Programmieren C++:
Template-Klassen für Paare und Arrays

Klaus Kusche

Hinweis: Wir schreiben im folgenden Beispiel in den beiden Klassen nur die Methoden, die wir wirklich im Hauptprogramm brauchen.
In Wirklichkeit müssten beide Klassen noch zahlreiche Methoden mehr haben (**`Array`** bräuchte beispielsweise einen **`operator=`** und einen Copy-Konstruktor!).

Die Klasse `Paar`:

Die Objekte der Klasse **`Paar`** sollen *zwei Werte* (den "*ersten*" und den "*zweiten*" Wert) *zweier beliebiger Typen* enthalten, und zwar als private Member.

Weiters enthält die Klasse:

- Einen *Standard-Konstruktor*, der überhaupt nichts tut
  (also die Member uninitialisiert lässt).
- Einen *Konstruktor mit 2 Parametern*: Den beiden Werten,
  die im ersten und im zweiten Member gespeichert werden sollen.
- *Zwei Get-Methoden* für den ersten und für den zweiten Wert.
- Einem **`operator==`** und einem **`operator<`** jeweils mit **`bool`**-Ergebnis,
  die den *ersten Wert* beider Operanden entsprechend *vergleichen*
  (der zweite Wert wird beim Vergleich ignoriert).

Die Klasse `Array`:

Die Objekte der Klasse **`Array`** sollen ein *dynamisch angelegtes Array* von *Elementen beliebigen Typs* enthalten. Weiters enthalten sie zwei **`int`**'s: Einen für die *Größe* des Arrays und einen für die *Anzahl der tatsächlich belegten Elemente* im Array.

Weiters enthält die Klasse:

- Einen *Konstruktor* mit einem **`int`** (der gewünschten Größe des Arrays)
  als Parameter. Er legt das Array an, speichert dessen Größe im Objekt,
  und setzt die Anzahl der belegten Elemente auf 0.
  Die Arrayelemente selbst bleiben uninitialisiert.
- Einen *Destruktor* (was macht der?).
- Eine *Methode* "**`dazu`**" mit *einem Parameter* vom selben Typ wie die Array-Elemente
  (als konstante Referenz). Wenn im Array noch Platz ist,
  speichert sie den Parameter im ersten unbelegten Element des Arrays,
  erhöht die Anzahl der belegten Elemente, und liefert **`true`** als Returnwert.
  Ist kein Platz im Array mehr, tut sie nichts und liefert **`false`**.
- Eine *Methode* "**`kleinstes`**" *ohne Parameter* und eine *Methode* "**`naechstes`**"
  mit *einem Parameter*, der vom selben Typ wie die Array-Elemente ist
  (auch **`const &`**). Beide liefern einen *Pointer auf ein Array-Element als Returnwert*:

  `kleinstes` liefert einen Pointer auf *das kleinste Element* des Arrays,
  und **`naechstes`** liefert einen Pointer auf das *kleinste Element*,
  das *größer als* der als *Parameter* übergebene Wert ist.

Ist das Array *leer*, oder gibt es bei **naechstes** *kein Element mehr*,
das größer als der Parameter ist, ist der Returnwert **NULL**.

Beide Methoden sollen *außerhalb* des **class** implementiert werden.

Das Hauptprogramm soll Folgendes leisten:

- Es *definiert* einen **typedef**-*Typ* für **Paar**-Objekte,
  deren *erster Wert* vom Typ **string** und deren *zweiter Wert* vom Typ **int** ist.
  (Ohne dieses **typedef** wäre der folgende Code viel unleserlicher…)
- Es *deklariert* ein **Array**-*Objekt* mit **argc−1** Elementen,
  dessen Element-Typ der soeben definierte **Paar**-Typ ist.
- Es geht *alle Worte der Befehlszeile* durch.

  Für jedes Wort erzeugt es ein **Paar**-*Objekt*, dessen erster Wert das Wort
  und dessen zweiter Wert der Index dieses Wortes auf der Befehlszeile ist,
  und fügt dieses **Paar**-Objekt *zum Array dazu*.
- Es macht *eine Schleife mit einem Pointer* auf unsere Paare als Schleifen-Variable:
  Der Pointer beginnt beim **kleinstes**-Element unseres Arrays
  und geht nach jedem Durchlauf mittels **naechstes** zum nächstgrößeren Element.
  Wird der Pointer **NULL**, endet die Schleife und das Programm.

  In der Schleife werden beide Werte des **Paar**-Objektes *ausgegeben*,
  auf das der Pointer gerade zeigt.