4. Übung

1. Zyklen und Felder

Für zwei Felder *a*, *b* mit je *n* Elementen soll geprüft werden, ob ein Element in beiden Arrays enthalten ist. Dazu wird jedes Element des Feldes a mit jedem Element des Feldes b verglichen. Bei der ersten gefundenen Übereinstimmung kann die Suche abgebrochen werden.

Wir betrachten dazu das folgende Code-Fragment:

```
int i,j;

for(i = 0; i < n; i++)
  for(j = 0; j < n; j++)
    if(a[i] == b[j]) {   } /* Merken und Zyklen beenden */
```

Wie können bei einer gefundenen Übereinstimmung beide Zyklen beendet werden?

2. Felder und Matrizen

Geben Sie handschriftlich ein C-Programm an, das eine Matrix-Vektor-Multiplikation mit einer vorgegebenen Matrix *m* und dem Vektor *x* durchführt!

```
double m[4][4] = { {1,0,0,0}, {0, 0.7071, -0.7071, 0 }, {0, 0.7071, 0.7071, 0 }, {0,0,0,1} };
double x[4] = {120, 80, 77.5, 1};
```

Die Matrix-Vektor-Multiplikation ist anhand der folgenden mathematischen Vorschrift (z.B. aus einer Formelsammlung entnommen) durchzuführen:

$$y_i = \sum_{j=1}^{n} m_{ij} x_j$$

Achten Sie auf die richtig Wahl der Indizes für das erste und das letzte Element in einer Dimension.

3. Sortierte Felder und Einfügen

Gehen Sie von einem Feld aus, das bereit n Elemente aufsteigend sortiert enthält. Schreiben Sie eine Funktion, die ein neues Element in dieses Feld an seine der Sortierung entsprechende Position einfügt! Nutzen Sie das folgende Programmfragment und die angegebene Funktionsschnittstelle!

```
int einfuegen(int feld[], int n_elem, int max_elem,
              int einfueg);

…

int werte[100] = {1,4,12,17,21,34,39,42,47};

int max=100; int n=8;

int neu=27;

n = einfuegen(werte, n, max, neu);
```

Funktion einfuegen:
Parameter:
int feld[] … Feld in das eingefügt werden soll
int n_elem … aktuelle Anzahl belegter Elemente
int max_elem … maximale Anzahl Elemente
int einfueg … einzufügender Wert
Rückgabe: neue Anzahl der Elemente nach dem Einfügen

4. Finden von Fehlern in C-Programmen

Die unten dargestellten Varianten einer Funktion *transpose()* sollen eine quadratische Matrix transponieren. Die Matrix wird als Parameter *m* übergeben, die Anzahl der Elemente je Zeile und Spalte als Parameter *n (n ≤ 20)*.
Einige Varianten enthalten Fehler, die entweder schon beim Übersetzen der Funktion gefunden werden (syntaktische Fehler und z.B. fehlerhafte Typen) oder zu einem Fehlverhalten bei Ausführung der Funktion führen würden (fehlerhafte Ergebnisse und Laufzeitfehler).Eine Variante ist fehlerfrei.
Der Aufruf der Funktion geschieht wie folgt:

```
#define N 20
void transpose( float m[N][N], int n);
int main()
{ float matrix[N][N];
  int nn=8;
  // …
  transpose(matrix,nn);
  // …
}
```

Markieren Sie die Fehler und entscheiden Sie, ob der Fehler bei der Übersetzung der Funktion oder zur Laufzeit des Programms auftritt! Markieren Sie auch die fehlerfreie Variante!

```
// VARIANTE A
# define N 20
…
void transpose( float m[N][N], int n)
{
  float i, j, tmp;
  for (i=0; i<n; i=i+1)
  {
    for (j=i+1; j<n; j=j+1)
    {  tmp = m[i][j];
       m[i][j] = m[j][i];
       m[j][i] = tmp;
     }
  }
}
```

```
// VARIANTE B
# define N 20
…
void transpose( float m[ ][N], int n)
{
  int i, j;
  float tmp;
  for (i=0; i<n; i=i+1)
  {
    for (j=i+1; j<n; j=j+1)
    {  tmp = m[i][j];
       m[i][j] = m[j][i];
       m[j][i] = tmp;
     }
  }
}
```

```
// VARIANTE C
# define N 20
…
void transpose( float m[][], int n)
{ int i, j ;
  float tmp[20][20];
  for ( i=0; i<n; i=i+1)
  {  for ( j=0; j<n; j=j+1)
     {  tmp[i][j] = m[j][i];
     }
  }
 m = tmp;
}
```

```
// VARIANTE D
# define N 20
…
void transpose( float m[N][N], int n)
{ int i, j ;
  for ( i=0; i<n; i=i+1)
    for ( j=i+1; j<n; j=j+1)
    {  m[N-1][N] = m[i][j];
       m[i][j] = m[j][i];
       m[j][i] = m[N-1][N];
     }
}
```