*  ------------------------------------------------------------
160 Ausgabe des Programms Bojen06:
161
162 Bojen06: Jetzt geht es los!
163 ------------------------------------------------------
164 A Please enter an integer:
165 B Your input was not correct!
166 C Please enter an integer:
167 D Here comes your input times three:
168 ------------------------------------------------------
169 A Bitte geben Sie eine Ganzzahl ein:
170 B Ihre Eingabe war nicht zulaessig!
171 C Bitte geben Sie eine Ganzzahl ein:
172 D Hier ist das Dreifache Ihrer Eingabe:
173 ------------------------------------------------------
174 E Bitte geben Sie eine Ganzzahl ein:
175 F Please enter an integer:
176 G Hier ist das Dreifache Ihrer Eingabe:
177 H Here comes your input times three:
178
```



```
179 ------------------------------------------------------
180 Ein paar Adressvariablen als Bojen dargestellt:
181 |m      |--<40CA3C>--|<842E64>|--|<842E74>|--|<40A140>|--|B|
182 |m_de   |--<40CA30>--|<842E74>|--|<40A140>|--|B|
183 |m_en   |--<40CA34>--|<842E84>|--|<40A1AB>|--|P|
184 ------------------------------------------------
185 |m_de_01|--<40A128>--|<40A140>|--|B|
186 |m_de_02|--<40A12C>--|<40A163>|--|I|
187 |m_de_03|--<40A130>--|<40A185>|--|H|
188 ------------------------------------------------
189 |m_en_01|--<40A134>--|<40A1AB>|--|P|
190 |m_en_02|--<40A138>--|<40A1C4>|--|Y|
191 |m_en_03|--<40A13C>--|<40A1E0>|--|H|
192 ------------------------------------------------------
193 Ein paar Adressausdruecke und ihre Werte etc.:
194     m+DE      : <842E64>--|<842E74>|--|<40A140>|--|B|
195     m+EN      : <842E68>--|<842E84>|--|<40A1AB>|--|P|
196 ------------------------------------------------
197   *(m+DE)+M1  : <842E74>--|<40A140>|--|B|
198   *(m+DE)+M2  : <842E78>--|<40A163>|--|I|
199   *(m+DE)+M3  : <842E7C>--|<40A185>|--|H|
200 ------------------------------------------------
201   *(m+EN)+M1  : <842E84>--|<40A1AB>|--|P|
202   *(m+EN)+M2  : <842E88>--|<40A1C4>|--|Y|
203   *(m+EN)+M3  : <842E8C>--|<40A1E0>|--|H|
204 ------------------------------------------------
205 *(*(m+DE)+M1) : <40A140>--|B|
206 *(*(m+DE)+M2) : <40A163>--|I|
207 *(*(m+DE)+M3) : <40A185>--|H|
208 ------------------------------------------------
209 *(*(m+EN)+M1) : <40A1AB>--|P|
210 *(*(m+EN)+M2) : <40A1C4>--|Y|
211 *(*(m+EN)+M3) : <40A1E0>--|H|
212 ------------------------------------------------------
213 Bojen06: Das war's erstmal!
214 ------------------------------------------------------ */
```

3.2 Die Aufgabe Bojen06

Zeichnen Sie die Adressvariablen `m`, `m_de`, `m_en`, `m_de_01`, `m_de_02`, `m_de_03`, `m_en_01`, `m_en_02` und `m_en_03` als Bojen. Als Adressen sollen Sie darin jeweils die letzten 3 Hex-Ziffern der in den Zeilen 180 bis 213 wiedergegebenen Adressen einzeichnen. Aus der Zeile 181 geht z.B. hervor, dass die Variable `m` die Adresse `40CA3C` hat. Somit sollen Sie in Ihre Bojen `A3C` als Adresse von `m` einzeichnen.