BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL
GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

Fachbereich 7

MATHEMATIK

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

e-mail: Juergen.Buhl@math.uni-wuppertal.de

Grundzüge der objektorientierten Programmierung

WS2001/2002 – Übungsblatt 11

Abgabetermin: 28. Jamuar 2002

Aufgabe 1. *Matrizen , 10 Punkte*

Schreiben Sie ein C++-Programm zum Arbeiten mit quadratischen unteren Dreiecksmatrizen. Damit die Matrizen wenig Speicherplatz belegen, soll die folgende Datenstruktur verwendet werden:
Die Matrix wird über einen Zeiger auf ein Array mit Zeilenzeigern repräsentiert. Jeder Zeilenzeiger zeigt auf ein Array mit den Elementen einer Zeile der Matrix. Für die Matrixelemente oberhalb der Diagonalen wird dabei kein Speicherplatz reserviert. Matrizen sollen dynamisch angelegt und anschließend wieder freigegeben werden können.
Gehen Sie wie folgt vor:

- Implementieren Sie eine Klasse `udMatrix` mit einem positiven ganzzahligen Wert `zeile` für die Anzahl der Zeilen der Matrix und einem Zeiger auf ein Array von Zeilenzeigern `komp` als private Datenmember. Als Matrixelemente sollen Zahlen vom Datentyp `double` abgespeichert werden können.

- Beim Aufruf des Konstruktors soll die gewünschte Anzahl von Zeilen der Matrix als Argument übergeben werden. Die Matrixelemente sollen alle mit Null initialisiert werden.

- Schreiben Sie einen Destruktor.

- Sehen Sie eine Memberfunktion zum elementweisen (schreibenden) Zugriff auf eine Komponente der Matrix vor. Überladen Sie hierzu den Operator `()`.

- Überladen Sie den Operator `<<`, so daß die Ausgabe von unteren Dreiecksmatrizen entsprechend der unten angegebenen Beispielausgabe möglich ist.

- Fragen Sie im Hauptprogramm zunächst vom Benutzer die Anzahl der Zeilen n der Matrix ab. Legen Sie dann eine entsprechende untere Dreiecksmatrix A an und weisen Sie die Werte $A_{ij} = i \cdot j$, $i = 1(1)n, j = 1(1)i$ zu. Geben Sie die Matrix auf dem Bildschirm aus.

Beispielausgabe:

```
Anzahl der Zeilen = 6
(   1                       )
(   2   4                   )
(   3   6   9               )
(   4   8  12  16           )
(   5  10  15  20  25       )
(   6  12  18  24  30  36   )
```

Aufgabe 2. *Automatische Differentiation, 10 Punkte*

Die Differentiationsarithmetik ist eine Arithmetik geordneter Paare der Form

$$U = (u, u') \text{ mit } u, u' \in \mathbb{R}.$$

In der ersten Komponente von U steht der Funktionswert, in der zweiten der Wert der Ableitung. Die Regeln[1] für die Arithmetik lauten:

$$\begin{array}{lcll} U + V & = & (u, u') + (v, v') & = & (u + v, u' + v') \\ U\text{-}V & = & (u, u')\text{-}(v, v') & = & (u\text{-}v, u'\text{-}v') \\ U \cdot V & = & (u, u') \cdot (v, v') & = & (u \cdot v, u \cdot v' + u' \cdot v) \\ U \,/\, V & = & (u, u')/(v, v') & = & (u/v, (u'\text{-}(u/v) \cdot v')/v), v \neq 0 \\ \sin(U) & = & \sin((u, u')) & = & (\sin(u), u' \cdot \cos(u)) \\ \exp(U) & = & \exp((u, u')) & = & (\exp(u), u' \cdot \exp(u)) \end{array}$$

Für die Variable x und eine beliebige Konstante c folgt wegen $\frac{dx}{dx} = 1$ und $\frac{dc}{dx} = 0$

$$X = (x, 1) \text{ und } C = (c, 0).$$

Implementieren Sie die Klasse `autodiff` in Form einer Bibliothek, die später in andere Programme eingebunden werden kann. Stellen Sie hierzu eine Headerdatei `autodiff.h` und eine Implementierungsdatei `autodiff.cc` zur Verfügung.
Die Headerdatei `autodiff.h` soll im wesentlichen die Deklaration der Klasse `autodiff`, die Datei `autodiff.cc` die Implementierung der Element- und `friend`-Funktionen enthalten.
Die Klasse soll folgende Attribute, Elementfunktionen und befreundete Funktionen haben:

- **Als `private`–Datenelemente:**
  - Zwei Komponenten `u` und `du` vom Typ `double` zur Speicherung der Werte der Funktion und ihrer Ableitung.

[1] Die Regeln für die zweite Komponente sind gerade die entsprechenden Differentiationsregeln für die betrachteten Operationen.

- **Als `public`–Elementfunktionen:**
  - Einen Konstruktor mit zwei Argumenten vom Typ `double`, der den Funktionswert und den Wert der Ableitung initialisiert. Als Default-Wert für beide Komponenten soll jeweils der Wert 0 verwendet werden.
  - Eine Funktion `value()` ohne Argumente mit Ergebnistyp `double`, die den Funktionswert eines Objekts vom Typ `autodiff` zurückgibt.
  - Eine Funktion `deriv()` ohne Argumente mit Ergebnistyp `double`, die den Wert der Ableitung eines Objekts vom Typ `autodiff` zurückgibt.
- **Als `friend`-Funktionen:**
  - Vier zweistellige arithmetische Operatoren $+, -, *, /$, die die Grundrechenarten der Differentiationsarithmetik gemäß den oben angegebenen Regeln durchführen.
  - Eine Funktion `sin()` mit einem Argument vom Typ `autodiff` und Ergebnistyp `autodiff`, die Funktionswert und Ableitung der Sinusfunktion gemäß der oben angegebenen Regeln berechnet.
  - Eine Funktion `exp()` zur Berechnung der Exponentialfunktion.
  - Eine Funktion `identity()` mit einem Argument vom Typ `double` und Ergebnistyp `autodiff`, die für die Variable x das Paar $(x, 1)$ bereitstellt.

©Prof. Dr. Hans-Jürgen Buhl und Prof. Dr. Walter Krämer