



Programmierung 1

Übungsskript

Mitschrift von Falk Jonatan Strube

Übung von Prof. Dr.-Ing. Beck

15. Januar 2016

Inhaltsverzeichnis

1	Eingebaute Datentypen	1
1.1	Zahlentypen	1
1.2	Variable	2
1.2.1	Initialisierung	2
2	Ausdrücke	3
3	Speicherklassen	5
4	Funktionen	5
5	Pointer	6
6	Benutzerdefinierte Datentypen	7
7	Listen	8

1 Eingebaute Datentypen

- int
- string
- short (oft 16 Bit / 2 Byte)
- long (so groß wie eine Adresse, abhängig vom System [32/64 bit])
- char (kann Symbol sein, aber auch eine Zahl [die Symbol repräsentiert], 8 Bit)
- float
 - kann einige Zahlen (ganzzahlig) nicht darstellen (bspw. 2)
 - hat Probleme sehr große und sehr kleine Zahlen miteinander zu addieren (durch Normierung der Exponenten kann die kleine zu 0 werden, oder Nachkommastellen verloren gehen)
- Achtung: boolean kein Datentyp in C!
 - ↪ Abfrage von true/false durch int:
 - $int = 0 \hat{=} false$
 - $int \neq 0 \hat{=} true$

1.1 Zahlentypen

Zahl: $1 \cdot 10^2 + 0 \cdot 10^1 + 8 \cdot 10^0$

⇒ 10er-System (Decimal)

Zahl: $0110 \cdot 1 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 0 \cdot 10^0 = 0 + 0 + 4 + 8 + 0 + 32 + 64 + 0 = 108$

⇒ 2er-System (Binär)

Zahl: $001 \cdot 10^2 + 101 \cdot 10^1 + 100 \cdot 10^0 = 108$

⇒ 8er-System (Octal)

Zahl: $0110 \cdot 10^1 + 1100 \cdot 10^0 = 108$

⇒ 16er-System (Hexa)

Unterschied: 108_{10} , 01101100_2 , 154_8 (in C gekennzeichnet durch $0154 \rightarrow$ Octalzahl) und $6C_{16}$ (in C gekennzeichnet durch $0x6C$)

Veranschaulichung

$108 : 2 = 54$	$R0$
$65 : 2 = 27$	$R0$
$27 : 2 = 13$	$R1$
$13 : 2 = 6$	$R1$
$6 : 2 = 3$	$R0$
$3 : 2 = 1$	$R1$
$1 : 2 = 0$	$R1$

⇒ 1101100 von unten nach oben gelesen

$$\begin{array}{ll}
 108 : 8 = 13 & R4 \\
 13 : 8 = 1 & R5 \\
 1 : 8 = 0 & R1 \\
 \Rightarrow 154 &
 \end{array}$$

$$\begin{array}{ll}
 108 : 16 = 6 & R12 = RC \\
 6 : 16 = 0 & R6 \\
 \Rightarrow 6C &
 \end{array}$$

Beispielzahl 0x12AB

Speicherblock:

1	2	A	B
---	---	---	---

 big-endian

B	A	1	2
---	---	---	---

A	B	1	2
---	---	---	---

 little-endian

Letzte Version ist die, die heutzutage meistens (Intel) verwendet wird: Das niederwertigste Byte liegt auf der niedrigsten Adresse.

2er Komplement positive Zahl: 0110 1100

Negation: 1001 0011

+1

Komplement: 1001 0100 = -108 = 0x94

	0	1	1	0	0	1	1	0	+108
1	1 ₁	0 ₁	0 ₁	1 ₁	0 ₁	1	0	0	-108
1	0	0	0	0	0	0	0	0	

1.2 Variable

4 Kennzeichen einer Variable:

- Adresse im Speicher (Ort)
- Datentyp (Verarbeitungsbreite)
- Bitkombination (Wert)
- Symbolischer Name

Ein Vektor fasst mehrere Variablen gleichen Datentyps unter einer zusammen.

1.2.1 Initialisierung

Bei der Initialisierung hat die Variable einen Ausgangswert:

- Initialisierung innerhalb einer Funktion: zufälliger Wert (alte Speicherbelegung)
- Init. außerhalb einer Funktion: 0

```

1 char c='c'; // 'c'=99 (ASCII)
2 char c=99;

```

Die Anführungszeichen bei der Wertzuweisung einer char ist nur bei einem Zeichen wichtig, nicht bei Zahlen.

- `'x'` einzelne Anführungszeichen für ein einzelnes Zeichen
- `"xy"` doppelte Anführungszeichen bei mehrere Zeichen

```

1 int i=8, j=5, k;
2 char c=99, d='d';
3 float x=0.005, y=-0.01, z;
4 x=(i+j); // k bleibt int (Wert: 13)
5 z=y+x; // x+y werden als double zusammengerechnet (Rechnung immer in double).
6 // z ist dann wieder float (Wert: -0.005)
7 k=x+y; // k ist int, beim Rechnen wird gebrochenzahliger Teil abgeschnitten (Wert: 0)
8 z=i/j; // i/j wird in int gerechnet, also 8/5=1 und nicht 1.6!
9 // z ist dann trotzdem float (Wert: 0.000...)
10 char a,b;
11 a=b=c; // erst wird b=c='c'=99, dann wird a=(b=c)=99 zugewiesen.
12 // a und b haben nachfolgend den Wert 'c'
13 i=j=1.1; // j=1.1(bzw. 1), also i auch Wert 1
14 z=k=x; // k=x=0 (wird abgeschnitten), also z=0.00000...
15 i+=2; // i=i+2;
16 y-=x; // y=y-x=-0.015
17 k=j=5?i:j; // das selbe wie k=((j==5)?i:j);
18 // ist j=5? Wenn ja, dann k=i. Wenn nein, dann k=j.
19 z=y>=0?1.0:2.0; // das selbe wie z=((y>=0)?1.0:2.0);
20 printf("%d\n", (3*i-2*j)*(2*d-c)); // Punkt vor Strich! 14%101=14
21 printf("%d\n", (2*((i/5)*(4*(j-3))%(i+j-2)));
22 // Achtung: i/5=8/5=1 (s.o.)!
23 // 2*9=18
24 printf("%d\n", (i-3*j)*(c+2*d)/(x-y));
25 // -7%29 = -7
26 // dann: -7/0.015)=-466.6667
27 printf("%d\n", i<=j); // 0
28 printf("%d\n", i>=j); // 1
29 printf("%d\n", i==6); // 0
30 printf("%d\n", i!=6); // !6 entspricht !(ungleich Null)=0
31 printf("%d\n", i!=6); // 1
32 printf("%d\n", i>0 && j<5);
33 printf("%d\n", i&j); // i bitweise mit j Verknüpft (ge-UND-et):
34 // 00001000
35 // &00000101
36 // =00000000
37 i=8, j=5;
38 printf("%s\n", i && j? "true": "false");
39 printf("%d\n", i && j);
40 printf("%d\n", i & j);

```

2 Ausdrücke

Simple Sort

```

1 #include <stdio.h>
2
3 int data[] = {7,3,9,2,5};
4

```

```

5 int main(){
6     int ige, iro; // entsprechende Pfeile unter den Zahlen auf Papier
7     for (irt=0; irt<(5-1); irt++){
8         for (ige = irt+1, ige<5, ige++){
9             if (data[ige] < data[irt]){
10                int tmp = data [ige];
11                data[ige] = data[irt];
12
13                data[irt] = tmp;
14                // tauschen alternativ (ohne Zwischenspeichern): (^= ist XOR)
15                // data[irt]^=data[ige];
16                // data[ige]^=data[irt];
17                // data[irt]^=data[ige];
18            }
19        }
20    }
21    for (irt=0; irt<5; irt++){
22        printf("%d ", data[irt]);
23    }
24    printf("\n");
25    return 0;
26 }

```

Alphabetische Sortierung

```

1 #include <stdio.h>
2
3 #define N 10
4
5 // [10]: Länge der Zeichenkette
6 char data[][10] = {"Max", "Huckebein", "Bolte", "Lempel", "Maecke",
7 "Helene", "Antonius", "Schlich", "Moritz", "Boeck"};
8
9 int main(){
10     int ige, iro, ibl; // entspr. Pfeile unter den Zahlen auf Papier
11     for (irt=0; irt<(N-1); irt++){
12         for (ige = irt+1, ige<N, ige++){
13             for (ibl = 0; data[irt][ibl] == data[ige][ibl] &&
14                 data[irt][ibl]!=0; ibl++){
15                 ;
16             }
17             if (data[irt][ibl] > data[ige][ibl]){
18                 char tmp;
19                 // ibl muss nicht auf 0 gesetzt werden, vertauscht muss sowieso
20                 // erst ab dem ungleichen Zeichen getauscht werden
21                 for (/* ibl = 0 */; ibl<N ; ibl++){
22                     tmp = data[irt][ibl];
23                     data[irt][ibl] = data[ige][ibl];
24                     data[ige][ibl] = tmp;
25                     // alternativ wieder:
26                     // data[irt][ibl] ^= data[ige][ibl];
27                     // data[ige][ibl] ^= data[irt][ibl];
28                     // data[irt][ibl] ^= data[ige][ibl];
29                 }
30             }
31         }
32     }
33     for (irt=0; irt<N; irt++){
34         printf("%d ", data[irt]);
35     }
36     printf("\n");
37     return 0;
38 }

```

3 Speicherklassen

- Register: Prozessor-Register relativ schnell
- Volatile: Variable wird immer im Hauptspeicher gespeichert (Gegenteil von Register)
- Static: Liegt die Variable in einer Funktion, dann existiert sie über die gesamte Laufzeit des Programms (kann aber trotzdem nur innerhalb der Funktion verwendet werden). Liegt die Variable außerhalb einer Funktion, dann wird Variablennahme nur im aktuellen C-Quelltext verwendet (wenn sich Programm aus mehreren Quelltexten zusammengesetzt wird).
- Extern: Gegenteil von Static außerhalb einer Funktion
- Auto: automatische Variable. Wird beim Aufruf der Funktion, die die Variablendefinition enthält, angelegt. Bei jedem Funktionsaufruf neu. Wird vernichtet, wenn Funktion beendet ist.

4 Funktionen

```

1 int test(){...}
2 // gleich wie int main: Leerer Ausgabewert ist int (nicht void!)
3 test(){...}
4 // void: unbestimmter Ausgabewert bzw. kein Ausgabewert
5 void test(){...}

```

Übung Sinus-Funktion: $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots$

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <math.h>
4
5 double sinus(double x);
6
7 char vbuf[128];
8
9 int main(){
10     double x,y;
11     fgets(vbuf,128,stdin);
12     x = atof(vbuf);
13     y = sinus(x);
14     printf("sin(%lf): %lf\n",x,y);
15     return 0;
16 }
17
18 double sinus(double x){
19     int i=3, vz=-1;
20     double erg=x, summand=x;
21     while (summand> 0.00005){
22         summand = summand * x * x/(i*(i+1));
23         i += 2;
24         erg += summand * vz;
25         vz += -1;
26     }
27     return erg;
28 }

```

5 Pointer

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <ctype.h>
4
5 char* upperstr(char* data){
6     int i = 0;
7     while(data[i] != '\n'){ // (*data != '\n')
8         // data[i]=toupper(data[i]);
9         // Alternative zu toupper:
10        // A ist 0x41 = 0100 0001
11        // a ist 0x61 = 0110 0001
12        // also bloß ein Bit verschieben!
13        if (data[i] >= 'a' && data[i] <= 'z'){
14            data[i] &= ~(1<<5);
15            // 1101 1111, damit verunden=> alle werden negiert 0010 0000
16            // &= Bitweise addition => invertierung von der 3. Stelle
17        }
18        // oder auch (entsprechend angepasst ohne i in while usw.)
19        // *data = toupper(*data);
20        // data++;
21        i++;
22    }
23    return data;
24 }
25
26 char vbuf[128]
27
28 int main(){
29     printf("Eingabe: ");
30     fgets (vbuf,128,stdin);
31     upperstr(vbuf);
32     puts(vbuf);
33     return 0;
34 }

```

Weiterführend:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Interpretation als Array
5 int mystrlen1(char *p){
6     int i;
7     for (i=0; p[i] != 0; i++);
8     return i;
9 }
10
11 // Nutzen des Pointers
12 int mystrlen2(char *p){
13     int count;
14     while (*p++)
15         count++;
16     return count;
17 }
18
19 // Noch mehr nutzen des Pointers
20 int mystrlen3(char *p){
21     char *px=p; // Pointer auf das erste Zeichen merken
22     while (*p++); // Pointer hoch zählen (wenn letzte Stelle, ist *p++ 0, also false)
23     return p-px-1; // (letzte Stelle)-(erste Stelle)-1
24 }

```



```

25
26 int intarr[] = {5,7,2,8,9};
27
28 // Interpretation als Array
29 int containsint(int n, int* pdata, int test){
30     int i;
31     for (i=0; i<n && pdata[i]!=test; i++);
32     return pdata[i]==test;
33 }
34
35 // Nutzen der Pointer: *(pdata+i) ist das selbe wie pdata[i]
36 int containsint2(int n, int* pdata, int test){
37     while(*(pdata++n)!= test && n!=0);
38     return *(pdata+n)== test;
39 }
40
41 int mystrcmp(char* p1, char* p2){
42     int i;
43     for (i=0; p1[i]==p2[i] && p1[i]; i++);
44     return p1[i]-p2[i];
45 }
46
47 int mystrcmp2(char* p1, char* p2){
48     while (*p1-*p2 && *p1) p1++,p2++;
49     return *p1-*p2;
50 }
51
52 int main (int argc, char* argv[]){
53     int i, len;
54
55     i=atoi(argv[1]);
56     if(containsint(sizeof intarr/sizeof(int), intarr, i)) puts("Enthalten");
57     else puts("nicht Enthalten");
58
59     if(mystrcmp(argv[1],argv[2])==0) printf("%s gleich %s\n",argv[1],argv[2]);
60     else printf("%s ungleich %s\n",argv[1],argv[2]);
61
62     for (i=0; i<argc; i++){
63         puts(argv[i]); // Ausgabe der Eingabeparameter
64                         // (wenn aufgerufen durch "./a.out e1 e2" werden
65                         // "./a.out", "e1" und "e2" ausgegeben)
66         // Ausgabe des jeweils ersten Zeichens
67         printf("%c\n",argv[i][0]); // alternativ auch *argv[i]
68         // argv[0] ist immer der Programmname
69
70         for (i=0; i<argc; i++){
71             len=mystrlen1(argv[i]);
72             printf("len: %d\n", len);
73         }
74     }
75     return 0;
76 }

```

6 Benutzerdefinierte Datentypen

Enum:

Aufzählungstyp (festgesetzte Bezeichnungen auf einen integer-Wert).

Struct:

Zusammenfassung von mehreren Komponenten (unterschiedliche eingebaute Dateitypen als un-

initialisierte Variablen), die durch einen Namen beschrieben werden. Verwendung zur Modellierung eines Sachverhalts (wie im Beispiel Student mit seinen Eigenschaften).

Typedef:

Es wird ein synonymer Typname für einen existierenden Typnamen erstellt. So kann die Variable-initialisierung verkürzt werden (im Skript: struct tStudent→tStud).

Union:

Datensätze werden im Vergleich zum Struct übereinander geschrieben.

7 Listen

Implementation 1

list.h:

```

1 // Strukturtyp für Konnektor (Element mit Inhalt):
2 typedef struct TCNCT{
3     struct TCNCT* next; // tCnct geht noch nicht innerhalb!
4     void *pltem; // void für generische Daten
5 }tCnct;
6
7 typedef struct{
8     tCnct* pFirst;
9     tCnct* pLast;
10    tCnct* tCurr;
11 }tList;

```

Listenimplementation (list.c):

```

1 #include <stdlib.h>
2 #include "list.h"
3
4 // erzeugt leere Liste:
5 tList *CreateList(void){
6     tList* ptmp;
7     // Speicher freigeben:
8     ptmp=malloc(sizeof(tList));
9     if(ptmp!=NULL){
10         // offene Liste: anfängliches tList hat nur NULL-Pointer
11         ptmp->pFirst=ptmp->pLast=ptmp->pCurr=NULL;
12     }
13     return ptmp;
14 }
15
16 // hinten einfügen:
17 int InsertTail (tList* pList, void *pltemIns){
18     // Verschieden Situationen: Anfügen an leere oder schon vorhandene Liste
19     tCnct *ptmp = malloc(sizeof(tCnct));
20     ptmp->next=NULL;
21     if(ptmp){
22         ptmp->pltem = pltemIns; // Connector mit Inhalt füllen
23         if (pList->pFirst!=NULL){ // Liste Leer
24             pList->pFirst=pList->pLast = ptmp;
25         } else { // Liste enthält schon Konnektoren
26             pList->pLast->next=ptmp; // Das vorher letzte Element zeigt nun auf das eingefügte
27                                     // und damit neue letzte Element
28             pList->pLast = ptmp; // das neue letzte Element
29         }
30         pList->pCurr=ptmp; // Das Element, mit dem zuletzt hantiert wurde ist pCurr
31     }
32     return (int)ptmp;

```

```

33 }
34
35 // gibt ersten Eintrag aus:
36 void* GetFirst (tList* pList){
37     tCnct *ptmp;
38     ptmp = pList->pFirst;
39     if (ptmp){
40         pList->pCurr=ptmp;
41         return ptmp->pltem;
42     }
43     return NULL;
44 }
45
46 // gibt nächsten Eintrag aus:
47 void* GetNext (tList* pList){
48     tCnct *ptmp = pList->pCurr;
49     if (ptmp){
50         if (ptmp==pList->pLast){ // kein Nachfolger vorhanden
51             return NULL;
52         } else {
53             ptmp = ptmp->next;
54             pList->pCurr = ptmp;
55             return ptmp->iltem;
56         }
57     }
58     return GetFirst(pList);
59 }
60
61 // gibt letzten Eintrag aus:
62 void* GetLast (tList* pList){
63 }

```

Vorgehen bei malloc/fopen usw. immer:

- 1.) Malloc machen
- 2.) Überprüfen, ob es geklappt hat!

Unterschied: Offene Liste und Ringliste. Offene Liste startet mit NULL-Zeigern.

Implementation 2 (Doppelt-verkettete Ringlist)

Vorteil: Jedes Element hat einen Vorgänger und einen Nachfolger. Dadurch reicht eine Funktion, die nach einem Element ein neues einfügen kann. Das kann an beliebiger Stelle passieren.

list.h