Programmieren C++: Operatoren usw. plus Templates: Umbau der Messwert-Klasse auf ein Template

Klaus Kusche

Unsere **Messw**-Klasse aus der alten Übung kann derzeit nur **double**-Werte als Messwerte speichern.

1.) Umbau auf ein Template

Bau die Klasse auf eine *Template-Klasse* mit dem *Typ der Messwerte als Parameter* des Templates um, sodass man z.B. auch komplexe Zahlen als Messwerte verarbeiten könnte.

Einziges *Problem* dabei ist der **<<**-*Operator*: Lies im Skript nach, *welche Schritte* nötig sind, um eine außerhalb der Klasse als Template implementierte Operator-Funktion in einer Template-Klasse zum **friend** zu machen.

Aus dem *Typumwandlungs*-Operator **operator double** wird ein **operator T** .

Dann sollte das *Hauptprogramm wie bisher* funktionieren, wenn man die beiden bisherigen **Messw**-Objekte als **Messw<double>** deklariert.

2.) Test mit Bruchzahlen

Um zu testen, ob das Template wirklich *für beliebige Datentypen* funktioniert (solange diese ein **+** usw. implementieren), wollen wir es für Messwerte testen, die *Bruchzahlen* sind.

Wir verwenden dafür die **Bruch**-Implementierung aus unserer vonangegangenen Bruch-Taschenrechner-Übung:

- Übernimm den *kompletten Code* (außer **main**) aus der **Bruch**-*Musterlösung* (oder deiner Lösung).
- *Ersetze* in unserem **main** alle **double** durch **Bruch**.
- Falls du im *Typumwandlungs-Operator* **+=** zum *Aufsummieren* der Elemente verwendet hast: *Ersetze* das durch **=** ... **+** , weil Bruch ja kein **+=** bietet.
- Ein weiteres *Problem im Typumwandlungs-Operator* ist schwerer zu analysieren:
  Der Compiler verweigert vermutlich das **/** zur *Mittelwert-Berechnung*.
  Der Grund ist, dass die Auswahl des **/** *nicht eindeutig* ist:
  - Einerseits kann der Compiler sowohl die **Bruch**-Summe als auch die **int**-Anzahl in einen **double** konvertieren und dann eine **double**-*Division* machen.
  - Andererseits kann der Compiler die **int**-Anzahl in einen **Bruch** konvertieren und dann eine **Bruch**-*Division* machen.

  Da wir einen **Bruch**-*Mittelwert* wollen, ist die *zweite* Variante die richtige.
  Um den *ersten Fall auszuschließen*, markiert man am besten die **double**-Typumwandlung in der **Bruch**-Klasse als **explicit** .
  Dann wird sie vom Compiler *nicht mehr für automatische Typ-Anpassungen* verwendet, sondern nur mehr bei explizit programmierten Typumwandlungen.

- Für unser *Hauptprogramm* brauchen wir einen **>>**-*Operator* für **Bruch**-Werte.
  Zum Testen reicht uns eine *Lösung über einen Umweg*:
  Wir wollen den schon vorhandenen **Bruch**-*Konstruktor* verwenden,
  der aus einem **`const char *`** einen neuen **Bruch** macht.

  In *produktivem Code* ist das Einlesen mit **>>** in ein **char**-Array *fixer* Größe
  aber ***strikt verboten***, weil es *ohne Längenprüfung* erfolgt
  und daher ein *Absturz- und Sicherheitsrisiko* ist!

  Also gehen wir in unserem **>>**-Operator wie folgt vor:

  - Wir deklarieren eine **`string`**-*Hilfsvariable*
    und lesen in diese mit **>>** *das nächste Wort* ein.
  - Die **`string`**-Methode **`c_str()`** liefert einen *Pointer* auf das *interne* **char**-*Array*
    eines **`string`**-Objektes, das den String als normalen C-String incl. **`\0`** enthält.

    Wir wenden also **`c_str()`** auf unsere *Hilfsvariable* an
    und konstruieren aus diesem **char**-Array ein *temporäres* **Bruch**-*Objekt*.
  - Dieses Bruch-Objekt *speichern* wir *im rechten Operanden* des **>>** .

Wenn man unser Programm jetzt mit *Input-Files* füttert, die *ganze Zahlen und Brüche*
enthalten, sollte es auch Brüche und Bruch-Mittelwerte ausgeben.