Fakultät Informatik/Mathematik

HOCHSCHULE FÜR
TECHNIK UND WIRTSCHAFT
DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Lehrstuhl für Programmierung, Compilerbau/Interpreter, Mikrocontroller

Übungsmitschrift

PROGRAMMIERUNG 1

Mitschrift von

Falk-Jonatan Strube

Übung von

Prof. Dr.-Ing. Arnold Beck

25. März 2018

INHALTSVERZEICHNIS

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 represäntiert], 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!
  $\curvearrowright$ Abfrage von true/false durch int:
  $int = 0 \mathrel{\widehat{=}} false$
  $int \neq 0 \mathrel{\widehat{=}} true$

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

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

$\Rightarrow 1101100$ 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.

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	

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

```
char c='c'; // 'c'=99 (ASCII)
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

```
int i=8, j=5, k;
char c=99, d='d';
float x=0.005, y=-0.01, z;
x=(i+j);  // k bleibt int (Wert: 13)
z=y+x;  // x+y werden als double zusammengerechnet (Rechnung immer
   in double).
        // z ist dann wieder float (Wert: -0.005)
k=x+y;  // k ist int, beim Rechnen wird gebrochenzahliger Teil
   abgschnitten (Wert: 0)
z=i/j;  // i/j wird in int gerechnet, also 8/5=1 und nicht 1.6!
        // z ist dann trotzdem float (Wert: 0.000...)
char a,b;
a=b=c;  // erst wird b=c='c'=99, dann wird a=(b=c)=99 zugewiesen.
        // a und b haben nachfolgend den Wert 'c'
i=j=1.1;  // j=1.1(bzw. 1), also i auch Wert 1
z=k=x;  // k=x=0 (wird abgeschnitten), also z=0.00000...
i+=2; // i=i+2;
y-=x; // y=y-x=-0.015
k=j=5?i:j;  // das selbe wie k=((j==5)?i:j);
            // ist j=5? Wenn ja, dann k=i. Wenn nein, dann k=j.
z=y>=0?1.0:2.0; // das selbe wie z=((y>=0)?1.0:2.0);
printf("%d\n", (3*i-2*j)%(2*d-c) ); // Punkt vor Strich! 14%101=14
printf("%d\n", (2*((i/5)*(4(j-3))%(i+j-2) );
// Achtung: i/5=8/5=1 (s.o.)!
// 2*9=18
printf("%d\n", (i-3*j)%(c+2*d)/(x-y) );
// -7%29 = -7
// dann: -7/0.015)=-466.6667
printf("%d\n", i<=j );  // 0
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

```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++){
```

```
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

```c
int test(){...}
// gleich wie int main: Leerer Ausgabewert ist int (nicht void!)
test(){...}
// void: unbestimmter Ausgabewert bzw. kein Ausgabewert
void test(){...}
```

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

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

double sinus(double x);

char vbuf[128];

int main(){
  double x,y;
  fgets(vbuf,128,stdin);
  x = atof(vbuf);
  y = sinus(x);
  printf("sin(%lf): %lf\n",x,y);
  return 0;
}

double sinus(double x){
  int i=3, vz=-1;
  double erg=x, summand=x;
  while (summand> 0.00005){
    summand = summand * x * x/(i*(i+1));
    i += 2;
    erg += summand * vz;
    vz += -1;
  }
  return erg;
}
```

5 POINTER

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

char* upperstr(char* data){
  int i = 0;
  while(data[i]!='\n'){ // (*data!='\n')
    // data[i]=toupper(data[i]);
    // Alternative zu toupper:
    // A ist 0x41 = 0100 0001
    // a ist 0x61 = 0110 0001
    // also bloß ein Bit verschieben!
    if (data[i]>='a' && data[i]<='z'){
      data[i] &= ~(1<<5);
      // 1101 1111, damit verunden=> alle werden negiert 0010 0000
      // &= Bitweise addition => invertierung von der 3. Stelle
    }
    // oder auch (entsprechend angepasst ohne i in while usw.)
    // *data = toupper(*data);
    // data++;
    i++;
  }
  return data;
}

char vbuf[128]

int main(){
  printf("Eingabe: ");
  fgets (vbuf,128,stdin);
  upperstr(vbuf);
  puts(vbuf);
  return 0;
}
```

Weiterführend:

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

// 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");
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).

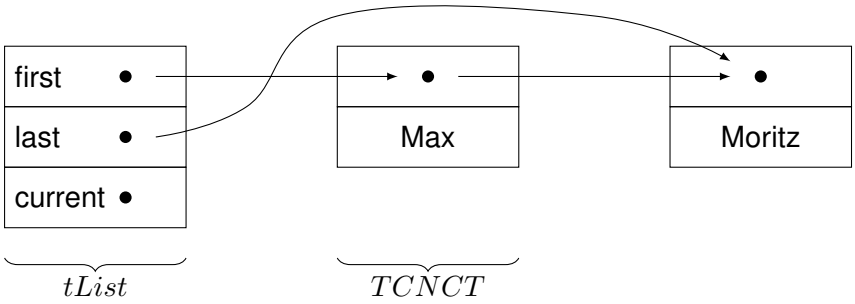
Struct:
Zusammenfassung von mehreren Komponenten (unterschiedliche eingebaute Dateitypen als un- intialisierte Variablen), die durch einen Namen beschrieben werden. Verwendung zur Modellierung eines Sachverhalts (wie im Beispiel Student mit seinen Eigenschaften).

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

Union:
Datensätze werden im Vergleich zum Struct übereinander geschrieben.

7 LISTEN

Implementation 1 (Ringliste)



list.h:

```
// Strukturtyp für Konnektor (Element mit Inhalt):
typedef struct TCNCT{
  struct TCNCT* next; // tCnct geht noch nicht innerhalb!
  void *pItem;  // void für generische Daten
}tCnct;

typedef struct{
  tCnct* pFirst;
  tCnct* pLast;
  tCnct* tCurr;
}tList;
```

Listenimplementation (list.c):

```
#include <stlib.h>
#include "list.h"

// erzeugt leere Liste:
tList *CreateList(void){
  tList* ptmp;
  // Speicher freigeben:
  ptmp=malloc(sizeof(tList));
  if(ptmp!=NULL){
    // offene Liste: anfängliches tList hat nur NULL-Pointer
    ptmp->pFirst=ptmp->pLast=ptmp->pCurr=NULL;
  }
  return ptmp;
}

// hinten einfügen:
int InsertTail (tList* pList, void *pItemIns){
  // Verschieden Situationen: Anfügen an leere oder schon
     vorhandene Liste
  tCnct *ptmp = malloc(sizeof(tCnct));
  ptmp->next=NULL;
```

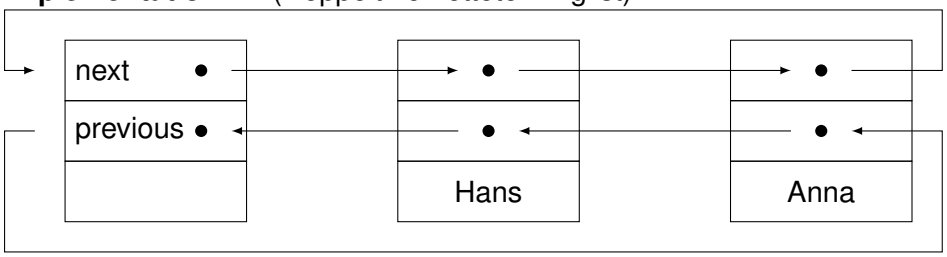
```
21    if(ptmp){
22      ptmp->pItem = pItemIns; // Connector mit Inhalt füllen
23      if (pList->pFirst!=NULL){ // Liste Leer
24        pList->pFirst=pList->pLast = ptmp;
25      } else {  // Liste enthält schon Konnektoren
26        pList->pLast->next=ptmp;  // Das vorher letzte Element zeigt
             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        pList->pCurr = ptmp;
55        return ptmp->iItem;
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!

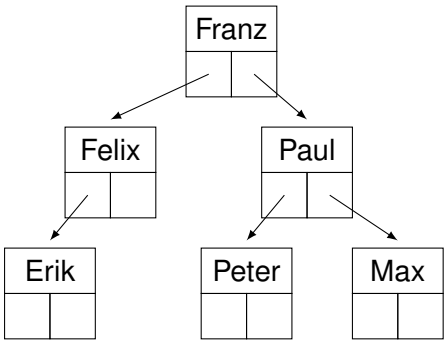
Unterschied: Offene Liste und Ringliste. Offene Liste startet mit NULL-Zeigern.

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

8 BINÄRE BÄUME




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

typedef struct{
  char *pdata;
  // pointer weil:
  // char geht nicht: nur ein character
  // string gibt's noch nicht
  // char[] ungünstig, weil die Inhalte auch unterschiedlich lang
     seien könnten.
  // also char*
  struct TNODE *pl;
  struct TNODE *pr;
} tNode

tNode treeInit={};

char* data[]={"moritz", "max", "melli", "albert", "hans", "peter",
   "malte",
              "maximilian", "james", "johannes" , "paul",  NULL);
tNode *pTree;
char buf[128];

void addToTree(tNode *pt, char* pdata){
  if(pt->pdata == NULL){ // im Falle, dass noch kein Node existiert
    pt->pdata=pdata;
  } else {
    if (strcmp(pt->pdata, pdata)>0){
      //links
      if(pt->pl == NULL){
        pt->pl = malloc(sizeof(tNode));
        *(pt->pl) = treeInit;
        attToTree(pt->pl,pdata);
      }
```

```
    } else {
      //recht
      if(pt->pr == NULL){
        pt->pr = malloc(sizeof(tNode));
        *(pt->pr) = treeInit;
        attToTree(pt->pl,pdata);
      }
    }
  }
}

void dispTree(tNode *pt){
  if (pt->pl != NULL) dispTree(pt->pl);
  puts(pt->pdata);
  if (pt->pr != NULL) dispTree(pt->pr);
}

int main(){
  char *ptmp;
  char **p=data;
  // erstes Node erstellen (leer):
  pTree = malloc(sizeof(tNode));
  *pTree = treeInit;

  while  (*p){
    addToTree(pTree, *p);
    p++;
  }

  dispTree(pTree);
}
```

Alternativ mit array, void pdata (Funktionspointer) usw.:

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

typedef struct{
  void *pdata;
  struct TNODE* px[2];
} tNode

tNode treeInit={};

char* data[]={"moritz", "max", "melli", "albert", "hans", "peter",
   "malte",
                "maximilian", "james", "johannes" , "paul",  NULL);
tNode *pTree;
char buf[128];

int mycmp(void*p1, void*p2){
```

```
18    return (strcmp((char*)p1, (char*)p2>0)?0:1;
19  }
20
21  void addToTree(tNode *pt, void* pdata,  int (*fcmp)(void*, void*)){
22    int i;
23    if(pt->pdata == NULL){
24      pt->pdata=pdata;
25    } else {
26      i=fcmp(pt->pdata, pdata);
27        if(pt->px[i] == NULL){
28          pt->px[i] = malloc(sizeof(tNode));
29          *(pt->px[i]) = treeInit;
30        }
31        attToTree(pt->px[i],pdata, mycmp);
32    }
33  }
34
35  void mydisp(void *pdata){
36    puts((char*)pdata);
37  }
38
39  void dispTree(tNode *pt, void (*putx)(void*)){
40    if (pt->px[0] != NULL) dispTree(pt->px[0], putx);
41    putx(pt-pdata);
42    if (pt->px[1] != NULL) dispTree(pt->px[1], putx);
43  }
44
45  int main(){
46    char *ptmp;
47    char **p=data;
48    // erstes Node erstellen (leer):
49    pTree = malloc(sizeof(tNode));
50    *pTree = treeInit;
51
52    while  (*p){
53      addToTree(pTree, *p, mycmp);
54      p++;
55    }
56
57    dispTree(pTree, mydisp);
58
59    while (1){
60      fgets(buf,128,stdin);
61      buf[strlen(buf)-1]=0;
62      ptmp=malloc(strlen(buf)+1);
63      strcpy(ptmp,buf);
64      puts("==========");
65      addToTree(pTree, ptmp, mycmp);
66      dispTree(pTree, mydisp);
67  }
```