Programmierung 1

Übungsskript

Falk Jonatan Strube

Übung von Prof. Dr.-Ing. Beck

25. November 2015

Inhaltsverzeichnis

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.

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

```
#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

```
#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
        // erst ab dem ungleichen Zeichen getauscht werden
        for (/*ibl = 0*/; ibl<N ; ibl++){
          tmp = data[irt][ibl];
          data[irt][ibl] = data[ige][ibl];
          data[ige][ibl] = tmp;
          // alternativ wieder:
          // data[irt][ibl] ^= data[ige][ibl];
          // data[ige][ibl] ^= data[irt][ibl];
          // data[irt][ibl] ^= data[ige][ibl];
      }
    }
  }
  for (irt=0; irt<N; irt++={
    printf("%d ", data[irt]);
  }
  printf("\n");
  return 0;
}
```

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

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

```
void test(){...}
```

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

```
#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

```
#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;
}
```