# Programmierung 1

## Übungsskript

Mitschrift von Falk Jonatan Strube

Übung von Prof. Dr.-Ing. Beck

13. Januar 2016

# Inhaltsverzeichnis





*Mitschrift von*
*Falk Jonatan Strube*

# 1 Eingebaute Datentypen

## 1.1 Zahlentypen

Zahl: $\underset{8\cdot10^0}{1}\ \underset{0\cdot10^1}{0}\ \underset{1\cdot10^2}{8}$
$\Rightarrow$ 10er-System (Decimal)
Zahl: $0110\ \underset{\leftarrow usw.}{1}\ \underset{1\cdot2^2}{0}\ \underset{0\cdot2^1}{0}\ \underset{0\cdot2^0}{0} = 0+0+4+8+0+32+64+0 = 108$
$\Rightarrow$ 2er-System (Binär)
Zahl: $\underset{1}{001}|\underset{5}{101}|\underset{4}{100} = 108$
$\Rightarrow$ 8er-System (Octal)
Zahl: $\underset{6}{0110}|\underset{C}{1100} = 108$
$\Rightarrow$ 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

$$\begin{aligned} 108 : 2 &= 54 && R\emptyset \\ 65 : 2 &= 27 && R\emptyset \\ 27 : 2 &= 13 && R1 \\ 13 : 2 &= 6 && R1 \\ 6 : 2 &= 3 && R\emptyset \\ 3 : 2 &= 1 && R1 \\ 1 : 2 &= 0 && R1 \end{aligned}$$

$\Rightarrow 1101100$ von unten nach oben gelesen

$$\begin{aligned} 108 : 8 &= 13 && R4 \\ 13 : 8 &= 1 && R5 \\ 1 : 8 &= 0 && R1 \end{aligned}$$

$\Rightarrow 154$

$$\begin{aligned} 108 : 16 &= 6 && R12 = RC \\ 6 : 16 &= 0 && R6 \end{aligned}$$

$\Rightarrow 6C$

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.


Mitschrift von
Falk Jonatan Strube

**2er Komplement** positive Zahl: $\boxed{0}110\ 1100$
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

```c
#include <stdio.h>

int data[] = {7,3,9,2,5};

int main(){
  int ige, iro; // entsprechende Pfeile unter den Zahlen auf Papier
  for (irt=0; irt<(5-1); irt++={
    for (ige = irt+1, ige<5, ige++){
      if (data[ige] < data[irt]){
        int tmp = data [ige];
        data[ige] = data[irt];

        data[irt] = tmp;
        // tauschen alternativ (ohne Zwischenspeichern): (^= ist XOR)
        // data[irt]^=data[ige];
        // data[ige]^=data[irt];
        // data[irt]^=data[ige];
      }
    }
  }
  for (irt=0; irt<5; irt++={
    printf("%d ", data[irt]);
  }
  printf("\n");
  return 0;
}
```

Alphabetische Sortierung

```c
#include <stdio.h>

#define N 10

//[10]: länge der Zeichenkette
char data[][10] = {"Max", "Huckebein", "Bolte", "Lempel", "Maecke",
  "Helene", "Antonius", "Schlich", "Moritz", "Boeck"};

int main(){
  int ige, iro, ibl; // entspr. Pfeile unter den Zahlen auf Papier
  for (irt=0; irt<(N-1); irt++={
    for (ige = irt+1, ige<N, ige++){
      for (ibl = 0; data[irt][ibl] == data[ige][ibl] &&
        data[irt][ibl]!=0; ibl++){
        ;
      }
      if(data[irt][ibl] > data[ige][ibl]){
        char tmp;
        // ibl muss nicht auf 0 gesetzt werden, vertauscht muss sowieso
```



Mitschrift von
Falk Jonatan Strube

```
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: $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots$

```
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>
```

```c

// Interpretation als Array
int mystrlen1(char *p){
  int i;
  for (i=0; p[i]!=0; i++);
  return i;
}

// Nutzen des Pointers
int mystrlen2(char *p){
  int count;
  while (*p++)
    count++;
  return count;
}

// Noch mehr nutzen des Pointers
int mystrlen3(char *p){
  char *px=p; // Pointer auf das erste Zeichen merken
  while (*p++); // Pointer hoch zählen (wenn letzte Stelle, ist *p++ 0, also false)
  return p-px-1; // (letzte Stelle)-(erste Stelle)-1
}

int intarr[] = {5,7,2,8,9};

// Interpretation als Array
int containsint(int n, int* pdata, int test){
  int i;
  for (i=0; i<n && pdata[i]!=test; i++);
  return pdata[i]==test;
}

// Nutzen der Pointer: *(pdata+i) ist das selbe wie pdata[i]
int containsInt2(int n, int* pdata, int test){
  while(*(pdata+--n)!= test && n!=0);
  return *(pdata+n)== test;
}

int mystrcmp(char* p1, char* p2){
  int i;
  for (i=0; p1[i]==p2[i] && p1[i]; i++);
  return p1[i]-p2[i];
}

int mystrcmp2(char* p1, char* p2){
  while (*p1-*p2 && *p1) p1++,p2++;
  return *p1-*p2;
}

int main (int argc, char* argv[]){
  int i, len;

  i=atoi(argv[1]);
  if(containsInt(sizeof intarr/sizeof(int), intarr, i)) puts("Enthalten");
  else                                                  puts("nicht Enthalten");

  if(mystrcmp(argv[1],argv[2])==0)  printf("%s gleich %s\n",argv[1],argv[2]);
  else                              printf("%s ungleich %s\n",argv[1],argv[2]);

  for (i=0; i<argc; i++){
    puts(argv[i]);  // Ausgabe der Eingabeparameter
                    // (wenn aufgerufen durch "./a.out e1 e2" werden
                    // "./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 unintialisierte 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 Variableninitialisierung 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 *pItem;  // 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 <stlib.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 *pItemIns){
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->pItem = pItemIns; // 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->pItem;
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(ptmpt==pList->pLast){ // kein Nachfolger vorhanden
51        return NULL;
52      } else {
53        ptmp = ptmp->next;
54        return ptmp->iItem;
55      }
56    }
57    return GetFirst(pList);
58  }
59
60  // gibt letzten Eintrag aus:
61  void* GetLast (tList* pList){
62  }
```

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-verkettte 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