Hinweis

Diese Dokumentation ist in deutscher Sprache geschrieben. Aufgrund der enormen Verbreitung der englischen Sprache im Computerbereich, insbesondere in der Programmierung, werde ich englische Variablen-/ Funktionsnamen und Kommentare verwenden. Der Dialog der Programme mit dem Benutzer wird weiterhin auf deutsch geführt.

Wichtige Stellen in der Dokumentation sind *kursiv* geschrieben. Um die Übersichtlichkeit zu erhöhen, werden Quelltexte in der Schriftart `Courier New 8pt` verfasst.

Alle Programme wurden mit Visual Studio 6 erstellt und getestet. Ich werde vermutlich nur C++, gegebenenfalls noch die MFC verwenden. Dabei versuche ich, mich an die Ungarische Notation von Microsoft zu halten.

Sollten Sie Rückfragen haben, so bin ich jederzeit unter mail@stephan-brumme.com erreichbar.

Aufgabe M1.1

Vorbemerkung: Ich arbeite mit einem 32-Bit-Compiler, `int` steht deshalb für einen 32-Bit-Integerwert.

Die Aufgabenteile 1 und 2 greifen auf die Konstante π zurück. Computer sind nicht in der Lage, solche irrationalen Zahlen darzustellen, für die vereinfachte Rechnung genügt jedoch ein angenäherter Wert. Ich habe mich daher für `double`, einen Gleitkomma-Datentyp mit 15 Stellen Genauigkeit, entschieden und die ersten 11 Stellen von π angegeben.

Im folgenden werden zusätzliche Randbedingungen für die Datentypen angegeben, die sich aus mathematischen Notwendigkeiten ergeben, z.B. darf die Höhe eines Kegelstumpfes nie 0 oder negativ sein. Diese Bedingungen haben für die programmtechnische Umsetzung keinen relevanten Einfluss, sie sollen bloß ungültige Ergebnisse verhindern. Da es bei der Aufgabenstellung im Wesentlichen jedoch um die Programmierung geht, habe ich keinerlei Bereichsüberprüfungen implementiert, erwähne sie aber in dieser Dokumentation.

a) Die Anzahl der Ecken `nEdges` eines regelmäßigen N-Ecks muss eine natürliche Zahl >2 sein. Diesen Ansprüchen genügt der Datentyp `unsigned int`. Die obere Grenze $2^{32}-1 \approx 4 \cdot 10^9$ sollte kein Problem darstellen.
   Der Radius `dRadius` muss dagegen eine positive Gleitkommazahl sein. Hier bietet sich - ebenso wie bei π - der Datentyp `double` an, der einen guten Kompromiss aus Speicherverbrauch, Darstellungsgenauigkeit und Geschwindigkeit darstellt. Die maximal darstellbare Zahl ist $1{,}7 \cdot 10^{308}$, sie sollte groß genug sein, um unerwünschte Bereichsüberläufe zu verhindern.
   Der Radius erzwingt, dass auch die Fläche `dArea` eine `double`-Zahl ist.
b) Sowohl die Höhe `dHeight`, als auch die Radien `dRadius1` und `dRadius2` sollten zweckmäßigerweise als `double` implementiert werden, da sie jede positive rationale Zahl sein können. Das Ergebnis `dVolume` ist dann dementsprechend auch vom Datentyp `double`.
c) Ich habe mich entschieden, für x_i rationale Zahlen zuzulassen. Deshalb habe ich erneut den Datentyp `double` für das Array `arData` benutzt.
   Die Anzahl der Werte `nAmount` ist eine natürliche Zahl, `unsigned int` stellt daher eine gute Wahl dar. Die Laufvariable `nLoop` hat den gleichen Wertebereich wie `nAmount` und ist somit auch vom Datentyp `unsigned int`.
   Das Ergebnis wird aus `double`-Werte ermittelt, sinnvollerweise verwende ich deshalb dafür auch `double`.

(Quellcode `M01_1.cpp` im Anhang)

Aufgabe M1.2

Das Hornerschema lässt sich folgendermaßen umformen:

$$f(x) = \sum_{i=0}^{n} \left(a_i x^i \right) = ((((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2}) \cdot x + a_{n-3}) \cdot \quad \ldots \quad + a_1) \cdot x + a_0$$

Der Kerncode für die iterative Berechnung unter Verwendung obiger Variablenbezeichner lautet:

```
double f = 0.0;
for (int i = n; n>=0; n--)
{
        f *= x;
        f += a[i];
}
```

Die Koeffizienten werden, wie gefordert, in einem Feld gespeichert, das exakt soviel Hauptspeicher benötigt, wie Koeffizienten verwendet werden, d.h. sich dynamisch der Situation anpasst.

(Quellcode `M01_2.cpp` im Anhang)

Aufgabe M1.3

Ich gehe davon aus, dass die Zinseszins-Methode verwendet wird.

Die allgemeine Formel zur Berechnung des Guthaben G_n nach n Jahren bei einer Anfangsinvestition von G_0 und einem Zinssatz von z in Prozent lautet:

$$G_n = G_0 \cdot (1+z)^n$$

Das Kapital verdoppelt sich (*ld* ist der duale Logarithmus):

$$\begin{aligned} 2 \cdot G_0 &= G_0 \cdot (1+z)^n \\ 2 &= (1+z)^n \\ 2 &= 2^{n \cdot ld(1+z)} \\ 1 &= n \cdot ld(1+z) \\ n &= \frac{1}{ld(1+z)} = \frac{\ln 2}{\ln(1+z)} \end{aligned}$$

Es ist ersichtlich, dass die benötigte Zeit nicht direkt vom Kapitaleinsatz abhängt. Indirekt besteht aber doch eine Beziehung, da der Zinssatz z je nach Kapitaleinsatz entweder 3% oder 4% beträgt.
Die Programme sind im Quellcode gut dokumentiert, so dass ich hier nur zwei Beispielrechnungen angebe:

```
Bitte geben Sie das Anfangsguthaben in DM ein: 1000
Ihr Zinssatz betraegt 3%.
Guthaben am 01.01.2001: DM 1000
Guthaben am 01.01.2002: DM 1030
Guthaben am 01.01.2003: DM 1060.9
Guthaben am 01.01.2004: DM 1092.73
Guthaben am 01.01.2005: DM 1125.51
Guthaben am 01.01.2006: DM 1159.27
Guthaben am 01.01.2007: DM 1194.05
Guthaben am 01.01.2008: DM 1229.87
Guthaben am 01.01.2009: DM 1266.77
Guthaben am 01.01.2010: DM 1304.77
Guthaben am 01.01.2011: DM 1343.92

Ihr Guthaben wird sich in 23.4498 Jahren verdoppelt haben.
```

```
Bitte geben Sie das Anfangsguthaben in DM ein: 10000
Ihr Zinssatz betraegt 4%.
Guthaben am 01.01.2001: DM 10000
Guthaben am 01.01.2002: DM 10400
Guthaben am 01.01.2003: DM 10816
Guthaben am 01.01.2004: DM 11248.6
Guthaben am 01.01.2005: DM 11698.6
Guthaben am 01.01.2006: DM 12166.5
Guthaben am 01.01.2007: DM 12653.2
Guthaben am 01.01.2008: DM 13159.3
Guthaben am 01.01.2009: DM 13685.7
Guthaben am 01.01.2010: DM 14233.1
Guthaben am 01.01.2011: DM 14802.4

Ihr Guthaben wird sich in 17.673 Jahren verdoppelt haben.
```

(Quellcode `M01_3.cpp` im Anhang)

Anhang

Aufgabe M1.1:

```
//////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M1.1
//
// author:            Stephan Brumme
// last changes:      October 14, 2000


// include I/O-streams and math library (we use sin())
#include <iostream.h>
#include <math.h>

// define pi (11 digits should be enough)
const double pi = 3.1415926535;


// calculate the area of a n-angled surface
void AreaNEdges()
{
       // amount of edges
       unsigned int nEdges;
       // radius
       double dRadius;
       // surface area
       double dArea;

       // read edges
       cout<<"Ecken: ";
       cin>>nEdges;

       // read radius
       cout<<"Umkreisradius: ";
       cin>>dRadius;

       // calculate surface area using the given formula
       dArea = (double)nEdges*0.5 * dRadius*dRadius * sin(2*pi/(double)nEdges);

       // print result
       cout<<"Ein "<<nEdges<<"-Eck mit einem Umkreisradius von "<<dRadius
              <<" hat einen Flaecheninhalt von "<<dArea<<".\n\n";
}


// calculate the volume of a cone stub
void VolumeConeStub()
{
       // height of the stub
       double dHeight;
       // radius'
       double dRadius1;
       double dRadius2;
       // volume
       double dVolume;

       // get height
       cout<<"Height of the cone stub: ";
       cin>>dHeight;

       // get both radius'
       cout<<"Radius 1: ";
       cin>>dRadius1;
       cout<<"Radius 2: ";
       cin>>dRadius2;

       // calculate volume using the given formula
       dVolume = (pi*dHeight/3.0) * (dRadius1*dRadius1 + dRadius1*dRadius2 +
dRadius2*dRadius2);

       // print result
       cout<<"Ein gerader Kreiskegelstumpf der Hoehe "<<dHeight<<" mit den Radien "<<dRadius1
              <<" und "<<dRadius2<<" hat ein Volumen von "<<dVolume<<".\n\n";
```

```
}


// calculate the arithmetic average of a data set
void ArithmeticAverage()
{
	// size of data set
	unsigned int nAmount;
	// data set
	double *arData;
	// result
	double  dAverage;

	// get size of data set
	cout<<"Von wievielen Zahlen soll das arithmetische Mittel berechnet werden: ";
	cin>>nAmount;

	// allocate memory on heap to store the data set
	arData = new double[nAmount];

	// read data set
	for (unsigned int nLoop=0; nLoop<nAmount; nLoop++)
	{
		cout<<"Wert "<<nLoop+1<<": ";
		cin>>arData[nLoop];
	}

	// sum up the data
	dAverage = 0;
	for (nLoop=0; nLoop<nAmount; nLoop++)
		dAverage += arData[nLoop];
	// divide by size of data set
	dAverage /= nAmount;

	// print result
	cout<<"Das arithmetische Mittel dieser Werte ist "<<dAverage<<".\n\n";
}


// main function
void main()
{
	AreaNEdges();
	VolumeConeStub();
	ArithmeticAverage();
}
```

Aufgabe M1.2:

```
//////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M1.2
//
// author:            Stephan Brumme
// last changes:      October 14, 2000


// include I/O-streams
#include <iostream.h>


// read all coefficients
int Read(double **arCoefficients)
{
       // store max. exponent
       int nMaxExponent;

       // get max. exponent
       cout<<"Hoechste Potenz: ";
       cin>>nMaxExponent;

       // allocate some memory on the heap
       *arCoefficients = new double[nMaxExponent+1];

       // read all coefficients
       for (int nLoop=nMaxExponent; nLoop>=0; nLoop--)
       {
              cout<<"a"<<nLoop<<": ";
              cin>>(*arCoefficients)[nLoop];
       }

       // return max. exponent
       return nMaxExponent;
}


// calculate correspending Y-value for a given X using Horner formula
double Horner(double dX, int nMaxExponent, double *arCoefficients)
{
       // result, initialize it
       double dResult = 0.0;

       // go thru the whole array and perform Horner
       for (int nLoop = nMaxExponent; nLoop>=0; nLoop--)
       {
              // dResult_new = dResult_old * dX
              dResult *= dX;
              // dResult_new = dResult_old * dX + arCoefficients[nLoop]
              dResult += arCoefficients[nLoop];
       }

       // return result
       return dResult;
}

// main function
void main()
{
       // store coefficients
       double *arCoefficients;
       // get max exponent
       int     nMaxExponent = Read(&arCoefficients);
       // the X value
       double  dX;

       // get X
       cout<<"An welcher Stelle soll der Funktionswert berechnet werden: ";
       cin>>dX;

       // calculate and print Y (using Horner)
       cout<<"f("<<dX<<") = "<<Horner(dX, nMaxExponent, arCoefficients)<<".\n\n";
}
```

Aufgabe M01_3:

```
//////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M1.3
//
// author:            Stephan Brumme
// last changes:      October 14, 2000


// include I/O-streams and mathematical support
#include <iostream.h>
#include <math.h>


void main()
{
       // initial payment
       double       dPayment;
       // interest rate
       double          dInterestRate;
       // years since payment (0..10, not 2001..2011 !)
       unsigned int nYear;

       // duration of investment
       const int    nDuration = 10;
       // interest rates
       const double dLowInterestRate  = 0.03;
       const double dHighInterestRate = 0.04;
       // boundary between both interest rate (in DM)
       const double dLimit            = 5000;


       // read payment
       cout<<"Bitte geben Sie das Anfangsguthaben in DM ein: ";
       cin>>dPayment;

       // determine interest rate
       if (dPayment < dLimit)
              dInterestRate = dLowInterestRate;
       else
              dInterestRate = dHighInterestRate;
       cout<<"Ihr Zinssatz betraegt "<<dInterestRate*100<<"%.\n";

       // show overview for first 10 years
       for (nYear = 0; nYear <= nDuration; nYear++)
              cout<<"Guthaben am 01.01."<<(2001+nYear)<<": DM "
                     <<dPayment * pow(1.0+dInterestRate, nYear)<<"\n";

       // calculate expected duration for doubling the investment
       cout<<"\nIhr Guthaben wird sich in "<<log(2)/log(1+dInterestRate)
              <<" Jahren verdoppelt haben.\n";
}
```