Vorlesung 6:
Erste Schritte in C++

- Elementare Datentypen
- Ein- und Ausgabe
- Zeichenketten
- Dateien

Klassen und Objekte

- Die hervorstechendste Eigenschaft an C++ im Vergleich zu C ist die Möglichkeit, mit *Objekten* zu arbeiten.
- Ein Objekt ist im Grundsatz eine spezielle Verbundvariable; die zugehörigen Verbunddatentypen heißen *Klassen*.
- Eine Klassendefinition ist also vergleichbar mit einer Datentypdefinition.
- Objekte vom Typ der Klasse heißen **Instanzen.**
- Der Unterschied einer C++-Klasse zu einem C-`struct`-Typ ist, dass sie neben Datenfeldern auch Funktionen enthalten kann, die spezifisch zur Verarbeitung von Objekten dieser Klasse geeignet sind.

  Diese Funktionen heißen *Methoden,* sie werden wie Datenelemente mit dem Namen eines Objektes und Punktoperator aufgerufen und arbeiten dann mit den Daten dieses Objektes. Die Methode selbst ist aber für alle Objekte einer Klasse ident.

UMIT
the health & life sciences university

Prozedural versus objektorientiert

- **Prozedurale Programmierung:**
  - Trennt Daten von Methoden
  - Korrektheit und Kompatibilität von Daten und Funktionen muss durch Programmierer/in sicher gestellt werden
  - Werden Daten(strukturen) geändert, so müssen alle abhängigen Funktionen geändert werden und umgekehrt
- **Objektorientierte Programmierung:**
  - Fasst Datenstrukturen und zugehörige Funktionen in Objekten zusammen
  - Ermöglicht Abstraktion von Objekttypen in Klassenbeschreibungen und davon abhängige Konzepte, z. B. Vererbung
  - **Weniger fehleranfällig, besser wiederverwendbar, einfachere Wartung**

Klassen und Objekte

- Wir werden im nächsten Semester sehen, wie man Klassen und Objekte in C++ selbst deklariert und implementiert.
- Die C++-Standardbibliothek bietet aber auch eine Vielfalt an vordefinierten Klassen für wichtige Anwendungsfälle.

  Schon ganz ohne eigene Klassen zu definieren, kann man durch Benutzung dieser Standardklassen erheblich komfortabler programmieren als in C.

  Wir wollen uns daher heute zunächst damit beschäftigen, wie man „C++ als besseres C“ benutzt.

UMIT
the health & life sciences university

Elementare Datentypen in C++ – boolesche Werte

- C++ kennt dieselben elementaren Datentypen wie C. Auch alle Operatoren funktionieren im Wesentlichen genauso wie in C.
- Der Datentyp `bool` für Wahrheitswerte, der in C99 mit dem Header `stdbool.h` nachgetragen wurde, ist in C++ ebenfalls ein elementarer Datentyp
- Darüber hinaus ist der gesamte Sprachumfang von C in C++ enthalten.
- Auch die Funktionen und Definitionen der C-Standardbibliothek stehen in C++ zur Verfügung.

  Standardheader von C sind in C++ umbenannt: aus `stdio.h` wird `cstdio` (ohne `.h`), aus `math.h` witd `cmath` usw.
- Allerdings bietet die C++-Standardbibliothek für viele Aufgaben „C++-gemäßere“ Umsetzungen, die in der Regel bevorzugt werden sollten

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode verstehen

→`0601-orthodrome.cpp`

- Legen Sie ein neues **C++**-Konsolenprojekt mit dem Quelltext aus der (in Moodle bereit gestellten) Datei `0601-orthodrome.cpp` an
- Dieses Programm soll einen Großkreiskurs (kürzeste Flugverbindung) zwischen zwei Orten auf der Erde berechnen
- Bringen Sie dieses Programm zum Laufen
- Machen Sie sich anhand dieses Programms die grundlegenden Sprachelemente für Ein- und Ausgabe in C++ sowie die im Programm auftretenden Verwendungen von C++-Zeichenketten klar
- Achten Sie auch auf die Include-Dateien

UMIT
the health & life sciences university

Streams

- Die C++-Standardbibliothek für Ein- und Ausgabe wird mittels

  ```
  #include <iostream>
  ```

  eingebunden (keine Dateiendung `.h`!)

- Außerdem sollte nach den Headerdeklarationen die Zeile

  ```
  using namespace std;
  ```

  stehen.

- Eine einfache Ausgabe kann nun wie folgt erfolgen:

  ```
  cout << "Dies ist ein Test.\n";
  ```

- **Streams** (Datenströme) sind ein typisches C++-Konzept.

UMIT
the health & life sciences university

I/O-Standard-Streams

- C++ stellt vier Standard-Streams zur Verfügung.
- Bereits aus dem Beispiel bekannt: `cout`
  - `cout` ist ein Ausgabestrom (ein Objekt der Klasse `ostream`) und repräsentiert die Standardausgabe `stdout`
- Weitere Standard-Streams: `cin`, `cerr` und `clog`
  - `cin` ist ein Eingabestrom (ein Objekt der Klasse `istream`) und entspricht der Standardeingabe `stdin`
  - `cerr` ist ein Ausgabestrom und wird zur Ausgabe von Fehlermeldungen benutzt; er entspricht `stderr` (Kanal 2 auf Systemebene)
  - `clog` ist ein Ausgabestrom und ist zur Ausgabe von Statusmeldungen, Warnungen u. Ä. vorgesehen; in der Regel erfolgen die Ausgaben ebenfalls in Kanal 2, jedoch mit Pufferung wie bei `stdout`

auch in C 3 Kanäle:
0...Ausgabe
1...Eingabe
2...Fehlerausgabe

UMIT
the health & life sciences university

Ausgabe auf Standard-Streams

- Wie im einfachen Beispiel zwei Folien vorher können Zeichenketten einfach mit dem Operator `<<` an `cout` (analog natürlich auch `cerr` oder `clog`) „gesendet“ werden.

```
cout << "Zeichenkette\n";
```

- Dies funktioniert aber nicht nur für Zeichenketten, sondern für alle elementaren Datentypen. Ein Formatstring oder dergleichen ist nicht erforderlich.

```
const double euler = 2.718281828459045;
cout << euler;
cout << "\n";
```

- Mehrere Ausgaben können aneinandergehängt werden:

```
int a=2, b=3;
cout << "Die Summe von " << a << " und " << b
     << " ist " << a+b << "\n";
```

- Lesbarer als durch `"\n"` können Zeilenumbrüche durch `endl` erzeugt werden:

```
cout << "Ausgabetext" << endl;
```

UMIT
the health & life sciences university

Ausgabe auf Standard-Streams

- Daten vom Typ `char` werden bei der Streamausgabe als Druckzeichen interpretiert.

  Beispiel: `char ch='0'; cout << ch << ' ' << 'A';` $\longrightarrow$ `0 A`

- *Übung:* Welche Ausgaben erzeugen folgende Zeilen?
  (erst überlegen, dann ausprobieren!)

```
int code = 'A'; cout << code;
```
int Zahl, entsprechend ASCII-Code von 'A'

```
char ch = code; cout << ch;
```
A

```
cout << (char) code;
```
A

- *Frage:* Was passiert bei `cout << A;` ?

wenn A definiert ist, wird entsprechendes ausgegeben, ansonsten nichts (oder Fehler?)

Auch bei der Ausgabe von Daten mittels `cout << ...` *gibt es Möglichkeiten zur Formatierung der Ausgabe. Erarbeiten Sie sich diese bitte in der häuslichen Nacharbeit!*

UMIT
the health & life sciences university

Eingabe aus `cin`

- Mit dem Operator `>>` können Daten aus dem Eingabestrom `cin` in Variablen „verschoben“ werden.

```
int n;
double x;
cin >> n; // liest Ganzzahl ein
cin >> x; // liest Gleitkommazahl ein
```

- Die Eingabe mittels `cin >>...` ist grundsätzlich mit den gleichen Schwierigkeiten und Nachteilen verbunden wie `scanf()`.

  Auch hier wird man für eine robuste Eingabe (die mit Fehleingaben der Nutzer/innen umgehen kann) am ehesten die Eingabe in Zeichenketten einlesen und diese weiter auswerten, vgl. die Lösungen im Beispielprogramm.

UMIT
the health & life sciences university

Unformatierte Ein- und Ausgabe

- Im Gegensatz zur feldweisen und damit formatierten Ein- und Ausgabe ist es auch in C++ möglich, einzelne Zeichen unabhängig von einer Formatierung zu lesen und zu schreiben.
- `cin.get(char ch)` liest das nächste Zeichen als Character ein, unabhängig davon, ob es sich um eine Ziffer, ein Leerzeichen, einen Buchstaben o. a. handelt.
- `int cin.get()` liest den Zeichencode des nächsten Zeichens ein, unabhängig davon, ob es sich um eine Ziffer, ein Leerzeichen, einen Buchstaben o. a. handelt.
- `cout.put('A')` gibt ein Zeichen (hier „`A`“) aus. Es ist gleichwertig zu `cout << 'A'`, vorausgesetzt, dass die Feldbreite 1 (oder nicht gesetzt) ist.
- `getline(cin, text)` liest eine ganze Zeile von `cin` und weist sie der Variablen `text` (Weiteres zum C++-Zeichenkettentyp `string` folgt im Anschluss) zu. Das Ende der Zeile wird durch ein Newline-Zeichen (`\n`) erkannt.
- `getline(cin, text, ":")` liest eine Zeile bis zum Auftreten von „ : “ ein.

UMIT
the health & life sciences university

Zeichenketten in C++

- Die Standardklasse `string` stellt mächtige und komfortable Methoden zum Erzeugen und Handhaben von Zeichenketten zur Verfügung.

  Dazu muss über `#include <string>` die entsprechende Header-Datei eingebunden werden.

- Wir besprechen hier nur die wichtigsten Konzepte und Methoden.

- Auch in einem C++-`string` versteckt sich ein `char`-Array. Die Speicherverwaltung des Arrays wird aber durch die Standardklasse erheblich vereinfacht.

UMIT
the health & life sciences university

Erzeugen und Initialisieren von Strings

- Deklaration einer Variablen vom Typ `string` erzeugt einen String der Länge 0:

```
string s;
```

- Eine Stringvariable kann mit einer Stringkonstanten oder mit einer anderen Stringvariablen initialisiert werden:

```
string s1 = "Hallo";
string s2 = s1;
```

- Der benötigte Speicherplatz wird automatisch, für den Programmierer *transparent*, bestimmt und reserviert, auch bei Neuzuweisungen wie

```
string s1 = "Hallo";
s1 = "Jetzt aber Platz da!";
```

UMIT
the health & life sciences university

Verkettung von Strings

Strings können mit + verkettet werden:

```
#include <iostream>
#include <string>

using namespace std;

int main (int argc, char* argv[])
{
  string s1 = "Eins";
  string s2 = "Zwei";
  string ausgabe;
  ausgabe = ausgabe + s1 + " und " + s1 + " ist " + s2;

  cout << ausgabe << endl;

  return 0;
}
```

UMIT
the health & life sciences university

Verkettung von Strings

- Die Auswertung mehrerer Verkettungen erfolgt von links nach rechts.
- Mindestens ein Summand muss vom Datentyp `string` sein!
- Selbstverständlich funktioniert auch die Verkettung mit `+=`, etwa

```
ausgabe += s1;
```

UMIT
the health & life sciences university

Vergleich von Strings

Strings können mit <, ==, >, <=, !=, >= verglichen werden (lexikografisch wie bei strcmp()):

```
#include <iostream>
#include <string>

using namespace std;

int main (int argc, char* argv[])
{
  string s2 = "Zwei";
  string s3 = "Drei";

  cout << s2 + "<" + s3 << "ist";
  cout << boolalpha << (s2 < s3) << endl;

  return 0;
}
```

UMIT
the health & life sciences university

Methoden der Klasse `string`

- Die Methode `length()` bestimmt die Länge einer Zeichenkette:

  0 1 2 3 4 5 6 7 8 9 ... 15 16

```
string s1 = "Haller Innbruecke";
cout << s1.length() << endl;
```

  0-Zeichen wird nicht mitgezählt, also 17 hier

  Die Methode `size()` ist für Strings synonym mit `length()`.

- `substr (anf, len)` liefert die Teilzeichenkette (als String) zurück, die ab Position `anf` maximal `len` Zeichen lang ist (kürzer, wenn der aufrufende String nicht lang genug ist).

  s1.substr(7,3) ergibt "Inn" (ab 0 gezählt)
  s1.substr(16,5) ergibt "e"

  ^

UMIT
the health & life sciences university

Zugriff auf einzelne Zeichen einer Zeichenkette

- Auf jedes Zeichen eines Strings kann über seinen Index zugegriffen werden. Wie bei Feldern sind die Indizes ab 0 gezählt.

```
string s1 = "Pirol";
s1[0] = 'T';
cout << s1[0] << s1[1] << s1[2] << s1[3] << s1[4] << endl;
```

- Was passiert bei `char ch = s1[100];` ?

  Es gibt keine Bereichsprüfung beim Feldzugriff! je nachdem was vorher an diesem Speicherplatz war, wenn Zeichenkette nicht 100 Felder lang ist

- Sicherer ist die Benutzung der Methode `at()`:

```
s1.at (0) = 'T';
```

  Hier erfolgt eine Bereichsprüfung: `char ch = s1.at (100);` löst eine *Exception* aus.

Durchsuchen einer Zeichenkette

- `find()` wird zum Suchen und Lokalisieren einer Zeichenfolge in einem String verwendet:

```
string s1 = "Haller Innbruecke";
size_t position = s1.find ("Inn");
```

size_t ... Datentyp der garantiert groß genug ist

  - Dies liefert die Position des ersten Zeichens der gesuchten Folge, hier also 7.
  - Wird die Zeichenkette nicht gefunden, so wird der Wert der Konstante `string::npos` zurückgegeben.
    Man kann also testen:

```
size_t position = s1.find (s2);
if (position == string::npos) {
  ... // Behandlung des "nicht gefunden"-Falles
}
```

    Die Konstante `string::npos` ist von einem vorzeichenlosen (`unsigned`) Ganzzahltyp (`size_type`, im Normalfall ident mit `size_t`) und mit dessen größtmöglichem Wert definiert. Bei einem Vergleich mit `==` ist sie daher gleich `-1`, sodass auch `if (s1.find(s2)==-1) {...}` funktioniert. Der Vergleich `if (s1.find(s2)<0) {...}` schlägt aber fehl!

UMIT
the health & life sciences university

Methoden der Klasse `string`

- `c_str()` liefert ein `char`-Array zurück, in dem der String als C-String (also mit Nullzeichen terminiert) steht – benötigt, wenn der String an C-Funktionen übergeben werden soll

- Zwar steht in der Darstellung eines C++-Strings im Speicher nach dem letzten Zeichen des Strings ebenfalls ein Nullzeichen, dennoch haben Nullzeichen in C++ nicht die Wirkung, die Zeichenkette zu beenden:

```
string s = "Tirol";
cout << s.length() << " " << s << endl; // 5 Tirol
cout << (int) s[5] << endl;             // 0
s[1] = 0;
cout << s.length() << " " << s << endl; // 5 Trol
cout << s.c_str() << endl;              // T
```

In der Hausübung sollen Sie sich mit weiteren Methoden der Klasse `string` vertraut machen, die zum Einfügen, Löschen und Ersetzen von Zeichen dienen.

UMIT
the health & life sciences university

Call by Reference

Alternativ zur Übergabe von Zeigern als Parameter kann in C++ ein *Call by Reference* erzielt werden, indem ein Funktionsargument durch Voranstellen von & als *Referenz* gekennzeichnet wird:

```
void orthodrome (double phi_dep, double lam_dep,
                 double phi_arr, double lam_arr,
                 double &ll, double &azi_dep, double &azi_arr)
{ ... }
```

und in `main()`:

```
orthodrome (phi1, lam1, phi2, lam2, pathangle, azi1, azi2);
```

Call by Reference

- Auch bei der Übergabe als Referenzparameter wird nicht der *Wert* der Parametervariablen, sondern ihre *Speicheradresse* übergeben.
- Analog zur Übergabe mittels Zeiger ist es auch hier nicht möglich, Literale (wie `3`) oder zusammengesetzte Ausdrücke (wie `a+b`) als Referenzparameter zu übergeben.
- **Vorteil gegenüber Zeigern:** Der Adressoperator beim Aufruf der Funktion entfällt. Damit muss der/die Programmierer/in beim Aufruf der Funktion sich in den meisten Fällen keine Gedanken darüber machen, ob die einzelnen Parameter als Werte oder Referenzen übergeben werden!
- Im Unterschied zu Zeigern müssen Referenzen aber stets auf gültige Variablen verweisen, es gibt keine „Null-Referenz“!
- Referenzparameter können auch *read-only* übergeben werden:

```
void print (const string &s);
```

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode verstehen

→`0602-iir.cpp`

- Legen Sie ein neues **C++**-Konsolenprojekt mit dem Quelltext aus der (in Moodle bereit gestellten) Datei `0602-iir.cpp` an
- Dieses Programm filtert Daten aus einer Eingabedatei mittels eines IIR-Filters (infinite impulse response)
- Bringen Sie dieses Programm zum Laufen
- Machen Sie sich anhand dieses Programms die darin vorkommenden weiteren C++-Sprachelemente klar
- Achten Sie auch wieder auf die Include-Dateien

Dieses Projekt benötigt C++11, daher nötigenfalls die entsprechende Compileroption aktivieren!

UMIT
the health & life sciences university

File-Streams

- Ähnlich wie die Streams `cin` und `cout` für die Standardein- und -ausgabe werden auch Streams zur Eingabe aus und Ausgabe in Dateien verwendet.

  Die zugehörigen File-Stream-Klassen sind von `istream` und `ostream` abgeleitet.

- Unterschied zur Standardein-/-ausgabe: Dateien müssen im Dateisystem gefunden, vor dem Lesen oder Schreiben geöffnet und danach wieder geschlossen werden.

  Zwischen dem Öffnen und Schließen stehen grundsätzlich die gleichen Möglichkeiten wie für `cin` und `cout` zur Verfügung:

  - Operationen `<<` und `>>` zum „Streamen“ von Daten (in Textform) in oder aus Dateien,
  - Manipulatoren und Methoden zur Formatierung dieser Daten,
  - Methoden zum Lesen und Schreiben von Zeichen (Bytes) oder Blöcken,
  - Methoden für Statusabfragen.

UMIT
the health & life sciences university

Standard-File-Stream-Klassen

Die folgenden Standard-File-Stream-Klassen werden im Header `fstream` deklariert:

- `ifstream`: abgeleitet von `istream`, dient zum Lesen aus Dateien
- `ofstream`: abgeleitet von `ostream`, dient zum Schreiben in Dateien
- `fstream`: abgeleitet von `iostream`, dient zum Erzeugen, Öffnen und Schließen von Dateien.

Öffnen einer Datei

- Methode `ifstream::open()` bzw. `ofstream::open()` bzw. `fstream::open()`
- Parameter:
  - Dateiname (ggf. mit Pfadangabe)
  - Modus (Lesen/Schreiben/Lesen und Schreiben/Anhängen etc.)
- *Beispiele:*

```
ifstream infile;
ofstream outfile;
infile.open ("Eingabe.txt");
outfile.open ("Ausgabe.txt");
```

  Da kein Modus angegeben wurde, wird der Defaultmodus (Lesen für `ifstream`, Schreiben für `ofstream`) verwendet.

- Beispiel mit Modus:

```
outfile.open ("Ausgabe.txt", ios::app);
```

  Jetzt wird die Datei `Ausgabe.txt` nicht überschrieben, sondern die Ausgabe daran angehängt.

UMIT
the health & life sciences university

Öffnen einer Datei

- Alternativ: Öffnen mit der Deklaration:

```
ifstream infile ("Eingabe.txt");
ofstream outfile ("Ausgabe.txt");
```

- Auch hier kann die Modusangabe hinzutreten:

```
ofstream outfile ("Ausgabe.txt", ios::app);
```

UMIT
the health & life sciences university

Modi beim Öffnen

Modi werden mittels vordefinierter Konstanten des Typs `ios_base::openmode` definiert. Im Prinzip ist das eine ganze Zahl, die als Bitmaske interpretiert wird.

Mehrere Modusangaben können so durch Oder-Verknüpfung verbunden werden.

- `ios::in`
  Bestehende Datei zum Lesen öffnen.
- `ios::out`
  Datei zum Schreiben öffnen. Ohne Kombination mit `ios::in`, `ios::app` oder `ios::ate` wird die Datei gelöscht.
- `ios::app`
  Anhängen: Vor jeder Schreiboperation wird auf das Dateiende positioniert.
- `ios::trunc`
  Abschneiden: Eine bestehende Datei wird beim Öffnen auf Länge 0 gekürzt.
- `ios::ate`
  At End: Positioniere beim Öffnen auf das Dateiende.
- `ios::binary`
  Schreib- und Leseoperationen im Binärmodus. Ohne dieses Flag gilt Textmodus (Steuerzeichen werden ggf. systemabhängig interpretiert)

UMIT
the health & life sciences university

Modi beim Öffnen; Fehlerbehandlung

- Defaultmodi: `ios::in` für `ifstream`, `ios::out | ios::trunc` für `ofstream`
- *Fehlerbit:* Wenn das Öffnen, Schreiben oder Lesen fehlschlägt, wird ein Fehlerbit gesetzt.

  Dies kann durch eine Abfrage der Form

  ```
  if (!infile.good()) {...}
  ```

  getestet werden (prüft, ob das geöffnete File lese-/schreibbar ist).
- *End of file:* Lesefehler können auch durch das Dateiende verursacht sein. Dies kann durch das EOF-Bit getestet werden:

  ```
  if (infile.eof()) {...}
  ```
- Neben `eof()` können konkrete Gruppen von Fehlern spezifisch mit

  `if (infile.fail()) {...}` (explizit),
  `if (!infile) {...}` (implizit, gleichwertig mit `fail()`),
  `if (infile.bad()) {...}`

  überprüft werden (Details siehe Literatur/Referenzen).

Schließen von Dateien

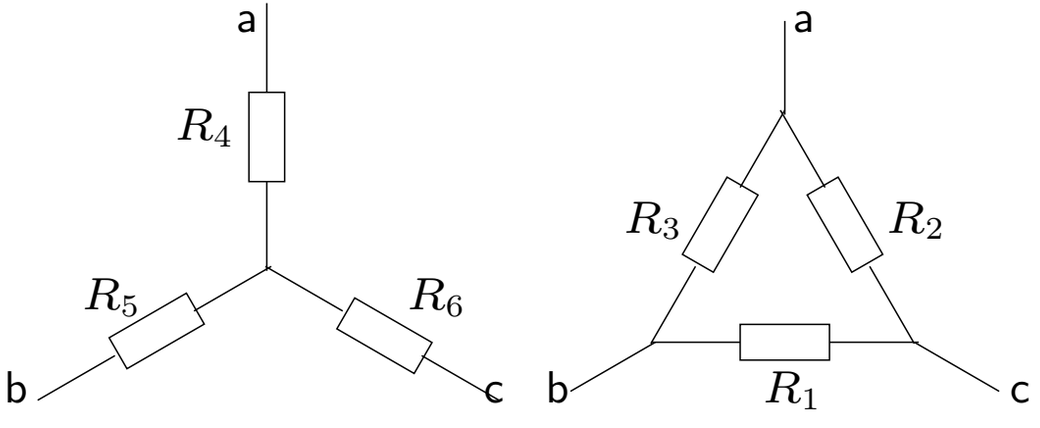
- Wie in C muss auch in C++ eine geöffnete Datei wieder geschlossen werden. Dies sollte möglichst geschehen, sobald das Lesen/Schreiben darin beendet ist, aber spätestens vor dem Programmende. Bei nicht geschlossenen Dateien sind Datenverluste möglich!
- Zum Schließen dient bei allen Filestreams die Methode `close()`.
- Nachdem eine Datei geschlossen wurde, kann die betreffende Streamvariable wieder verwendet werden, um dieselbe oder eine andere Datei zu öffnen.
- Mittels der Methode `is_open()` kann für eine Streamvariable getestet werden, ob ihr eine geöffnete Datei zugeordnet ist.
- Alle geöffneten Dateien werden auch geschlossen, wenn die Funktion `exit()` zum Beenden des Programms aufgerufen wird oder wenn `main()` mittels `return` verlassen wird.

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode schreiben

→0603-rnw (nach der Vorlesung)

Erstellen Sie ein Programm in C++, das Widerstandsnetzwerke in Dreiecks- und Sternschaltung ineinander umrechnet:



$$R_4 = \frac{R_2 R_3}{R_1 + R_2 + R_3} \qquad R_1 = \frac{R_4 R_5 + R_4 R_6 + R_5 R_6}{R_4}$$

$$R_5 = \frac{R_1 R_3}{R_1 + R_2 + R_3} \qquad R_2 = \frac{R_4 R_5 + R_4 R_6 + R_5 R_6}{R_5}$$

$$R_6 = \frac{R_1 R_2}{R_1 + R_2 + R_3} \qquad R_3 = \frac{R_4 R_5 + R_4 R_6 + R_5 R_6}{R_6}$$

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode schreiben

→`0604-calendar` (nach der Vorlesung)

Erstellen Sie ein Programm in C++, das Zeiträume zwischen Kalenderdaten (1901 bis 2099) in Tagen berechnet.

Beispiel: Vom 8. März 1962 bis 4. November 2008 sind 17043 Tage vergangen.

Eingabe erstes Datum: 08.03.1962
Eingabe zweites Datum: 04.11.2008
Differenz: 17043 Tage

Hinweis: Eine Möglichkeit ist, zu jedem Datum die Anzahl der seit 31.12.1900 vergangenen Tage zu berechnen und diese Werte für die beiden eingegebenen Daten zu subtrahieren.