Objektorientierte Programmierung, Übungsaufgabe 1

Schnittstelle für Langzahl-Arithmetik

Jan Tammen <foobar@fh-konstanz.de>

29. März 2005

Inhaltsverzeichnis

1 Einleitung

Um die begrenzte Genauigkeit des Datentyps `double` zu erweitern, muss eine andere Speicherungsform für gebrochene Zahlen gewählt werden. Mögliche Optionen sind die Speicherung als String bzw. das Ablegen jeder einzelnen Ziffer in einem `integer`-Feld.

Solch eine „Langzahl" soll in einer Klasse gekapselt werden, welche die üblichen arithmetischen Operationen zur Verfüfung stellt. Um Implementierungsdetails vor dem Anwender der Klasse zu verbergen, wird ihm ein einfaches Interface, eine Schnittstelle, zur Verfügung gestellt. In der folgenden Dokumentation sollen Ansatz und Entwurf dieser Schnittstelle dargestellt werden.

2 Schnittstelle der Langzahl-Klasse

```
/**
 * @file    Langzahl.h
 * @synopsis Langzahl Interface
 * @author  Jan Tammen (FH Konstanz), <jan.tammen@fh-konstanz.de>
 * @date    2005-03-21
 */

#ifndef LANGZAHL_H
#define LANGZAHL_H

class Langzahl
{
   public:
      /// virtueller Destruktor
      virtual ~Langzahl() {};

      /// Ueberladene Operatoren
      Langzahl& operator= (const Langzahl& z);

      Langzahl& operator+ (const Langzahl& z);
      Langzahl& operator- (const Langzahl& z);
      Langzahl& operator* (const Langzahl& z);
      Langzahl& operator/ (const Langzahl& z);

      Langzahl& operator+= (const Langzahl& z);
      Langzahl& operator-= (const Langzahl& z);
      Langzahl& operator*= (const int x);
      Langzahl& operator/= (const Langzahl& z);

```

```
      bool isNull () const { return mIsNull; }
      long int getNumber(void) const;

   private:
      /// bool'scher Wert: ist Zahl = 0?
      bool     mIsNull;

      /// Vorzeichen: -1: negativ, 1: positiv
      short int mSign;
};

#endif
```

Das Konzept beruht auf der abstrakten Basis-Klasse `Langzahl`, von welcher die späteren konkreten Implementierungen abgeleitet werden. Im Einzelnen müssen die Kindklassen mindestens folgende Methoden implementieren:

operator$=$ Zuweisungsoperator. Kopiert die Datenkomponenten des Parameter-Objektes in das Zielobjekt. Gibt Selbst-Referenz zurück.

operator$+$ Additionsoperator. Addiert die gekapselte Zahl des Parameter-Objektes zur Zahl des Zielobjeteks. Gibt Selbst-Referenz zurück.

operator$-$ Subtraktionsoperator. Subtrahiert die gekapselte Zahl des Parameter-Objektes von der Zahl des Zielobjektes. Gibt Selbst-Referenz zurück.

operator$*$ Multiplikationsoperator. Multipliziert die gekapselte Zahl des Parameter-Objektes mit der Zahl des Zielobjektes. Gibt Selbst-Referenz zurück.

operator$/$ Divisionsoperator. Dividiert die Zahl des Zielobjektes durch die gekapselte Zahl des Parameter-Objektes. Gibt Selbst-Referenz zurück.

operator$+=$ Zusammengesetzter Zuweisungsoperator: Addition. Gibt Selbst-Referenz zurück.

operator$-=$ Zusammengesetzter Zuweisungsoperator: Subtraktion. Gibt Selbst-Referenz zurück.

operator$*=$ Zusammengesetzter Zuweisungsoperator: Multiplikation. Gibt Selbst-Referenz zurück.

operator$/=$ Zusammengesetzter Zuweisungsoperator: Division. Gibt Selbst-Referenz zurück.

getNumber Gibt den **ganzzahligen** Anteil der Zahl zurück.

isNull Abfrage der Datenkomponente `mIsNull`. Gibt bool'schen Wert zurück: `true`, falls Zahl $= 0$, `false`, falls Zahl $\neq 0$.

Die weiteren Implementierungsdetails, wie z.B. die interne Speicherung der Zahl, müssen jeweils in den abgeleiteten Klassen festgelegt werden.

3 Anwendung: Reihenentwicklung

In einem Anwendungsprogramm soll die Verwendung der Schnittstelle demonstriert werden. Dabei werden durch das Programm zwei einfache unendliche Reihenentwicklungen simuliert. Folgende Reihen werden dazu hier betrachtet:

$$\sum_{k=0}^{\infty} \frac{1}{2^k} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots = \mathbf{2} \qquad (1)$$

$$\sum_{k=1}^{\infty} \frac{1}{k(k+1)} = \frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \ldots = \mathbf{1} \qquad (2)$$

4 Klasse mit Datentyp double

Eine konkrete Klasse `MyDouble` welche von `Langzahl` abgeleitet wird, benutzt zur Speicherung der Zahl den C++-Standard-Datentyp `double`. Dazu der passenden Ausschnitt aus der Klassen-Definition:

```
#include "Langzahl.h"
class MyDouble : public virtual Langzahl
{
[...]
   private:
      long double mNumber;
};
```

Listing 1: Ausschnitt MyDouble.h

Die o.g. Methoden/Operatoren können nun ohne weitere Umwege direkt implementiert werden, z.B. für den Additionsoperator:

```
MyDouble& MyDouble::operator+ (const MyDouble& z)
{
   this->mNumber = (this->mNumber+z.mNumber);
   return (*this);
}
```

Listing 2: Additionsoperator MyDouble

4.1 Anwendung im Testprogramm

Im Testprogramm sieht die Verwendung der `MyNumber`-Klasse für die in (1) beschriebene Reihenentwicklung folgendermaßen aus:

```
#include "MyDouble.h"
int main ()
{
   /// [...] Abfrage Anzahl Iterationen

   MyDouble sum;
   MyDouble* potenz = new MyDouble(1.0);
   for (int i = 0; i < numIterations; ++i)
   {
      sum += potenz->getKehrwert();
      potenz->setNumber(potenz->getNumber()*2.0);
   }

   std::cout << "Summe: " << sum << std::endl;
}
```

Listing 3: Verwendung MyDouble im Testprogramm

Die dafür benutzte Hilfsmethode `getKehrwert` ist wie folgt definiert:

```
MyDouble& MyDouble::getKehrwert(void)
{
   MyDouble* tmp = new MyDouble((1.0/this->mNumber));
   return *tmp;
}
```

Listing 4: Deklaration MyDobule::getKehrwert

5 Klasse mit Langform

Für die Klassenversion, welche die Zahl intern in einer Langform speichert, werden folgende Erweiterungen des Datenteils in der Deklaration vorgenommen:

```
#include "MyNumber.h"
class MyNumber : public virtual Langzahl
{
[...]
   private:
      static const unsigned int mMaxDecimalPlaces = 100;
```

```
      long     mNumber;
      vector<int> mDecimalPlaces;
};
```

Listing 5: Ausschnitt MyNumber.h

Beschreibung der Datenkomponenten:

mMaxDecimalPlaces Anzahl der maximal speicherbaren Nachkommastellen.

mNumber Vorkommateil.

mDecimalPlaces Integer-Feld mit den Nachkommastellen.

6 Arithmetik mit Langzahlen

In den folgenden Abschnitten werden die Algorithmen für das Rechnen (Addition, Subtraktion, Multiplikation) mit den durch die zu implementierende `Langzahl`-Klasse repräsentierten Langzahlen (in Pseudocode) vorgestellt.

6.1 Addition

Bei der **Addition** von Langzahlen werden jeweils die Vorkommateile der Faktoren getrennt summiert. Dies ist durch den bei diesem Design gewählten Datentyp (`long int`) des Vorkommateils ohne weitere Hilfsmittel durchführbar. Hierbei besteht natürlich die Gefahr eines Überlaufes, auf den der Anwender entsprechend hingewiesen werden muss.

Es folgt die Summation der Nachkommateile der Faktoren. Dabei wird nach der Schulmethode vorgegangen – die einzelnen Ziffern der Faktoren werden von rechts beginnend addiert, ein eventuell auftretender Übertrag wird in die nächsthöhere Stelle übernommen. Sollte am „Ende“ der Zahl ebenfalls ein Übertrag aufgetreten sein, so muss dieser zum Vorkommateil-Summanden addiert werden.

6.1.1 Vorzeichenbehaftete Faktoren

Neben dem Standardfall $s = x + y$ müssen bei den Additionsoperatoren folgende Spezialfälle behandelt werden:

- $(-x) + y \equiv y - x$
- $x + (-y) \equiv -(y - x)$

- $(-x) + (-y) \equiv -(x + y)$

Die Vorgehensweise lässt sich dabei folgendermaßen zusammenfassen: Sind die Vorzeichen der Faktoren identisch, so ist der Betrag der Summe die Summe der Faktoren-Beträge und das Vorzeichen das gemeinsame Vorzeichen der Faktoren.

Wenn die Faktoren unterschiedliche Vorzeichen haben, muss zunächst der größere von beiden Faktoren-Beträgen ermittelt werden. Der Betrag der Summe ergibt sich nun aus der Differenz des größeren und des kleineren Faktor-Betrages. Das Vorzeichen des größeren Faktor-Betrages ist gleichzeitig auch das Vorzeichen der Summe.

6.1.2 Beispiel zur Addition

```
gegeben:
x = 12,854571, x_v = 12, x_n = 0,854571
y = 98,957235, y_v = 98, y_n = 0,957235

gesucht:
s = (x_v+x_n) + (y_v+y_n)

---------------------
Vorkommateil:
s_v = x_v + y_v = 12 + 98 = 110

Nachkommateil:
s_n = x_n + y_n:

    0,854571
+   0,957235
---------------------
    0,701706
+   1,11010  Übertrag
---------------------
=   1,811806

Addieren des Übertrags zu s_v: s_v + 1 = 111

Gesamtergebnis: s = 111,811806
```

Listing 6: Beispiel Addition Schulmethode

6.1.3 Pseudocode

Hinweis: Hier wird lediglich der „Normalfall“, also die Addition zweier positiven Zahlen betrachtet. Die anderen Fälle lassen sich auf diesen Fall bzw. die Subtraktion zurückführen.

```
Langzahl x, y, s

// extension: Zwischenergebnis, carry: Übertrag
carry, extension in ℕ
carry, extension ⟵0

s.setVorkommateil(x.vorkommaTeil + y.vorkommaTeil)

for <i> from 0 to <Länge y> do
   extension ⟵y[i] + x[i]
   extension ⟵(extension + carry)
   carry ⟵0

   if <extension ≥ 10> then
      extension ⟵(extension − 10)
      carry ⟵1
   end if

   s.addNachkommastelle(extension)
end for

if <carry > 0> then
   s.vorkommateil +⟵ (s.vorkommateil + 1)
end if
```

Listing 7: Pseudocode Addition

6.2 Subtraktion

Da bei diesem Entwurf die Speicherung der Zahl intern im Dezimalsystem erfolgt, wird für die **Subtraktion** ebenfalls die bekannte Schulmethode angewandt. Zunächst sollten ebenfalls wieder die Vorzeichen der Faktoren betrachtet werden, um Spezialfälle zu behandeln: $x - (-y) \equiv x + y$, $(-x) - y \equiv -(x + y)$, $(-x) - (-y) \equiv y - x$. Das Vorgehen zur Aufteilung der Rechnung in Vor- und Nachkommateil ist dabei analog zur Additions-Methode.

6.2.1 Beispiel zur Subtraktion

```
gegeben:
x = 12,854571, x_v = 12, x_n = 0,854571
y = 98,957235, y_v = 98, y_n = 0,957235

gesucht:
d = (x_v+x_n) - (y_v+y_n)

// Bestimmung des größeren Betrages -> "Tausch" der Variablen!

---------------------
Vorkommateil:
s_v = x_v - y_v = 98 - 12 = 86

Nachkommateil:
s_n = x_n - y_n:

    0,957235
-   0,854571
---------------------
    0,103764
-   0,00110  Übertrag
---------------------
=   0,102664

Gesamtergebnis: da getauscht wurde: d = -d = -86,102664
```

Listing 8: Beispiel Addition Schulmethode

6.2.2 Pseudocode

Hinweis: Hier wird lediglich der „Normalfall“, also die Subtraktion zweier positiven Zahlen betrachtet. Die anderen Fälle lassen sich auf diesen Fall bzw. die Addition zurückführen. Es wird weiterhin angenommen, dass in x bereits die betraglich größere Zahl vorliegt.

```
Langzahl x, y, d

// extension: Zwischenergebnis, carry: Übertrag
carry, extension in ℕ
carry, extension ⟵0

d.setVorkommateil(x.vorkommateil − y.vorkommateil)

for <i> from 0 to <Länge y> do
```

```
    if <y[i] > x[i]> then
        extension ←(x[i] + 10) − y[i]
        carry ←1
    else
        extension ←x[i] - y[i]
    end if

    extension ←(extension − carry)
    carry ←0

    d.addNachkommastelle(extension)
end for
```

Listing 9: Pseudocode Subtraktion

6.3 Multiplikation

Bei der Schulmethode für die **Multiplikation** muss zunächst der Betrag der zu multiplizierenden Faktoren gebildet werden. Anschließend wird der Multiplikand der Reihe nach von rechts nach links mit den einzelnen Ziffern des Multiplikators multipliziert. Dabei muss der Wertigkeit der Ziffern durch ein Herausrücken nach links Rechnung getragen werden. Nun werden die Teilsummen zum Gesamtergebnis aufsummiert.

Schließlich muss noch das Komma an der korrekten Stelle positioniert werden – die Anzahl der Nachkommastellen im Endergebnis ergibt sich dabei aus der Summe der Nachkommastellen der Multiplikanden.

6.3.1 Vorzeichenbehaftete Faktoren

Das Vorzeichen des Endergebnisses ist **positiv**, wenn kein oder beide Faktoren negativ sind und **negativ**, wenn einer der Faktoren negativ ist.

6.3.2 Beispiel zur Multiplikation

```
gegeben:
x = 12,345678, y = 87,654321

gesucht:
p = x * y

12345678 * 87654321
```

```
---------------------
    98765424
+    86419746
+     74074068
+      61728390
+       49382712
+        37037034
+         24691356
+          12345678
---------------------
=  1082152022374638

Setzen des Kommas nach 6+6=12 Stellen (von rechts):

p = 1082,152022374638
```

Listing 10: Beispiel Multiplikation Schulmethode

6.3.3 Pseudocode

```
Langzahl x, y, p
x ⟵abs(x)
y ⟵abs(y)

for <i> from 0 to <Länge y> do
  p +⟵ <multipliziere x mit i-ter Ziffer von y>
  <y eine Stelle nach rechts verschieben>
  <x eine Stelle nach links verschieben>
end for
```

Listing 11: Pseudocode Multiplikation Schulmethode

Anmerkung: Um o.g. Algorithmus zu verwenden, müsste eine entsprechende Methode implementiert werden, welche eine komplette Langzahl mit einem `integer` *multipliziert und dabei die „Wertigkeit“ der Stelle berücksichtigt. Alternativ könnte man dieses Vorgehen auch getrennt für Vor- und Nachkommateil wählen.*

6.4 Kehrwert einer ganzen Zahl

Der Kehrwert einer ganzen Zahl x ergibt sich allgemein durch die Bildung von $\frac{1}{x}$. Der folgende Algorithmus erreicht entweder sein Ende, falls das Ergebnis „gerade“, also endlich ist oder falls die gewünschte Anzahl an Nachkommastellen (a) erreicht ist.

6.4.1 Pseudocode

```
z ⟵1                // Zähler
n ⟵x                // Nenner
a in ℕ
Langzahl ergebnis

while <true> do
   x ⟵<größte ganze Zahl mit n * x ≤ z>
   ergebnis.setVorkommateil(x)

   if <a = 0> then
      exit
   else
      z ⟵(10*(z-n * x))
   end if

   if <z = 0> then
      exit
   end if

   x ⟵<größte ganze Zahl mit n * x ≤ z>
   ergebnis.addNachkommastelle(x)

   a ⟵a-1

end while
```

Listing 12: Pseudocode Kehrwertbildung