



Programmierung 1

Übungsskript

Mitschrift von Falk Jonatan Strube

Übung von Prof. Dr.-Ing. Beck

13. Januar 2016

Inhaltsverzeichnis

1	Eingebaute Datentypen	1
1.1	Zahlentypen	1
2	Ausdrücke	2
3	Speicherklassen	3
4	Funktionen	3
5	Pointer	4
6	Benutzerdefinierte Datentypen	6

1 Eingebaute Datentypen

1.1 Zahlentypen

Zahl: $1 \cdot 10^8 + 0 \cdot 10^7 + 0 \cdot 10^6 + 0 \cdot 10^5 + 1 \cdot 10^4 + 0 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 0 \cdot 10^0$

⇒ 10er-System (Decimal)

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

⇒ 2er-System (Binär)

Zahl: $001|101|100 = 108$

⇒ 8er-System (Octal)

Zahl: $0110|1100 = 108$

⇒ 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	

$108 : 8 = 13$	$R4$
$13 : 8 = 1$	$R5$
$1 : 8 = 0$	$R1$
⇒ 154	

$108 : 16 = 6$	$R12 = RC$
$6 : 16 = 0$	$R6$
⇒ $6C$	

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	

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 for (irt=0; irt < N; irt++){
33     printf("%d ", data[irt]);
34 }
35 printf("\n");
36 return 0;
37 }

```

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=mystrlen(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 Datentypen als uninitialisierte 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.