UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing

IPI: Einführung in die Praktische Informatik
Wintersemester 2018
Prof. Björn Ommer
ommer@uni-heidelberg.de

Übungsblatt 10: Klassen und Ein- und Ausgabe

Abgabe am 16.01.2019, 13:00.

Aufgabe 1: Fehlerrechnung (5P)

Da Messungen von Naturkonstanten fehlerbehaftet sind, möchte man üblicherweise wissen innerhalb welcher Fehlergrenzen das Messergebnis liegt. Dafür misst man den Wert w mehrfach und gibt dessen Mittelwert $\bar{w}$ mit seiner Standardabweichung (absoluter Fehler) Δw an. Das Ergebnis der Messungen wird dann üblicherweise geschrieben als $\bar{w} \pm \Delta w$. Bei einer Rechnung mit mehreren solchen fehlerbehafteten Variablen wird auch das Ergebnis der Rechnung von seinem richtigen Wert abweichen. Wie die einzelnen Messabweichungen das Ergebnis beeinflusst wird als Fehlerfortpflanzung bezeichnet. Nimmt man nun an, daß die Messfehler voneinander unabhängig und normalverteilt sind, dann gilt das Gaußsche Fehlerfortpflanzungsgesetz. Mit zwei Spezialfällen dieses Gesetzes wollen wir uns in dieser Aufgage beschäftigen.

- Bei einer Summe $s = a + b$ kann der absolute Fehler wie folgt berechnet werden:
$$\Delta s = \sqrt{(\Delta a)^2 + (\Delta b)^2}.$$
- Bei einem Produkten $p = a \cdot b$ wird üblicherweise der relative Fehler $\frac{\Delta p}{p}$ angegeben, der wie folgt berechnet wird:
$$\frac{\Delta p}{p} = \sqrt{\left(\frac{\Delta a}{a}\right)^2 + \left(\frac{\Delta b}{b}\right)^2}.$$

Beispiel: Angenommen die fehlerbehafteten Werte sind $a = 12 \pm 3$ und $b = 6 \pm 2$. Dann ist der absolute Fehler der Summe $\Delta s = \sqrt{13} \approx 3.61$ und wir schreiben kurz $s = 18 \pm 3.61$. Der relative Fehler des Produktes ist $\frac{\Delta p}{p} \approx 0.4166$, der absolute Fehler $\Delta p \approx 0.4166 * p = 30.00$ und wir schreiben kurz $p = 72 \pm 41.66\%$ bzw. $p = 72 \pm 30.00$.

Implementieren Sie eine Klasse `FehlerWert` mit den Methoden `operator+` und `operator*`, so dass das folgende Hauptprogramm läuft:

```
int main()
{
  // Konstruktor nimmt zwei double-Werte: den Wert und den absoluten Fehler
  FehlerWert a( 12.0, 3.0 ), b( 6.0, 2.0 );
```

```
  FehlerWert s = a + b;
  std::cout << s.wert() << " +- " << s.absolut() <<
    " (" << s.relativ() * 100 << " %)" << std::endl;
  FehlerWert p = a * b;
  std::cout << p.wert() << " +- " << p.absolut() <<
    " (" << p.relativ() * 100 << " %)" << std::endl;
}
```

Mit dieser Klasse können nun auch komplexere Formeln mit fehlerbehafteten Werten leicht berechnen werden.

Aufgabe 2: Lesen und Schreiben von Dateien (5P)

Wir haben uns bisher auf die Bearbeitung kurzer Zeichenfolgen beschränkt, die interaktiv vom Programm eingelesen oder auf der Kommandozeile übergeben wurden. Häufig möchte man jedoch größere Datenmengen verarbeiten, zum Beispiel längere Texte oder lange Tabellen aus gemessenen oder von einem Programm berechneten Werten.
Solche Daten werden in der Regel in Dateien gespeichert. Für den Umgang mit Dateien stehen über den Header `fstream` die beiden Klassen `std::ifstream` (Input Filestream) und `std::ofstream` (Output Filestream) zur Verfügung. Diese sind von den bereits bekannten, allgemeineren Streamklassen `std::istream` bzw. `std::ostream` abgeleitet, und man kann sie im Wesentlichen wie `std::cin` und `std::cout` benutzen. Ein solcher Filestream wird mit `stream.open(<Datei>)` geöffnet, mit `stream.good()` wird geprüft ob Lesen bzw. Schreiben möglich ist, und mit `stream.close()` wird der Filestream geschlossen.

Schreiben Sie eine Funktion, die für einen gegebenen Bezeichner `<DATEI>` auf die beiden Dateien `<DATEI>.txt` und `<DATEI>-a.txt` zugreift. Lesen Sie die erste Datei über die Funktion

```
std::istream& getline( std::istream& is, std::string& str )
```

zeilenweise ein, und schreiben Sie diesen String (wie mit `std::cout`) in die zweite Datei, allerdings mit vorangestellter Zeilennummer. Nach der Nummer sollen ein Doppelpunkt und ein Leerzeichen folgen, die sie von der ursprünglichen Zeile trennen. Testen Sie diese Funktion mit der folgenden Datei: `faust.txt`.

Anmerkung: Das Ende der Datei können Sie daran erkennen, dass die Methode `good()` `false` liefert. Für die String-Klasse sind die Operatoren so überladen, dass sich die Verknüpfung von Strings einfach als Addition schreiben lässt. Benutzen Sie dies bei der Erzeugung der Dateinamen.

Aufgabe 3: Zweidimensionales Array **(10P)**

Für viele Anwendungen benötigt man Datenstrukturen, welche einen indizierten Zugriff mit mehreren Indizes ermöglichen. Siehe beispielsweise das folgende Programmfragment, das für eine Matrixklasse Matrix eine Einheitsmatrix initialisiert:

```
Matrix A( 5, 5, 0.0 ); // Initialisiere 5x5 Matrix mit Nullen
for ( int i=0; i<A.rows(); i=i+1 )
{
  A[i][i] = 1.0;
}
```

Wir wollen uns in dieser Übung damit beschäftigen, wie man in C++ die bequeme Syntax mit zweifachem indizierten Zugriff realisieren kann. Dazu wollen wir einen Container für bool-Werte schreiben. Es gibt prinzipiell mehrere Möglichkeiten, den doppelten Zugriff zu implementieren:

- Man wählt als interne Datenstruktur ein Array von Arrays (Typ bool**). Der indizierte Zugriff kann dann direkt durch einen indizierten Zugriff in die interne Datenstruktur implementiert werden. Das korrekte Allozieren von Speicher ist in diesem Fall jedoch aufwendig und fehleranfällig.

- Man wählt als interne Datenstruktur ein einzelnes großes Array, welches alle Daten der Matrix, zeilenweise hintereinandergeschrieben, enthält. Der operator[] des Containers kann einen Pointer bool* auf den entsprechenden Zeilenanfang zurückgeben. Durch die Äquivalenz von Pointern und Arrays kann auf diesem Objekt wiederum ein operator[] aufgerufen werden. Diese Variante hat wie obige den Nachteil, daß der Nutzer, wenn er nur einen einfachen indizierten Zugriff ausführt, ein Objekt erhält, dessen Semantik unklar ist.

- Wiederum ausgehend von einem einzelnen großen Array, implementiert man den operator[] so, dass dieser ein temporäres Objekt zurückgibt auf dem nur als einzige Method ein operator[] ausgeführt werden kann. Dieser "innere" Operator hat beide Indizes zur Verfügung und muss den entsprechenden bool aus dem Datenarray heraussuchen. Es ist hier wichtig, eine Referenz anstatt einer Kopie zurückzugeben, denn sonst wäre ein Schreiben in den Container durch indizierten Zugriff (wie im Matrixbeispiel) nicht möglich.

a) Implementieren Sie die technisch eleganteste Variante Nummer 3, wobei Ihr Container das folgende Interface erfüllen soll:

```
class TwoDBoolArray
{
public:
```

```
  // Initialisiere ein n x m Array
  TwoDBoolArray( int n = 0, int m =0);
  // Copy-Konstruktor
  TwoDBoolArray( const TwoDBoolArray& other );
  // Destruktor
  ~TwoDBoolArray();
  // Zuweisungsoperator
  TwoDBoolArray& operator=( const TwoDBoolArray& other );
  // Gebe Zeilenzahl zurueck
  int rows();
  // Gebe Spaltenzahl zurueck
  int cols();

  // ein Objekt das vom operator[] zurueckgegeben wird
  class RowProxy
  {
  public:
    // Konstruktor
    RowProxy( bool* daten, int zeilenindex, int spaltenzahl );
    // der "innere" Klammerzugriffsoperator
    bool& operator[]( int j );
  private:
    bool* _daten;
    int zeilenindex;
    int spaltenzahl;
  };
  // der "aeussere" Klammerzugriffsoperator
  RowProxy operator[]( int i );
private:
  bool* daten;
  int m, n;
};
```

Beachten Sie bei Ihrer Implementierung die auf Blatt 9 besprochene "Rule of Three" [1]. [5 Punkte]

b) In der Vorlesung haben Sie die Streamoperatoren operator<< und operator>> kennengelernt. Wir wollen diese nun für unsere Klasse TwoDBoolArray überladen. Im Gegensatz zu den bisher überladenen Operatoren lassen sich diese nicht als Klassenmethoden implementieren. Dies ist bei binären Operatoren (also solchen, die zwei Argumente haben) immer nur dann möglich, wenn die Klasse als linker Operand auftritt, nicht jedoch wenn sie als rechter Operand auftritt. Dies wird besonders deutlich, wenn man die Multiplikation mit einem Skalar für eine Vektorklasse betrachtet:

[1] seit C++11 eigentlich "Rule of Five", siehe https://en.wikipedia.org/wiki/Rule_of_three_%28C%2B%2B_programming%29

```
Vektor x( 5, 1.0 ); // Initialisiere Vektor mit 5 Eintraegen
Vektor y;
double scalar;
y = x * scalar; // rufe Vektor::operator*( double scalar ) auf.
y = scalar * x; // Wir koennen double::operator* nicht ueberladen!!!
```

Auch die Streamoperatoren sind binäre Operatoren: Das linke Argument ist der Stream (auf den wir keinen Einfluss haben), das rechte Argument die Daten. Man schafft sich hier Abhilfe durch das Implementieren von "freien" Operatoren. Diese sind Funktionen, die nicht an einen Klassennamensraum gebunden sind. Der Compiler verwendet die Regeln des Function Overloadings um die richtige Implementierung auszuwählen. Sie können das folgende Gerüst für Ihre Implementierung verwenden:

```
std::ostream& operator<<( std::ostream& stream, TwoDBoolArray& array )
{
  // ...
  return stream;
}

std::istream& operator>>( std::istream& stream, TwoDBoolArray& array )
{
  // ...
  return stream;
}
```

Es ist wichtig, dass diese Funktionen den modifizierten Stream zurückgeben. Beim operator>> müssen Sie Annahmen darüber treffen, in welcher Form die Daten vorliegen. Gehen Sie davon aus, dass es sich um Textdateien bzw. Konsoleneingabe handelt, und dass die ersten beiden Zeichenketten der Eingabe die Zeilen- bzw. Spaltenzahl sind. Danach folgen zeilenweise die Einträge des Arrays. [5 Punkte]

Benutzen Sie das Hauptprogramm `TwoDBoolArray_main.cc` um die Korrektheit Ihrer Impementierung zu üeberprüfen.