

Aufgabe 1:

a) Umwandlung von Dezimal- in Hexadezimalzahlen

Schreiben Sie ein C++-Programm zur Umwandlung von Dezimalzahlen (ohne Nachkommastellen) in Hexadezimalzahlen (Basis 16) ohne Nachkommastellen. Berechnen Sie beispielsweise die Hexadezimaldarstellung der Zahlen: 562; 843; 565; 341; 2446. Die zu berechnende Zahl soll vom Programm eingelesen und das Ergebnis ausgegeben werden.

(Hexadezimalzahlen sind auch als Sechszehnersystem bekannt und erlauben bei Rechner eine übersichtliche Darstellung z.B. einer Speicheradresse. Die dabei verwendeten Ziffern sind 0..9, A, B, C, D, E, F; für Interessierte: <http://de.wikipedia.org/wiki/Hexadezimalsystem>, <http://de.wikipedia.org/wiki/Zahlensystem>, Sexagesimalsystem der Babylonier <http://www.christoph-grandt.com/BABYLON.pdf>)

b) Testfunktionen (Konzept der automatischen Tests)

Schreiben Sie Testfunktionen, die verschiedene Werte ihrer Umwandlungsfunktion überprüfen. Überlegen Sie sich, welche Randbedingungen besonders überprüft werden können. Die Testbedingung und das Ergebnis soll ausgegeben werden, z.B.: „f(255) = FA not OK“, „f(255) = FF OK“

Fügen Sie Ihrem Programm eine Eingabe hinzu, die es ermöglicht, den Test zu starten, oder die Umwandlungsfunktion zu nutzen.

(Solche Testfunktionen erscheinen zunächst lästig. Sie nehmen einem aber die ständige Eingabe von Testzahlen ab. Sie ersparen sich deshalb Arbeit, wenn Sie zuerst die Testfunktionen schreiben und danach die Umwandlungsroutine. Sie erhalten dann zunächst für alle Testfunktionen „not OK“; im Laufe Ihrer Programmierung erscheint dann für immer mehr Testfunktionen „OK“. So erhält man für jeden Implementierungsschritt ein Erfolgserlebnis.)

Aufgabe 2:

a) Umwandlung von Hexadezimal- in Dezimalzahlen

Schreiben Sie ein C++-Programm zur Umwandlung von Hexadezimalzahlen (ohne Nachkommastellen) in Dezimalzahlen ohne Nachkommastellen. Berechnen Sie beispielsweise die Dezimaldarstellung der Zahlen: FE43; ABCD; 123B; 875F. Die zu berechnende Zahl soll vom Programm eingelesen und das Ergebnis ausgegeben werden.

Erweitern Sie zunächst wieder ihre Testfunktionen.

b) Eingabepfung und Erweiterung

Überprüfen Sie die Korrektheit der Eingabe. Ändern Sie das Programm so, dass durch Festlegung einer beliebigen Basis im Bereich 2-16 die Darstellung im jeweiligen Zahlensystem erfolgt.

(Hinweis: Nutzen Sie ein Array, das die verfügbaren Ziffern enthält.)

Aufgabe 1:

a) Geldwechsel

Für einen einzugebenden Geldbetrag zwischen einem Cent und 99 Cent soll angegeben werden, wie dieser Betrag auf möglichst wenige Geldstücke (1-, 2-, 5-, 10- und 50-Cent-Stücke) aufgeteilt werden kann. Schreiben Sie hierzu ein C++-Programm.

b) Testfunktionen (Konzept der automatischen Tests)

Schreiben Sie Testfunktionen, die verschiedene Werte ihrer Geldwechselfunktion überprüft. Überlegen Sie sich, welche Randbedingungen besonders überprüft werden können. Die Testbedingung und das Ergebnis soll ausgegeben werden, z.B.: „f(50) = 1x1, 4x10 not OK“, „f(51) = 1x1, 1x50 OK“.

Fügen Sie Ihrem Programm eine Eingabe hinzu, die es ermöglicht, den Test zu starten, oder die Umwandlungsfunktion zu nutzen.

(Solche Testfunktionen erscheinen zunächst lästig. Sie nehmen einem aber die ständige Eingabe von Testzahlen ab. Sie ersparen sich deshalb Arbeit, wenn Sie zuerst die Testfunktionen schreiben und danach die Umwandlungsroutine. Sie erhalten dann zunächst für alle Testfunktionen „not OK“; im Laufe Ihrer Programmierung erscheint dann für immer mehr Testfunktionen „OK“. So erhält man für jeden Implementierungsschritt ein Erfolgserlebnis.)

Aufgabe 2:

a) Erweiterter Geldwechsel

Ein rechenfauler, jedoch computerbegeisterter Kellner muss seinen Gästen einen Geldbetrag (wieder zwischen einem und 99 Cent) herausgeben. Leider fehlen ihm immer wieder unterschiedliche Geldstücke und da er auch nicht sonderlich enthusiastisch bedient, bekommt er auch kein Trinkgeld. Unterstützen Sie ihn dadurch, dass sie ein C++ Programm schreiben, das zunächst die Werte der vorhandenen Geldstücke einliest (z.B. 1-, 5- und 50 Cent) und dann den auszugebenden Betrag auf möglichst wenige Geldstücke aufgeteilt wird. (Hinweis: Nutzen Sie ein Array, das die verfügbaren Werte der Geldstücke enthält und behandeln Sie das Problem als Stellenwertsystem; s.a. <http://de.wikipedia.org/wiki/Stellenwertsystem>, und eine Ausführung über das Sexagesimalsystem der Baylonier <http://www.christoph-grandt.com/BABYLON.pdf>.)

Erweitern Sie ihre Testfunktionen entsprechend.

b) Eingabepfung und Erweiterung

Überprüfen Sie die Korrektheit der Eingabe. Überprüfen Sie ebenfalls, ob der an die Gäste auszugebende Betrag mit den vorhandenen Geldstücken zusammengestellt werden kann. Ist das nicht der Fall, geben Sie einen Hinweis und stellen Sie den nächst höheren möglichen Betrag zusammen.

Aufgabe 1:

a) Geldwechsel

Für einen einzugebenden Geldbetrag zwischen einem Euro und 999 Euro soll angegeben werden, wie dieser Betrag auf möglichst wenige Geldscheine (1-, 10-, 20-, 50-, 100- und 500-Euroscheine) aufgeteilt werden kann. Schreiben Sie hierzu ein C++-Programm.

b) Testfunktionen (Konzept der automatischen Tests)

Schreiben Sie Testfunktionen, die verschiedene Werte ihrer Geldwechselfunktion überprüft. Überlegen Sie sich, welche Randbedingungen besonders überprüft werden können. Die Testbedingung und das Ergebnis soll ausgegeben werden, z.B.: „f(50) = 1x1, 4x10 not OK“, „f(51) = 1x1, 1x50 OK“.

Fügen Sie Ihrem Programm eine Eingabe hinzu, die es ermöglicht, den Test zu starten, oder die Umwandlungsfunktion zu nutzen.

(Solche Testfunktionen erscheinen zunächst lästig. Sie nehmen einem aber die ständige Eingabe von Testzahlen ab. Sie ersparen sich deshalb Arbeit, wenn Sie zuerst die Testfunktionen schreiben und danach die Umwandlungsroutine. Sie erhalten dann zunächst für alle Testfunktionen „not OK“; im Laufe Ihrer Programmierung erscheint dann für immer mehr Testfunktionen „OK“. So erhält man für jeden Implementierungsschritt ein Erfolgserlebnis.)

Aufgabe 2:

a) Erweiterter Geldwechsel

Ein rechenfauler, jedoch computerbegeisterter Bankangestellter muss seinen Kunden einen Geldbetrag (wieder zwischen einem und 999 Euro) herausgeben. Leider fehlen ihm immer wieder unterschiedliche Geldscheine. Unterstützen Sie ihn dadurch, dass sie ein C++ Programm schreiben, das zunächst die Werte der vorhandenen Geldscheine einliest (z.B. 1-, 5- und 50 Euro) und dann den auszugebenden Betrag auf möglichst wenige Geldstücke aufgeteilt wird.

(Hinweis: Nutzen Sie ein Array, das die verfügbaren Werte der Geldscheine enthält und behandeln Sie das Problem als Stellenwertsystem; s.a. <http://de.wikipedia.org/wiki/Stellenwertsystem>, und eine Ausführung über das Sexagesimalsystem der Babylonier <http://www.christoph-grandt.com/BABYLON.pdf>.)

Erweitern Sie ihre Testfunktionen entsprechend.

b) Eingabepfung und Erweiterung

Überprüfen Sie die Korrektheit der Eingabe. Überprüfen Sie ebenfalls, ob der an die Gäste auszugebende Betrag mit den vorhandenen Geldscheinen zusammengestellt werden kann. Ist das nicht der Fall, geben Sie einen Hinweis und stellen Sie den nächst höheren möglichen Betrag zusammen.

Aufgabe 1:

a) Römische Zahlen

Eine gegebene arabische Zahl soll von einem C++-Programm in die gleichwertige römische Zahl umgewandelt werden.

Interessantes und Wissenswertes findet man zu Additions- und Stellenwertsysteme bei <http://de.wikipedia.org/wiki/Zahlensystem>, Sexagesimalsystem der Babylonier (<http://www.christoph-grandt.com/BABYLON.pdf>.)

b) Testfunktionen (Konzept der automatischen Tests)

Schreiben Sie Testfunktionen, die verschiedene Werte ihrer Umwandlungsfunktion überprüfen. Überlegen Sie sich, welche Randbedingungen besonders überprüft werden können. Die Testbedingung und das Ergebnis soll ausgegeben werden, z.B.: „f(12) = XI not OK“, „f(12) = XII OK“.

Fügen Sie Ihrem Programm eine Eingabe hinzu, die es ermöglicht, entweder den Test zu starten, oder die Umwandlungsfunktion zu nutzen.

(Solche Testfunktionen erscheinen zunächst lästig. Sie nehmen einem aber die ständige Eingabe von Testzahlen ab. Sie ersparen sich deshalb Arbeit, wenn Sie zuerst die Testfunktionen schreiben und danach die Umwandlungsroutine. Sie erhalten dann zunächst für alle Testfunktionen „not OK“; im Laufe Ihrer Programmierung erscheint dann für immer mehr Testfunktionen „OK“. So erhält man für jeden Implementierungsschritt ein Erfolgserlebnis.)

Aufgabe 2:

Umwandlung von römischen Zahlen in arabische Zahlen

Eine gegebene römische Zahl soll von einem C++-Programm in die gleichwertige arabische Zahl umgewandelt werden.

Die Ziffern der römischen Zahl sollen direkt aufeinander folgend eingegeben werden; das Ende der Zahl soll mit RETURN markiert werden.

Beispiel:

Geben Sie bitte eine römische Zahl an.
Fehlerhafte Eingabe führt zu einem falschen Ergebnis.
MCMLXXXVII

Die gleichwertige arabische Zahl ist 1987

Erweitern Sie ebenso die bestehenden Testfunktionen.

Aufgabe:

Automatische Ableitung

Die Ableitung von Funktionen zu bestimmen ist ein wichtiges Gebiet der Mathematik. Für die Praxis werden immer wieder numerische Verfahren eingesetzt, um den Wert einer Ableitung an einer bestimmten Stelle zu bestimmen. Es gibt aber auch algebraische Verfahren, ein sehr interessantes soll hier nun mit den dualen Zahlen vorgestellt werden (nicht zu verwechseln mit Binärzahlen). Duale Zahlen sind ähnlich definiert wie komplexe Zahlen, wobei die Zahl ε eine ähnliche Stellung einnimmt wie die Zahl i .

Eine duale Zahl u ist wie folgt definiert:

$$u = u + \varepsilon u' \quad \text{mit } \varepsilon^2 = 0, \text{ hierbei bezeichnet } u \text{ den reellen und } u' \text{ den dualen Teil}$$

Die folgenden Regeln für die Addition und Multiplikation lassen sich leicht nachrechnen:

$$u + v = (u + \varepsilon u') + (v + \varepsilon v') = u + v + \varepsilon u' + \varepsilon v' = u + v + \varepsilon (u' + v')$$

$$u * v = (u + \varepsilon u') * (v + \varepsilon v') = uv + \varepsilon uv' + \varepsilon u'v + \varepsilon^2 u' v' = uv + \varepsilon (uv' + u'v)$$

Bei genauerer Betrachtung fällt auf, dass der duale Teil einer dualen Zahl bereits der gesuchten Ableitung entspricht. Schreibt man duale Zahlen als ein geordnetes Paar $u = (u, u')$ so lassen sich die folgenden Ableitungsregeln in die duale Schreibweise umsetzen:

$$(u, u') + (v, v') = (u+v, u'+v')$$

$$(u, u') - (v, v') = (u-v, u'-v')$$

$$(u, u') * (v, v') = (u*v, u'*v+v'*u)$$

$$(u, u') / (v, v') = (u/v, (u'*v-v'*u)/v^2)$$

Für die Anwendung von sinus und cosinus-Funktion ergibt sich:

$$\sin((u, u')) = (\sin(u), -\cos(u)*u') \quad \text{Kettenregel beachten!}$$

$$\cos((u, u')) = (\cos(u), \sin(u)*u') \quad \text{Kettenregel beachten!}$$

Schreiben Sie ein C++-Programm, das die Addition, Subtraktion, Multiplikation sowie die Division beinhaltet. Darüber hinaus schreiben Sie Funktionen für sin und cos. Verwenden Sie eine Struktur:

```
struct dualNumber
{
    double mdReal;
    double mdDual;
}

dualNumber add(const dualNumber& u, const dualNumber& v);
dualNumber sub(const dualNumber& u, const dualNumber& v);
dualNumber mult(const dualNumber& u, const dualNumber& v);
dualNumber div(const dualNumber& u, const dualNumber& v);
dualNumber sin(const dualNumber& u, const dualNumber& v);
dualNumber cos(const dualNumber& u, const dualNumber& v);
```

Berechnen Sie die Ableitung f' und g' der Funktionen

$$f(x) = 24 * x + 22 / x + 5;$$

$$g(x) = \sin(x/360*PI)^2 + \cos(x/180*PI)$$

an den Stellen $x = 200$, $x = 150$, $x = 360$ für $f(x)$ und

an den Stellen $x = 1.5$, $x = 2.3$, $x = 0.1$ für $g(x)$.

Beachten Sie, dass Konstanten c die duale Darstellung $c = (c, 0)$ und Variablen $x = (x, 1)$ besitzen.

b) Testfunktionen (Konzept der automatischen Tests)

Schreiben Sie Testfunktionen, die verschiedene Werte ihrer Funktionen überprüft. Überlegen Sie sich, welche Randbedingungen besonders überprüft werden können. Die Testbedingung und das Ergebnis soll ausgegeben werden, z.B.: „ $f(50) = 100$ not OK“, „ $f(51) = 0.1234231$ OK“.

Fügen Sie Ihrem Programm eine Eingabe hinzu, die es ermöglicht, den Test zu starten, oder die Funktionen zu nutzen.

(Solche Testfunktionen erscheinen zunächst lästig. Sie nehmen einem aber die ständige Eingabe von Testzahlen ab. Sie ersparen sich deshalb Arbeit, wenn Sie zuerst die Testfunktionen schreiben und danach die Umwandlungsroutine. Sie erhalten dann zunächst für alle Testfunktionen „not OK“; im Laufe Ihrer Programmierung erscheint dann für immer mehr Testfunktionen „OK“. So erhält man für jeden Implementierungsschritt ein Erfolgserlebnis.)