A3: Arbeiten mit Feldern

Neue Elemente von C++: Zeiger, Felder, dynamische Speicherverwaltung

Aufgaben:

1) Schreiben Sie eine Funktion

```
void Ausgabe( const double *feld, int n)
```

die ein Feld der Länge n auf dem Bildschirm ausgibt. Legen Sie dann im Hauptprogramm `main()` ein statisches `double`-Feld mit den Einträgen `{ 47, 11, 0, 8, 15}` an und geben Sie es mit Hilfe der Funktion `Ausgabe` auf dem Bildschirm aus.

2) Rufen Sie im Hauptprogramm `main()` die Funktion `Ausgabe` so auf, das diesmal das Feld ab dem dritten Eintrag ausgegeben wird, also `0, 8, 15`.

3) Schreiben Sie eine Funktion

```
double Durchschnitt( const double *feld, int n)
```

die das arithmetische Mittel der n Zahlen berechnet, die in dem übergebenen Feld gespeichert sind. Testen Sie Ihre Funktion mit dem Feld aus 1).

4) Schreiben Sie eine Funktion

```
int min_pos( const double *feld, int n)
```

die die Position $i_{\min} \in \{0, 1, \ldots, n-1\}$ des minimalen Eintrages in dem übergebenen Feld der Länge n bestimmt. Testen Sie Ihre Funktion wieder mit dem Feld aus 1).

5) Schreiben Sie eine Funktion

```
void sortiere( double *feld, int n)
```

die das übergebene Feld der Länge n aufsteigend sortiert. Dazu wird zunächst der minimale Eintrag des Feldes mit dem Eintrag an Position 0 getauscht. Anschließend wird der minimale Eintrag des verbleibenden Feldes `feld[1], ..., feld[n-1]` der Länge $n-1$ bestimmt und mit dem Eintrag an Position 1 getauscht, usw. Dabei soll jeweils die Funktion `min_pos` verwendet werden. Am Ende soll `feld` dann das sortierte Feld enthalten. Testen Sie Ihre Funktion wieder mit dem Feld aus 1).

6) Schreiben Sie eine Funktion

```
void umdrehen( double *feld, int n)
```

die die Reihenfolge der Einträge des Feldes umdreht. Damit Sie kein neues Feld benötigen, gehen Sie wie folgt vor: Vertauschen Sie erst `feld[0]` mit `feld[n-1]`, dann `feld[1]` mit `feld[n-2]` usw. Testen Sie Ihre Funktion mit dem sortierten Feld aus 5).

7) Schreiben Sie eine Funktion

```
double* glue( const double *feld1, int n1,  const double *feld2, int n2)
```

die ein *neues* Feld zurückgibt, welches zuerst die Einträge aus `feld1` und danach die Einträge aus `feld2` enthält. Dazu muss zunächst ein neues *dynamisches* Feld der Länge $n_1 + n_2$ mit Hilfe von `new double[n1+n2]` angelegt werden.

Testen Sie Ihre Funktion mit dem Feld aus 5) (zweimal hintereinander hängen). Achten Sie darauf, dass Sie den für das dynamisch erzeugte Feld angeforderten Speicher am Ende Ihres Hauptprogrammes `main()` wieder mittels `delete[]` freigeben.

Erweiterungen

1*) Schreiben Sie eine Funktion

```
int binSuche_pos( double x, const double *sfeld, int n)
```

die eine binäre Suche nach dem Element x in dem gegebenen *aufsteigend sortierten* Feld durchführt und dessen Position zurückgibt. Dabei wird x zunächst mit dem mittleren Eintrag a :=`sfeld[m]` verglichen, wobei `m = n/2` ist. Gilt $x = a$, so wird die Position `m` zurückgegeben. Für $x < a$ wird `binSuche_pos` mit der linken Hälfte `sfeld[0], ..., sfeld[m-1]` aufgerufen, andernfalls ($x > a$) mit der rechten Hälfte `sfeld[m+1], ..., sfeld[n-1]`. Ist $n \leq 0$, so wurde x nicht gefunden, und es wird `-1` zurückgegeben.

Testen Sie Ihre Funktion mit $x = 11$ bzw. $x = 42$ und dem sortierten Feld aus 5).

2*) Schreiben Sie eine Funktion

```
double* merge( const double *sfeld1, int n1,
               const double *sfeld2, int n2)
```

die die beiden aufsteigend sortierten Felder in einem *neuen* Feld so zusammenfügt, dass wieder ein aufsteigend sortiertes Feld entsteht. Dazu vergleicht man jeweils die ersten noch nicht bearbeiteten Einträge der beiden Felder und fügt den kleineren von beiden in das neue Feld ein.

3*) Schreiben Sie eine Funktion

```
void merge_sort( double *feld, int n)
```

die das übergebene Feld der Länge n aufsteigend sortiert. Dazu wird der Index `m = n/2` in der Mitte des Feldes bestimmt und die Funktion `merge_sort` rekursiv mit den Teilfeldern `feld[0], ..., feld[m-1]` sowie `feld[m], ..., feld[n-1]` aufgerufen. Anschließend werden die dann sortierten Teilfelder mithilfe der Funktion `merge` wieder in einem neuen Feld sortiert zusammengefügt. Zum Schluss werden die Einträge des neuen Feldes nach `feld` kopiert und das neue Feld anschließend mittels `delete[]` gelöscht.

4*) Schreiben Sie alle bisher geschriebenen Funktionen so um, dass mit `double`-Vektoren (Datentyp `std::vector<double>`) statt `double`-Feldern gearbeitet wird. Dazu müssen Sie mittels `#include <vector>` den Container `std::vector` aus der Standard Template Library (STL) einbinden.

Lernziele

1. **Zeiger/Felder:**
   - Zeiger als Variablen für Speicheradressen verstehen
   - statische Felder anlegen können
   - auf Elemente eines Feldes in richtiger Weise zugreifen (Vorsicht: erstes Element hat den Index 0!)
2. **dynamische Speicherverwaltung:**
   - Felder mit `new[]` dynamisch anlegen können
   - dynamische Felder mit `delete[]` wieder ordnungsgemäß freigeben