Programmieren 1 Übung: Exceptions: Bruchrechnen

Klaus Kusche

Wir wollen das Beispiel „einfacher Taschenrechner mit Brüchen“ von vor ein paar Wochen weiter verfeinern: Die Rechnung soll nicht mehr von der Befehlszeile kommen, sondern der Reihe nach eingetippt werden (mit Zwischenräumen oder Zeilenvorschüben zwischen den Brüchen und den Rechenzeichen).

Wenn ein ungültiger Bruch eingetippt wird, soll das Programm nicht gleich abbrechen, sondern den Benutzer so lange nach einem Bruch fragen, bis ein korrekter Bruch eingetippt wird, und mit diesem Bruch die bisherige Rechnung fortsetzen.
Auch bei der Eingabe eines falschen Rechenzeichens oder einer Division durch Null soll die Rechnung mit einer nochmaligen Eingabe ab dem falschen Zeichen fortgesetzt werden können.

Um die bisherige Struktur des Beispiels möglichst zu erhalten, wollen wir Exceptions verwenden. Der Einfachheit halber werfen wir nur einen Fehlermeldungs-Text.

Ausgehend von der Lösung des alten Bruch-Beispiels sind folgende Änderungen nötig:

- Unser **`class`** bleibt völlig unverändert.
- Bei der Implementierung von **`Bruch::Bruch(const char *str)`**
  geben wir uns mehr Mühe bei der Fehlererkennung.
  Dazu verwenden wir die vordefinierte Funktion **`strtol`**
  (ein erweitertes **`atoi`**, such dir die Man-Page am Internet!):

  Zuerst machen wir ein **`strtol`** ab dem Anfang von **`str`**
  und speichern das Ergebnis als Zähler unseres Bruches.

  Ist der von **`strtol`** gelieferte Ende-Pointer gleich **`str`**,
  so fehlt dem String ein gültiger Zähler ==› Fehlermeldung werfen!

  Zeigt der Ende-Pointer auf **`'\0'`**, so ist der String nach dem Zähler zu Ende:
  Wir setzen den Nenner auf 1 und sind fertig.

  Zeigt der Ende-Pointer auf etwas Anderes als **`'/'`**,
  so steht ein ungültiges Zeichen nach dem Zähler ==› Fehlermeldung werfen!

  Sonst machen wir ein zweites **`strtol`** ab dem Zeichen hinter dem **`'/'`**
  (d.h. eins hinter dem Ende-Pointer) für den Nenner.

  Zeigt der vom zweiten **`strtol`** gelieferte Ende-Pointer unmittelbar hinter den **`'/'`**, so fehlt dem String ein gültiger Nenner ==› Fehlermeldung werfen!

  Zeigt der Ende-Pointer auf etwas Anderes als **`'\0'`**,
  so steht ein ungültiges Zeichen nach dem Nenner ==› Fehlermeldung werfen!

  Sonst wird der Bruch noch gekürzt, dann endet der Konstruktur erfolgreich.
- In **`Bruch::Kuerze()`** werfen wir statt dem Ausgeben einer Fehlermeldung im Fall Nenner gleich 0 ebenfalls einen Fehlermeldungstext.
- Als nächstes schreiben wir eine Hilfsfunktion **`Bruch LiesBruch()`** :
  Sie soll so lange einen Bruch von **`cin`** einlesen, bis das Einlesen erfolgreich ist.

Deklariere einen Hilfs-String für das Einlesen.

Wiederhole dann in einer Endlosschleife Folgendes:

Lies ein Wort in den String ein (mit **>>**), ruf unseren **Bruch**-Konstruktor mit diesem String auf, und returniere das neu angelegte **Bruch**-Objekt.

Wenn im Konstruktor eine Exception auftritt, so fange sie
und gib den Text der Exception sowie den fehlerhaften String aus.
Bereinige im **catch** auch eventuell noch vorhandenen weiteren Input,
bevor die Schleife den nächsten Einlese-Versuch macht, und zwar mit

```
cin.ignore(10000, '\n');
if (cin.eof()) exit(EXIT_FAILURE);
```

- Ändere **main** wie folgt:

  Initialisiere den Ergebnis-Bruch am Anfang mit einem Aufruf von **LiesBruch()** .

  Lass den Bruch für die rechte Seite der Rechung weg, schreib stattdessen
  direkt in den vier Rechen-Operationen **LiesBruch()** als rechten Operanden
  (denn wir wollen ja nur dann einen Bruch einlesen,
  wenn wir ein gültiges Rechenzeichen erkannt haben).

  Wir brauchen für unser Rechenzeichen eine Variable für einen einzelnen **char**.
  Lies das Rechenzeichen unmittelbar vor dem **switch** mit **>>** in diese Variable ein.

  Wir müssen irgendwie das Ende der Rechnung bzw. der Eingabe erkennen.
  Wir führen dazu **'='** als neues Rechenzeichen ein:
  Das **switch** bekommt einen neuen Fall für **'='**, der einfach gar nichts macht,
  und die bisherige **for**-Schleife über alle Worte der Befehlszeile
  wird eine **do**-**while**-Schleife, solange unser Rechenzeichen nicht gleich **'='** ist.

  Im **default**-Fall (falsches Rechenzeichen) werfen wir wieder eine Exception,
  anstatt eine Fehlermeldung auszugeben.

  Das gesamte **switch** packen wir in ein **try** (außer einem falschen Rechenzeichen
  könnte nämlich auch eine Division durch 0 auftreten!), das dazugehörige **catch**
  gibt den gefangenen Fehlermeldungs-Text sowie das Rechenzeichen aus
  und räumt dann so wie oben beschrieben den Input auf.