10. Übung

1 Strukturen
2 Felder, Referenzen und Adressen
3 Alte Klausuraufgaben
4 (Weiterarbeit an Programmieraufgaben letzte Übung)

1. Strukturierte Datentypen in C

Bisher kennen wir für die Speicherung von Werten:

Einfache Datentypen (int a; float b; char c; double d)

Felder (int f[20]; int array[20][20];)

Haben wir inhaltlich zusammengehörige Informationen müssen wir sie in unterschiedlichen Variablen oder Feldern speichern und können nur über die Vergabe von Namen Zusammenhänge herstellen.

Beispiel: Jeder Studierende hat einen Namen und eine Punktzahl
Mayer 232
Mueller 199
Schulze 251
Lehmann 188

1. Strukturierte Datentypen in C

Beispiel: Jeder Studierende hat einen Namen und eine Punktzahl. Maximal sind 40 Studierende zu verwalten.
Mayer 232
Mueller 199
Schulze 251
Lehmann 188
...

Speicherung ohne Strukturen:

- Liste von Namen (2D-Array –
  1. Dimension- Größe 40 – Anzahl der zu verwaltenden Studierenden,
  2. Dimension – Größe 20 – Maximale Länge eines Namens,
  z.B. 20 Zeichen)
- Liste von Punktzahlwerten (int-Werte – Array der Länge 40)

```
#define N 40
char studnamen[N][20];
int studpunkte[N];
```

1. Strukturierte Datentypen in C

Speicherung mit Strukturen:
Anlegen eines Datentyps struct mit Name _studi und
den Feldern name und punkte
erzeugen einer Variable p als Feld von 40 Einträgen.

```
#define N 40

struct _studi {
        char name [20];
        int punkte;
} p[N];
```

1. Strukturierte Datentypen in C

Spielereien mit der Struktur:

```
#include <stdio.h>

#define N 40

struct _studi { /*Der Datentyp heisst struct _studi */
   char name [20];
   int punkte;
} p[N], pers; /* p[N] ist Variable (Array von 40 Studierenden), pers ist Variable */

int eingabe (struct _studi f[]){
           /* Funktion zur Eingabe in Feld f */
}

int main (void) {
   int a, b, i;
   struct _studi q ={"Meyer", 166}; /* Variable */
   struct _studi *z; /* Zeiger auf Struktur */
   /* Array mit Initialisierung von 3 Studierenden im Feld */
   struct _studi pfeld[] ={{"Mueller", 142},{"Meier", 152},{"Lehmann", 144} };

   printf(" Name = %s, Punkte = %d\n", q.name, q.punkte);/* Ausgabe Variable q */

   printf(" Laenge Feld = %d oder %d Eintraege\n",sizeof pfeld, (sizeof pfeld/sizeof pfeld[0]));
   for(i=0;i<3; i++) { /* Ausgabe der Eintraege eines Feldes */
             printf(" Name = %s, Punkte = %d\n", pfeld[i].name, pfeld[i].punkte);
   }
   ...
```

1. Strukturierte Datentypen in C

```
   printf(" Nun mit Zeigern!\n");
   z = &q;/* Adresse von Variable q */
   printf(" Name = %s, Punkte = %d\n",z->name, z->punkte);

   z = pfeld; ;/* Adresse vom Array pfeld*/
   for(i=0;i<3; i++) { /* Ausgabe der Eintraege eines Feldes */
            printf(" Name = %s, Punkte = %d\n",z->name, z->punkte);
   }

   a = eingabe(p); /* Funktion Eingabe muss noch programmiert werden */
   z = p;
   for(i=0;i<a; i++, z++) { /* Ausgabe der Eintraege eines Feldes */
            printf(" Name = %s, Punkte = %d\n",z->name, z->punkte);
   }
   for(i=0;i<a; i++) { /* Ausgabe der Eintraege eines Feldes */
            printf(" Name = %s, Punkte = %d\n",p[i].name, p[i].punkte);
   }
   return 0;
}
```

2. Felder und Referenzen

Betrachten wir folgendes Codefragment, wobei T irgend ein Typ (hier float) ist.

```
float a[5], *r;
r = a;
```

Dies besagt, r ist eine Referenz auf die Variable a[0].
Die Variable a[0] hat den Typ float. Eine Variable vom Typ float benötigt
s = sizeof float ➔ also 4 Byte Speicher.

Wenn also a[0] die Adresse p hat, dann besitzt a[1] die Adresse p + s.
Und allgemein hat a[k] die Adresse p + k * s.

Wenn r den Typ *float hat, und in r irgendeine Adresse p gespeichert ist, liefert der Ausdruck r+k die Adresse p + k * s, wobei gilt s = sizeof(float).

Den Wert s nennen wir manchmal auch Objektgröße einer Referenz.

2. Felder und Referenzen

Teilaufgabe aus Prüfung: Gegeben sei die folgende Deklaration eines Feldes: (9 Punkte)

```
float feld [5][4]; /* IEEE 754 single precision */
```

Vervollständigen Sie die folgende Tabelle, in dem Sie den Typ, die Adresse und die Objektgröße der folgenden Ausdrücke angeben.

Ausdruck	Typ	Adresse	Objektgröße
feld	float(*)[4]	p	16
&feld[0][0]			
&feld[1][2]			
feld[2]			
&feld			

2. Felder und Referenzen

Notizen:

```
float feld [5][4]; /* IEEE 754 single precision */
```

Ausdruck				
feld	Zeiger auf float[4]-Arrays			
&feld[0][0]	Adresse des ersten Feldelementes			
&feld[1][2]	Adresse des 3. Feldelementes in der 2. Zeile			
feld[2]	Adresse des ersten Feldelementes in der 3.Zeile			
&feld	Adresse eines 2-dimensionalen Feldes der Groesse 5 x 4			

feld[0][0], feld[0][1], feld[0][2], feld[0][3],
feld[1][0], feld[1][1], feld[1][2], feld[1][3],
feld[2][0], feld[2][1], feld[2][2], feld[2][3],
feld[3][0], feld[3][1], feld[3][2], feld[3][3],
feld[4][0], feld[4][1], feld[4][2], feld[4][3],

2. Felder und Referenzen

3) Gegeben sei die folgende Deklaration eines Feldes: (9 Punkte)

```
float feld [5][4]; /* IEEE 754 single precision */
```

Vervollständigen Sie die folgende Tabelle, in dem Sie den Typ, die Adresse und die Objektgröße der folgenden Ausdrücke angeben.

Ausdruck	Typ	Adresse	Objektgröße
feld	float(*)[4]	p	16
&feld[0][0]	float (*)	p	4
&feld[1][2]	float (*)	p+24	4
feld[2]	float (*)	p+32	4
&feld	float(*)[5][4]	p	80

feld[0][0], feld[0][1], feld[0][2], feld[0][3],
feld[1][0], feld[1][1], feld[1][2], feld[1][3],
feld[2][0], feld[2][1], feld[2][2], feld[2][3],
feld[3][0], feld[3][1], feld[3][2], feld[3][3],
feld[4][0], feld[4][1], feld[4][2], feld[4][3],

2. Felder und Referenzen

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int i, j;
      float feld[5][4]; /* IEEE 754 single precision */
      float (*p1)[4], *p2, *p3, *p4;
      float (*p5)[5][4];
      for(i=0; i<5;i++)
            for(j=0;j <4;j++)
            feld[i][j]= i*10+j;
            /* Feld gefuellt mit Werten */

      p1 = feld;
      p2 = &feld[0][0];
      p3 = &feld[1][2]; /* [0][0]... [0][4],[1][0],[1][1]   6 Werte ueberlesen */
      p4 = feld[2]; /* [0]-4 Werte,[1] – 4 Werte ([0][0]... [0][4],[1][0]...[1][4] ) 8 Werte ueberlesen */
      p5 = &feld;
      printf("sizeof feld ist %d\n", sizeof feld);
      /* Wert (*(*p1)) und naechste Zeile (*(*p1) + 1) */
      printf("sizeof p1 ist %d, Wert: %f, %f\n", sizeof *p1, (*(*p1)), (*(*p1) + 1));
      printf("sizeof p2 ist %d, Wert: %f, diff %d\n", sizeof *p2, *p2, (p2-&feld[0][0])*sizeof(float));
      printf("sizeof p3 ist %d, Wert: %f, diff %d\n", sizeof *p3, *p3, (p3-&feld[0][0])*sizeof(float));
      printf("sizeof p4 ist %d, Wert: %f, diff %d\n", sizeof *p4, *p4, (p4-&feld[0][0])*sizeof(float));
      printf("sizeof p5 ist %d, Wert: %f\n", sizeof *p5, *(*(*p5)));
      printf("\n");

  return 0;
}
```

```
$ pruef.exe
sizeof feld ist 80
sizeof p1 ist 16, Wert: 0.000000, 1.000000
sizeof p2 ist 4, Wert: 0.000000, diff 0
sizeof p3 ist 4, Wert: 12.000000, diff 24
sizeof p4 ist 4, Wert: 20.000000, diff 32
sizeof p5 ist 80, Wert: 0.000000
```

3. Aufgabe

Bestimmen Sie basierend auf dem folgenden Quelltextfragment, den Typ, die Speicheradresse sowie die Objektgröße (in Byte) der Ausdrücke in der Tabelle. (10 Punkte)

```
int a[10][5];
```

Ausdruck	Typ	Adresse	Objektgröße
a[0][0]	int	x	4
&a			
a			
*(a+1)			
a[3]+2			
&a[5]			

(10 Minuten selbstständig)

3. Aufgabe

Kreuzen Sie alle Aussagen an, die wahr sind. (2 Punkte)

- o Strukturvariablen werden Call-by-Reference an eine Funktion übergeben.
- o Der Wert einer float-Variablen wird bei der Übergabe an eine Prozedur in die lokale Variable kopiert.
- o Der Ausdruck &x liefert eine Referenz auf die Variable x.
- o Funktionen können nicht als Referenz an eine Prozedur übergeben werden.

Gegeben sei das folgende Quelltextfragment. Bestimmen Sie den Wert der Variablen x 3 Punkte und y nach dem Aufruf der Prozedur swap. Nennen Sie außerdem die Methode für die Parameterübergabe an die Prozedur swap. (3 Punkte)

```
1 void swap(int a, int b) {
2   int tmp = a;
3   a = b;
4   b = tmp;
5 }
6 void block() {
7   int x = 0, y = 1;
8   swap(x, y);
9 }
```

x = y=

Implementieren Sie eine Prozedur matrixMult, die drei Matrizen Call-by-Reference übergeben bekommt. Dabei sollen die ersten beiden Matrizen n x k und k x m Dimensionen enthalten. Die Prozedur soll die beiden Matrizen multiplizieren und das Ergebnis in der dritten übergebenen Matrix speichern. (5 Punkte)

[2] Matrizenmultiplikation der Matrizen $A = (a_{ij})$ und $B = (b_{jk})$ in die Ergebnismatrix $C = (c_{ik})$:

$$c_{ik} = \sum_{j=1}^{m} a_{ij} \cdot b_{jk}$$

3. Aufgabe

1.2 Adressberechnung und mehrdimensionale Felder

1) Welche Ausgaben erzeugt das Programm? Kennzeichnen Sie dabei fehlerhafte Werte und falsche Dereferenzierungen!

14 Punkte

```
#include <stdio.h>

int a = 8;

int change(int x, int *y, int *z){
   x++;
   a++;
   (*y)++;
   z[1]++;
   y = z+1;
   *(z+2) = 2**(y-1);
   return x;
}

void main() {
   int i = 5, j = 15, k;
   int *l;
   int m[3] = { 23, 24 };
   printf("%d,%d,%d,%d,%d,%d,%d\n", a, i, j, k, *l, m[1], m[2]);

   l = m;
   k = change(i, &j, m);
   printf("%d,%d,%d,%d,%d,%d,%d\n", a, i, j, k, *l, m[1], m[2]);
}
```

3. Aufgabe

1.2 Adressberechnung und mehrdimensionale Felder

2) In einem Array sollen 50 Zeichenketten abgelegt werden. Jede Zeichenkette ist maximal 16 Zeichen lang. Definieren Sie eine Variable in C, die alle Daten aufnehmen kann! Wie viel Speicher belegt diese Variable, wenn ein Zeichen ein Byte belegt? (2 Punkte)

3) In einem 2-dimensionalen Array von int-Werten mit 20 Zeilen und 30 Spalten soll die Zeile ermittelt werden, in der die meisten Zahlen größer als 100 vorkommen.
Schreiben Sie eine Funktion maxLargeNums, die den Index dieser Zeile liefert! Falls mehrere Zeilen mit der maximalen Anzahl von Zahlen größer als 100 existieren, kann der Index einer beliebigen davon zurückgegeben werden. (10 Punkte)

4. Aufgaben

2. Ermittlung der Häufigkeiten von 1er, 2er und 3er Buchstabenkombinationen in Texten

- Globale Variablen und Definitionen

```
/* 26 Buchstaben + Sonstige */
#define N 27

char alphabet[] = "abcdefghijklmnopqrstuvwxyz#";
double F_abc[N][N][N];
double F_ab[N][N];
double F_a[N];
double F;
```

- Schreiben Sie eine Funktion `int possiblechar(int c)`, die für das Zeichen c prüft, ob es sich um ein Zeichen aus dem Alphabet handelt (1 wenn zutreffend, 0 wenn nicht).
- Nutzen Sie die Funktion readchar um zeichenweise Texte einzulesen:

```
int readchar() { /* Lese gueltige Zeichen ein und ueberlese andere, ermittle index in Tabelle */
            int c, anz = 0;
            /* Ueberlese nicht gueltige Zeichen und mehrere Leerzeichen*/
            while (((c = getchar()) != EOF) && (!possiblechar(c))) {
                        anz++;
            }
            if(c == EOF) return EOF;
            if (c >= 'a'&&c <= 'z') return c - 'a';
                        else if (c >= 'A'&&c <= 'Z')return c - 'A';
                                    else return N;
}
```

- Schreiben Sie eine Funktion `void fcount()`, die in den Feldern F_abc, Fab, F_a und der Variablen F die 3er, 2er und 1er Häufigkeiten und die Gesamtanzahl der gültigen Zeichen zählt. Nach der Zählung einer Kombination rücken die Buchstaben von b zu a und von c zu b auf, auf c wird ein neues Zeichen (readchar) gelesen. Stimmt ihre prozentuale Verteilung der Buchstaben in F_a mit der Verteilung aus der Cäsarkodierung überein, wenn Sie das Buch Mobi dick analysieren?