# Programmieren in Fortran 77, C und C++

## Einführung in die Rechnernutzung am GPI

Thomas Forbriger

GPI, Karlsruhe & BFO, Schiltach

Juni 2009 (Update 2016)

# Wozu programmieren?

## Präambel

Selber ein Computer-Programm schreiben zu können ist eine **Schlüsselqualifikation** für das wissenschaftliche Arbeiten, aber nicht nur dafür.

Um grundlegend neue Fragen bearbeiten zu können, muss der Wissenschaftler das dafür geeignete Werkzeug oft zunächst selber herstellen. Zumindest muss er aber in der Lage sein, vorhandene Werkzeuge zu pflegen und zu erweitern.

Wer nicht selber programmieren kann, ist auf andere angewiesen oder kann nur Probleme lösen, die andere bereits zuvor gelöst haben.

# Typische Aufgabenstellungen

## Programmierung findet auf unterschiedlichen Skalen statt

- Mehrere Grafikdateien umbenennen und konvertieren

  Shell-Script
- Prozessierung seismischer Daten mit SeismicUnix

  Shell-Script, Makefile
- Eine Tabelle mit Messwerten auswerten

  awk oder perl Script
- Eine komplizierte Funktion für mehrere Werte ausrechnen

  gnuplot, Matlab, Fortran 77, awk
- Nullstellen einer Funktion suchen

  Matlab, Fortran77

# Typische Aufgabenstellungen

- Messwerte invertieren

  Fortran 77, Matlab
- Synthetische Seismogramme berechnen

  Fortran 77, C oder C++-Programm
- Seismische Daten lesen

  C/C++-Programm

# Ohne Compiler

Shell-Scripte, Makefile

- Verkettung anderer Programme
- Basierend auf Dateien (Eingabedatei $\rightarrow$ Ausgabedatei) oder Pipes

awk, perl

- Verarbeitung von Zeichenketten, die sich in Felder und Records gliedern lassen

gnuplot

- Berechnung von Funktionen
- Darstellung von Daten und umgerechneten Daten
- einfache Kurvenanpassung, Regression

# Matlab (auch ohne Compiler)

+ für umfangreiche Datenanalysen
+ Lineare Algebra (Lineare Gleichungssysteme)
+ komfortable Grafik-Ausgabe
+ interaktive Oberfläche
- Daten Eingabe aus ASCII-Tabellen
- binäre Ein-/Ausgabe ist programmierbar
- Einbindung von externem Binärcode ist möglich
– erfordert Lizenz
– nur bedingt portierbar
– größere CPU-Zeiten als optimierter Binärcode

# Fortran 77

## Compiler-Sprachen Fortran 77, C und C++

- \+ einfache Programmierung von Rechenoperationen
- \+ umfangreiche Bibliotheken für Numerik
- \+ erreicht potentiell die höchste CPU-Effizienz
- – unflexible Speicherverwaltung
- – keine Daten-Strukturen definierbar
- – Zugriff auf Binärdaten ist systemabhängig
- – nur mühsam und unsicher modulierbar

Fortran 90 und Fortran 95 bieten dynamische Speicherverwaltung und Datenstrukturen sowie Matrix und Vektortypen und Operatoren für lineare Algebra. Die genannten negativen Eigenschaften und Einschränkungen von Fortran 77 wurden in den neueren Standards überwunden (siehe `https://en.wikipedia.org/wiki/Fortran`).

# C

## Compiler-Sprachen Fortran 77, C und C++

+ flexible Speicherverwaltung
+ Daten-Strukturen sind definierbar
+ einfacher Zugriff auf Binärdaten
+ erreicht hohe CPU-Effizienz
– keine mehrdimensionalen Arrays
– keine komplexe Arithmetik (erst ab C99)
– Arbeit mit Pointern auf relativ elementarer Ebene

# C++

## Compiler-Sprachen Fortran 77, C und C++

- \+ abstrakte Programmierung (Templates)
- \+ mächtige Unterstützung modularer Programmierung
- \+ komplexe Arithmetik, Vektor-Arithmetik
- \+ const-Correctness
- \+ elegante, flexible Speicherverwaltung
- \+ Quellcode und Binärcode kompatibel zu C
- \+ erreicht gute CPU-Effizienz
- – noch kein etablierter Standard für Arrays und numerische Anwendungen

# Compiler-Sprachen Fortran 77, C und C++

Für Fortran 77, C und C++ gibt es frei verfügbare und gleichzeitig sehr leistungsstarke Compiler für nahezu jede Plattform (GNU Compiler Projekt und Netlib).

Der GNU Fortran Compiler (`gfortran`, Version 6.2.0) unterstützt die Standards Fortran 90 und Fortran 95 vollständig und die neueren Standards (2003 und 2008) teilweise.
Siehe `https://gcc.gnu.org/fortran/`.

# Vom Quellcode zum ausführbaren Programm

## Ein einfaches Beispiel

C Quellcode `hello.c`:

```
#include<stdio.h>
#define HELLOSTRING "Hello World!\n"
int main()
{
  printf(HELLOSTRING);
  return(0);
}
```

# Vom Quellcode zum ausführbaren Programm

## Ein einfaches Beispiel

Übersetzen des Quellcodes mit dem C Compiler:

```
thof@lizzy:~> gcc hello.c
```

Übersetztes Programm
(sogenanntes *Binary* oder *Executable*):

```
thof@lizzy:~> ls -l a.out | fold -w 29
-rwxr-xr-x 1 thof users 12545
 Nov  4 10:17 a.out
```

Ausführen des Programms:

```
thof@lizzy:~> a.out
Hello World!
```

# Vom Quellcode zum ausführbaren Programm

## Beteiligte Programme

Eingabe: Textdatei (Quellcode)

An der Compilierung beteiligte Programme:

1. Preprocessor
2. Compiler
3. Optimierer
4. Assembler
5. Linker

Ausgabe: Ausführbares Programm (Binary)

# Vom Quellcode zum ausführbaren Programm

## Ein einfaches Beispiel (alle Zwischenschritte)

Quellcode `hello.c`:

```
#include<stdio.h>
#define HELLOSTRING "Hello World!\n"
int main()
{
  printf(HELLOSTRING);
  return(0);
}
```

# Vom Quellcode zum ausführbaren Programm

## Ein einfaches Beispiel (alle Zwischenschritte)

Quellcode durch Preprocessor vorbereiten:

```
thof@lizzy:~> gcc -E -o hello.i hello.c
thof@lizzy:~> tail -8 hello.i

# 2 "hello.c" 2

int main()
{
  printf("Hello World!\n");
  return(0);
}
```

## Übersetzen des Quellcodes in Assemblercode (compiling):

```
thof@lizzy:~> gcc -S -o hello.s hello.i
```

## Assemblercode `hello.s`:

```
.file "hello.c"
.section .rodata
.LC0:
.string "Hello World!"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $.LC0, %edi
call puts
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (SUSE Linux) 4.8.3 20140627 [gcc-4_8-branch revision 212064]"
.section .note.GNU-stack,"",@progbits
```

# Vom Quellcode zum ausführbaren Programm

## Ein einfaches Beispiel (alle Zwischenschritte)

Übersetzen des Assemblercode in Binärcode (assembling):

```
thof@lizzy:~> gcc -c -o hello.o hello.s
```

# Ausgabe des Binärcodes:

```
thof@lizzy:~> od -A x -t x1z hello.o | head -30
000000 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  >.ELF............<
000010 01 00 3e 00 01 00 00 00 00 00 00 00 00 00 00 00  >..>.............<
000020 00 00 00 00 00 00 00 00 48 01 00 00 00 00 00 00  >........H.......<
000030 00 00 00 00 40 00 00 00 00 00 40 00 0d 00 0a 00  >....@.....@.....<
000040 55 48 89 e5 bf 00 00 00 00 e8 00 00 00 00 b8 00  >UH..............<
000050 00 00 00 5d c3 48 65 6c 6c 6f 20 57 6f 72 6c 64  >...].Hello World<
000060 21 00 00 47 43 43 3a 20 28 53 55 53 45 20 4c 69  >!..GCC: (SUSE Li<
000070 6e 75 78 29 20 34 2e 38 2e 33 20 32 30 31 34 30  >nux) 4.8.3 20140<
000080 36 32 37 20 5b 67 63 63 2d 34 5f 38 2d 62 72 61  >627 [gcc-4_8-bra<
000090 6e 63 68 20 72 65 76 69 73 69 6f 6e 20 32 31 32  >nch revision 212<
0000a0 30 36 34 5d 00 00 00 00 14 00 00 00 00 00 00 00  >064]............<
0000b0 01 7a 52 00 01 78 10 01 1b 0c 07 08 90 01 00 00  >.zR..x..........<
0000c0 1c 00 00 00 1c 00 00 00 00 00 00 00 15 00 00 00  >................<
0000d0 00 41 0e 10 86 02 43 0d 06 50 0c 07 08 00 00 00  >.A....C..P......<
0000e0 00 2e 73 79 6d 74 61 62 00 2e 73 74 72 74 61 62  >..symtab..strtab<
0000f0 00 2e 73 68 73 74 72 74 61 62 00 2e 72 65 6c 61  >..shstrtab..rela<
000100 2e 74 65 78 74 00 2e 64 61 74 61 00 2e 62 73 73  >.text..data..bss<
000110 00 2e 72 6f 64 61 74 61 00 2e 63 6f 6d 6d 65 6e  >..rodata..commen<
000120 74 00 2e 6e 6f 74 65 2e 47 4e 55 2d 73 74 61 63  >t..note.GNU-stac<
000130 6b 00 2e 72 65 6c 61 2e 65 68 5f 66 72 61 6d 65  >k..rela.eh_frame<
000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  >................<
*
000180 00 00 00 00 00 00 00 00 20 00 00 00 01 00 00 00  >........ .......<
000190 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  >................<
0001a0 40 00 00 00 00 00 00 00 15 00 00 00 00 00 00 00  >@...............<
0001b0 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  >................<
0001c0 00 00 00 00 00 00 00 00 1b 00 00 00 04 00 00 00  >................<
0001d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  >................<
0001e0 a8 05 00 00 00 00 00 00 30 00 00 00 00 00 00 00  >........0.......<
0001f0 0b 00 00 00 01 00 00 00 08 00 00 00 00 00 00 00  >................<
```

# Ausgabe des Disassemblers:

```
thof@lizzy:~> objdump -s -d hello.o

hello.o:     file format elf64-x86-64

Contents of section .text:
 0000 554889e5 bf000000 00e80000 0000b800  UH..............
 0010 0000005d c3                          ...].
Contents of section .rodata:
 0000 48656c6c 6f20576f 726c6421 00        Hello World!.
Contents of section .comment:
 0000 00474343 3a202853 55534520 4c696e75  .GCC: (SUSE Linu
 0010 78292034 2e382e33 20323031 34303632  x) 4.8.3 2014062
 0020 37205b67 63632d34 5f382d62 72616e63  7 [gcc-4_8-branc
 0030 68207265 76697369 6f6e2032 31323036  h revision 21206
 0040 345d00                               4].
Contents of section .eh_frame:
 0000 14000000 00000000 017a5200 01781001  .........zR..x..
 0010 1b0c0708 90010000 1c000000 1c000000  ................
 0020 00000000 15000000 00410e10 8602430d  .........A....C.
 0030 06500c07 08000000                    .P......

Disassembly of section .text:

0000000000000000 <main>:
   0: 55                    push   %rbp
   1: 48 89 e5              mov    %rsp,%rbp
   4: bf 00 00 00 00        mov    $0x0,%edi
   9: e8 00 00 00 00        callq  e <main+0xe>
   e: b8 00 00 00 00        mov    $0x0,%eax
  13: 5d                    pop    %rbp
  14: c3                    retq
```

# Vom Quellcode zum ausführbaren Programm

## Ein einfaches Beispiel (alle Zwischenschritte)

Zusammenfügen (binden) zum ausführbaren Programm (linking):

```
thof@lizzy:~> gcc -o hello hello.o
```

Ausführen des Programms:

```
thof@lizzy:~> hello
Hello World!
```

# Endungen und ihre Bedeutungen

| | |
|---|---|
| `.c` | C Quellcode |
| `.cpp, .C, .cc, .cxx` | C++ Quellcode |
| `.f` | Fortran 77 Quellcode |
| `.f90` | Fortran 90/95 Quellcode |
| `.i` | präprozessierter C Quellcode |
| `.ii` | präprozessierter C++ Quellcode |
| `.h` | C Header Datei |
| `.hh, .H, .h` | C++ Header Datei |
| `.s` | Assembler Quellcode |
| `.o` | Object File (Binärcode) |
| `.a` | Archiv (statisch) |
| `.so` | Bibliothek (dynamisch) |

Die Endungen entsprechen den üblichen Konventionen. Ihre Verwendung ist nicht zwingend. Es gibt abweichende Konventionen.

# Ausgewählte Compiler Frontends

| | |
|---|---|
| `gcc` | C Compiler in GNU Compiler Collection |
| `g++` | C++ Compiler in GNU Compiler Collection |
| `gfortran` | Fortran 95 Compiler in GNU Compiler Collection |
| `g77` | Fortran 77 Compiler in GNU Compiler Collection (wird nicht mehr gepflegt) |
| `cc` | C SGI IRIX, HP_UX,... |
| `CC` | C++ SGI IRIX, HP_UX,... |
| `f77` | Fortran 77 SGI IRIX, HP_UX,... |
| `f90` | Fortran 90/95 SGI IRIX, HP_UX,... |
| `icc` | C/C++ Intel compiler |
| `ifc` | Fortran 95 Intel compiler |
| `pgcc` | C/C++ Portland group compiler |
| `pgf77` | Fortran 77 Portland group compiler |
| `f2c` | Fortran 77 Konverter (nach C) |

# Ausgewählte Optionen und Parameter

| | |
|---|---|
| `-c` | Übersetzung vor dem Linken beenden |
| `-o out` | Ergebnis in Datei `out` schreiben |
| `-g` | Binde debug-Information in den Code ein |
| `-Wall` | Warnung bei Abweichung vom Standard |
| `-O` | Optimiere die Laufzeit-Effizienz |
| `-I dir` | Suche Header auch in `dir` |
| `-L dir` | Suche Bibliotheken auch in `dir` |
| `-lname` | Linke gegen `libname.a` |
| `-static` | Verwende keine dynamischen Bibliotheken |

# Ein Beispiel in C, C++ und Fortran 77

## Aufgabenstellung

Schreiben Sie ein Programm, das

1. die Koeffizienten eines Polynoms vom Grad zwei einliest,
2. das Polynom als Formel ausgibt,
3. prüft, ob komplexe Nullstellen vorliegen und
4. die Nullstellen berechnet und ausgibt.

Gliederen Sie allgemein lösbare Aufgaben in Unterprogramme aus, so dass diese auch von anderen Programmen verwendet werden können.

# Ein Beispiel in C, C++ und Fortran 77

## Generalisierbare Module

Für Polynome beliebigen Grades:

- Speichern der Koeffizienten
- Ausgabe der Formel

Für Polynome zweiten Grades:

- Berechnung der Diskriminante
- Prüfung auf komplexe Nullstellen
- Berechnung der Nullstellen

# Besonderheiten von Fortran

## Fossile Sprachelemente aus der Zeit der Lochkarten

Fortran 77 wurde in der Blütezeit der Lochkarten entwickelt. Deshalb entspricht jede Position einer **Programmzeile** einer Position auf einer **Lochkarte**. In Fortran 77 ist es **entscheidend, an welcher Position ein Zeichen steht**. Das gilt nicht nur für die **Quelltexte**, sondern häufig auch für die **Ein- und Ausgabe von Programmen**. Mehrere Zahlen in einer Eingabezeile werden dann nicht durch Trennzeichen (Leerzeichen oder Kommas) voneinander getrennt. Sie sind vielmehr durch ihre exakte Position in der Zeile definiert und dürfen nahtlos ineinander übergehen. Ist eine Zahl bezüglich der erwarteten Position verrutscht, wird sie nicht korrekt eingelesen!

# Besonderheiten von Fortran

## Aufbau einer Programmzeile/Lochkarte mit Quellcode

Nur die Positionen 1–72 dürfen für Quellcode verwendet werden.
Weitere Zeichen werden ignoriert.
Bedeutung der Positionen:

1 **Kommentarzeile:** Ein ’C’ oder ein ’*’ an dieser Position kennzeichnet die Zeile als Kommentar. Sie wird vom Compiler ignoriert.

1–5 **Anweisungsmarken:** Diese Positionen können ein- bis fünfstellige Zahlen enthalten, die im Programm als Sprungziele verwendet werden.

6 **Fortsetzungszeichen:** Ein Zeichen an dieser Position zeigt an, dass der folgende Programmtext die vorangegangene Zeile fortsetzt. Auf diese Weise kann eine Programmzeile um insgesamt 19 Zeilen verlängert werden.

7–72 **Programmtext**

# Besonderheiten von Fortran

## Dialekte und Abweichungen vom Standard

Der Fortran Standard schreibt vor:

- Namen von Variablen und Unterprogrammen dürfen maximal 6 Zeichen lang sein.
- Es dürfen nur Großbuchstaben verwendet werden.
- Schleifen sind nur über Sprünge zu Anweisungsmarken möglich.
- ...

Die meisten Compiler bieten Möglichkeiten, die von den Einschränkungen des Standards abweichen. Es gibt dadurch mehrere *Fortran Dialekte*. Kaum ein Fortran Programm ist vollständig konform zum Standard.

# Besonderheiten von Fortran

## Ist Fortran eine tote Sprache — das Latein unter den Programmiersprachen?

Wozu Fortran lernen?

In Fortran existiert eine riesige Menge wertvollen und gut getesteten Programmcodes. Diesen Code in eine andere Sprache zu übertragen und erneut zu testen ist ineffizient und würde Jahre dauern. Daher ist jeder wissenschaftliche Programmierer darauf angewiesen, diesen Code zumindest zu verstehen, um ihn in seine Programme einbinden zu können.

Da wichtige Datentypen (komplexe Zahlen, Matrix, Vektor) direkt in den neueren Standards Fortran 90 und Fortran 95 definiert sind, können die Compiler effizienter optimieren, als in Sprachen in denen diese Typen über Bibliotheken hinzugefügt werden.

# Lösung in Fortran 77

## Zutaten

```
thof@lizzy:~> ls -l polynomf.f polyf.f | cut -c 1-13,37-80
-rw-r--r-- 1 13:37 polyf.f
-rw-r--r-- 1  2015 polynomf.f
```

# Lösung in Fortran 77

`polyf.f` (lines 1 to 20):

```
c this is <polyf.f>
c ----------------------------------------------------
c $Id$
c find roots of polynomial
c ====================================================
c
      program polyf
c
c declare variables and arrays
      integer m
      parameter(m=3)
      double precision c(m), c0, c1, c2
c declare function return types
      double precision discriminant
      double complex root
      logical iscomplex
c declare functions
      discriminant(c0,c1,c2)=c1**2-4.*c0*c2
      iscomplex(c0,c1,c2)=(discriminant(c0,c1,c2).lt.0.)
c read polynomial coefficients and report function to terminal
[...]
```

# Lösung in Fortran 77

`polyf.f` (lines 20 to end):

```
[...]
c read polynomial coefficients and report function to terminal
      print 50,'Please enter polynomial coefficients (c0, c1, c2):'
      read(5, *) (c(i), i=1,3)
      call printpolynomial(c, m)
c report complex roots
      if (iscomplex(c(1),c(2),c(3)))
     &  print 50,'Polynomial has complex roots'
c report roots
      print 51, 1, root(c, m, .true.)
      print 51, 2, root(c, m, .false.)
      stop
c define output formats
   50 format(a)
   51 format(6hroot #,i1,3h: (,f8.3,2h, ,f8.3,1h))
      end
c
c ----- END OF polyf.f -----
```

# Lösung in Fortran 77

`polynomf.f` (lines 1 to 18):

```
c this is <polynomf.f>
c ----------------------------------------------------
c $Id$
c subroutines for root finding
c ====================================================
c

c print polynomial function to terminal
c -----------------------------------
      subroutine printpolynomial(c, m)
      integer m, i
      double precision c(m)
      if (m.ne.3) stop 'ERROR: polynomial is not of degree 2!'
      print 50, (c(i), i=3,1,-1)
      return
   50 format(7hf(x) = ,f6.3,9h *x**2 + ,f6.3,6h *x + ,f6.3)
      end

[...]
```

# Lösung in Fortran 77

`polynomf.f` (lines 19 to end):

```
[...]
c calculate and return root
c returns first root if f is .true.
c --------------------------------
      double complex function root(c, m, f)
      integer m
      double precision c(m)
      logical f
      double precision factor
c select root
      factor=-1.
      if (f) factor=-factor
c check for appropriate degree of 2
      if (abs(c(3)) .lt. 1.d-100)
     &  stop 'ERROR: polynomial effectively is not of degree two!'
      if (m.ne.3) stop 'ERROR: polynomial is not of degree two!'
c calculate root
      root = (-c(2)+factor*sqrt(c(2)**2-(4.,0.)*c(1)*c(3)))/(2.*c(3))
      return
      end
c
c ----- END OF polynomf.f -----
```

# Lösung in Fortran 77

## Ergebnis

```
thof@lizzy:~> gfortran -Wall -c -o polynomf.o polynomf.f
thof@lizzy:~> gfortran -Wall -c -o polyf.o polyf.f
thof@lizzy:~> gfortran -o polyf polyf.o polynomf.o
thof@lizzy:~> echo 1. 2. 3. | polyf
Please enter polynomial coefficients (c0, c1, c2):
f(x) =  3.000 *x**2 +  2.000 *x +  1.000
Polynomial has complex roots
root #1: (  -0.333,    0.471)
root #2: (  -0.333,   -0.471)
thof@lizzy:~> echo 0. -2. 4. | polyf
Please enter polynomial coefficients (c0, c1, c2):
f(x) =  4.000 *x**2 + -2.000 *x +  0.000
root #1: (   0.500,    0.000)
root #2: (   0.000,    0.000)
thof@lizzy:~> echo 1. 2. 0. | polyf
Please enter polynomial coefficients (c0, c1, c2):
f(x) =  0.000 *x**2 +  2.000 *x +  1.000
STOP ERROR: polynomial effectively is not of degree two!
```

# Lösung in C

## Zutaten

```
thof@lizzy:~> ls -l polynom_discriminant.c polynom_roots.c polynom_iscomplex.c poly
-rw-r--r-- 1  2015 poly.c
-rw-r--r-- 1  2015 polynom.h
-rw-r--r-- 1  2015 polynom_discriminant.c
-rw-r--r-- 1  2015 polynom_iscomplex.c
-rw-r--r-- 1  2015 polynom_printpolynomial.c
-rw-r--r-- 1  2015 polynom_roots.c
```

# Lösung in C

`polynom.h` (lines 1 to 23):

```
/*! \file polynom.h
 * ------------------------------------------------------------------------
 * $Id$
 * struct definition and function prototypes to handle polynomials (prototypes)
 * ========================================================================
 */

/* include guard */
#ifndef TF_POLYNOM_H_VERSION
#define TF_POLYNOM_H_VERSION

#include<stdio.h>

/* represent a complex number */
struct dcomplex {
  double real, imag;
};

/* represent a polynomial by its coefficients */
struct polynomial {
  double c0, c1, c2;
};

[...]
```

# Lösung in C

`polynom.h` (lines 24 to end):

```
[...]
/* returns 1 if complex roots are present */
int iscomplex(struct polynomial p);

/* print polynomial definition to file descriptor fd */
int printpolynomial(FILE* fd, struct polynomial p);

/* calculate roots (returns 1 in case of polynomial being not of degree two */
int roots(struct dcomplex* r1, struct dcomplex* r2, struct polynomial p);

/* return discriminat of polynomial */
double discriminant(struct polynomial p);

#endif /* TF_POLYNOM_H_VERSION (includeguard) */

/* ----- END OF polynom.h ----- */
```

# Lösung in C

`poly.c` (lines 1 to 20):

```
/*! \file poly.c
 * $Id$
 * find roots of polynomial
 * ========================================================
 */

/* include struct definition and prototypes of functions */
#include "polynom.h"
/* needs function abort from libstdc */
#include <stdlib.h>
/* needs functions printf and fprintf */
#include <stdio.h>

int main()
{
  /* define variables we need */
  struct polynomial p;
  struct dcomplex r1, r2;

  /* read polynomial coefficients and print to stdout */
[...]
```

# Lösung in C

`poly.c` (lines 21 to end):

```
[...]
  printf("Please enter polynomial coefficients (c0, c1, c2):\n");
  scanf("%lf %lf %lf", &p.c0, &p.c1, &p.c2);
  printpolynomial(stdout, p);

  /* check for complex roots and report if present */
  if (iscomplex(p))
  {
    printf("Polynomial has complex roots\n");
  }

  /* calculate roots and abort if polynomial is not of degree two */
  if (roots(&r1, &r2, p) != 0)
  {
    fprintf(stderr, "ERROR: polynomial is not of degree two!\n");
    abort();
  }

  /* report roots */
  printf("First root: (%g, %g)\n", r1.real, r1.imag);
  printf("Second root: (%g, %g)\n", r2.real, r2.imag);

  /* indicate success */
  return(0);
}

/* ----- END OF poly.c ----- */
```

# Lösung in C

`polynom_printpolynomial.c` (lines 1 to end):

```
/*! \file polynom_printpolynomial.c
 * ------------------------------------------------------
 * print polynomial (implementation)
 * ======================================================
 */

/* include struct definition and check consistency with prototypes */
#include "polynom.h"
/* needs function fprintf */
#include <stdio.h>

/* print polynmial to fd (always returns 0)
 * --------------------
 */
int printpolynomial(FILE* fd, struct polynomial p)
{
  fprintf(fd, "f(x)= %g *x^2 + %g *x + %g\n", p.c2, p.c1, p.c0);
  return(0);
} /* int printpolynomial(FILE* fd, struct polynomial p) */

/* ----- END OF polynom_printpolynomial.c ----- */
```

# Lösung in C

`polynom_discriminant.c` (lines 1 to end):

```
/*! \file polynom_discriminant.c
 * ------------------------------------------------------
 * $Id$
 * calculate discriminant (implementation)
 * ======================================================
 */

/* include struct definition and check consistency with prototypes */
#include "polynom.h"

/* return discriminant of polynomial
 * -------------------------------
 */
double discriminant(struct polynomial p)
{
  double result;
  result = p.c1*p.c1-4.*p.c0*p.c2;
  return(result);
} /* double discriminant(struct polynomial p) */

/* ----- END OF polynom_discriminant.c ----- */
```

# Lösung in C

`polynom_iscomplex.c` (lines 1 to end):

```
/*! \file polynom_iscomplex.c
 * ------------------------------------------------------
 * check for complex roots (implementation)
 * ======================================================
 */

/* include struct definition and check consistency with prototypes */
#include "polynom.h"

/* return 1 if polynomial has complex roots
 * ----------------------------------------
 */
int iscomplex(struct polynomial p)
{
  int result;
  result = discriminant(p) < 0 ? 1 : 0;
  return(result);
} /* int iscomplex(struct polynomial p) */

/* ----- END OF polynom_iscomplex.c ----- */
```

# Lösung in C

`polynom_roots.c` (lines 1 to 21):

```
/*! \file polynom_roots.c
 * ------------------------------------------------------
 * $Id$
 * calculate roots of polynomial (implementation)
 * ======================================================
 */

/* include struct definition and check consistency with prototypes */
#include "polynom.h"
/* needs function sqrt from libm.a */
#include <math.h>
/* needs function abs from libstdc */
#include <stdlib.h>

/* calculate roots of polynomial
 * -----------------------------
 * returns 1 if polynomial is not of degree two
 */
int roots(struct dcomplex* r1, struct dcomplex* r2, struct polynomial p)
{
  /* define variables */
[...]
```

# Lösung in C

`polynom_roots.c` (lines 19 to 39):

```
[...]
int roots(struct dcomplex* r1, struct dcomplex* r2, struct polynomial p)
{
  /* define variables */
  double d;
  int result;
  result=1;

  /* calculate roots only for degree two polynomial */
  if (fabs(p.c2) > 1.e-100)
  {
    /* indicate success */
    result=0;

    /* normalize polynomial */
    p.c0 /= (2.*p.c2);
    p.c1 /= (2.*p.c2);
    p.c2 = 0.5;
    /* calculate part independent of discriminant */
    r1->real = -p.c1;
    r2->real = -p.c1;

[...]
```

# Lösung in C

`polynom_roots.c` (lines 40 to end):

```
[...]
    /* distinguish between positive and negative discriminat */
    d = discriminant(p);
    if (d < 0.)
    {
      /* complex roots */
      r1->imag = sqrt(-d);
      r2->imag = -sqrt(-d);
    }
    else
    {
      /* real roots */
      r1->real += sqrt(d);
      r2->real -= sqrt(d);
      r1->imag = 0;
      r2->imag = 0;
    }
  }

  /* indicate success or error */
  return(result);
} /* int roots(struct dcomplex* r1, struct dcomplex* r2, struct polynomial p) */

/* ----- END OF polynom_roots.c ----- */
```

# Lösung in C

## Ergebnis

```
thof@lizzy:~> gcc -Wall -c -o polynom_roots.o polynom_roots.c
thof@lizzy:~> gcc -Wall -c -o polynom_iscomplex.o polynom_iscomplex.c
thof@lizzy:~> gcc -Wall -c -o polynom_printpolynomial.o polynom_printpolynomial.c
thof@lizzy:~> gcc -Wall -c -o polynom_discriminant.o polynom_discriminant.c
thof@lizzy:~> gcc -Wall -c -o poly.o poly.c
thof@lizzy:~> gcc -o poly poly.o polynom_*.o -lm
thof@lizzy:~> echo 1. 2. 3. | poly
Please enter polynomial coefficients (c0, c1, c2):
f(x)= 3 *x^2 + 2 *x + 1
Polynomial has complex roots
First root: (-0.333333, 0.471405)
Second root: (-0.333333, -0.471405)
thof@lizzy:~> echo 0. -2. 4. | poly
Please enter polynomial coefficients (c0, c1, c2):
f(x)= 4 *x^2 + -2 *x + 0
First root: (0.5, 0)
Second root: (0, 0)
thof@lizzy:~> echo 1. 2. 0. | poly
ERROR: polynomial is not of degree two!
/bin/bash: line 1:  5289 Done                    echo 1. 2. 0.
      5290 Aborted                 | poly
```

# Lösung in C++

## Zutaten

```
thof@lizzy:~> ls -l polynomxx.h polynomxx.cc polyxx.cc | cut
-rw-r--r-- 1  2015 polynomxx.cc
-rw-r--r-- 1 13:37 polynomxx.h
-rw-r--r-- 1  2015 polyxx.cc
```

# Lösung in C++

`polynomxx.h` (lines 1 to 18):

```
/*! \file polynomxx.h
 * ------------------------------------------------------
 * $Id$
 * A class to handle polynomials (prototypes)
 * ======================================================
 */

// include guard
#ifndef TF_POLYNOMXX_H_VERSION
#define TF_POLYNOMXX_H_VERSION

#include<complex>
#include<iostream>

// Code to handle polynomials is placed in module mymath
namespace mymath {

  /* Class template to hold coefficients for a polynomial of arbitrary degree N
[...]
```

# Lösung in C++

`polynomxx.h` (lines 18 to 46):

```
[...]
  /* Class template to hold coefficients for a polynomial of arbitrary degree N
   * and numerical type C
   * ------------------------------------------------------------------------
   */
  template<int N, class C>
  class Polynomial {
    public:
      // provide numerical type
      typedef C Tvalue;
      // provide degree
      static const int degree=N;

      // exception class
      class IllegalCoefficient { };

      // constructor initializes all coefficients to the same value
      Polynomial(const Tvalue& v=0)
      { for(int i=0; i<=degree; ++i) { Mc[i]=v; } }

      // read access
      C c(const int& i) const throw(IllegalCoefficient)
      {
        checkindex(i);
        return(Mc[i]);
      }
      // read access operator
      C operator()(const int& i) const throw(IllegalCoefficient)
      { return(c(i)); }
      // write access
[...]
```

# Lösung in C++

`polynomxx.h` (lines 46 to 70):

```
[...]
      // write access
      C& c(const int& i) throw(IllegalCoefficient)
      {
        checkindex(i);
        return(Mc[i]);
      }
      // write access operator
      C& operator()(const int& i) throw(IllegalCoefficient) { return(c(i)); }

    private:
      // polynomial coefficients
      C Mc[N+1];
      // check index range and abort if illegal
      void checkindex(const int& i) const throw(IllegalCoefficient)
      {
        if ((i<0) || (i>degree))
        {
          std::cerr << "ERROR: illegal index to coefficient: "
            << i << " for polynomial of degree" << degree << "!" << std::endl;
          throw IllegalCoefficient();
        }
      }
  }; // class Polynomial<N, C>

  /* output operator to print polynomial
[...]
```

# Lösung in C++

`polynomxx.h` (lines 70 to 82):

```
[...]
  /* output operator to print polynomial
   * ----------------------------------
   */
  template<int N, class C>
  std::ostream& operator<<(std::ostream& os, const Polynomial<N, C>& p)
  {
    os << "f(x) = ";
    for (int i=N; i>0; --i)
    { os << p(i) << " * x^" << i << " + "; }
    os << p(0) << std::endl;
    return(os);
  } // std::ostream& operator<<(std::ostream& os, const Polynomial<N, C>& p)

[...]
```

# Lösung in C++

`polynomxx.h` (lines 82 to 99):

```
[...]

  /* Polynomial of degree 2 and type double
   * --------------------------------------
   */
  class DPoly2: public Polynomial<2, double>
  {
    public:
      // provide base class type
      typedef Polynomial<2, double> Tbase;
      // provide valeu type
      typedef Tbase::Tvalue Tvalue;
      // provide type of root values
      typedef std::complex<Tvalue> Trootvalue;

      // exception class
      class NotOfDegree2 { };

      // constructor initializes all coefficients
[...]
```

# Lösung in C++

`polynomxx.h` (lines 99 to end):

```
[...]
      // constructor initializes all coefficients
      DPoly2(const Tvalue& v=0): Tbase(v) { };
      // return true if polynomial has complex roots
      bool iscomplex() const { return(this->discriminant() < 0.); }

      // check effective degree
      void checkdegree() const throw(NotOfDegree2);
      // return discriminant
      Tvalue discriminant() const;
      // return first root value
      Trootvalue root1() const throw(NotOfDegree2);
      // return second root value
      Trootvalue root2() const throw(NotOfDegree2);
  }; // class DPoly2

} // namespace mymath

#endif // TF_POLYNOMXX_H_VERSION (includeguard)

/* ----- END OF polynomxx.h ----- */
```

# Lösung in C++

## `polynomxx.cc` (lines 1 to 27):

```
/*! \file polynomxx.cc
 * ------------------------------------------------------
 * $Id$
 * A class to handle polynomials (implementation)
 * ======================================================
 */

#include "polynomxx.h"

namespace mymath {

  // calculate and return discriminant
  DPoly2::Tvalue DPoly2::discriminant() const
  {
    Tvalue result = (c(1)*c(1) - 4.*c(2)*c(0));
    return(result);
  }

  // calculate and return first root value
  DPoly2::Trootvalue DPoly2::root1() const throw(NotOfDegree2)
  {
    checkdegree();
    Trootvalue result
      = (-c(1) + std::sqrt(Trootvalue(discriminant())))/(2.*c(2));
    return(result);
  }

[...]
```

# Lösung in C++

`polynomxx.cc` (lines 28 to end):

```
[...]
  // calculate and return second root value
  DPoly2::Trootvalue DPoly2::root2() const throw(NotOfDegree2)
  {
    checkdegree();
    Trootvalue result
      = (-c(1) - std::sqrt(Trootvalue(discriminant())))/(2.*c(2));
    return(result);
  }

  // check effective polynomial degree
  void DPoly2::checkdegree() const throw(NotOfDegree2)
  {
    if (std::abs(c(2)) < 1.e-100)
    {
      std::cerr << "ERROR: effective degree of polynomial is less than 2!"
        << std::endl;
      std::cerr << "       c(2) = " << c(2) << std::endl;
      throw NotOfDegree2();
    }
  }

} // namespace mymath

/* ----- END OF polynomxx.cc ----- */
```

# Lösung in C++

`polyxx.cc` (lines 1 to 18):

```
/*! \file polyxx.cc
 * ------------------------------------------------------
 * $Id$
 * find roots of polynomial
 * ======================================================
 */

// use std::cout and std::cin
#include <iostream>
// use class definition for polynomial
#include "polynomxx.h"

// use standard input/output streams without namespace prefix
using std::cout;
using std::cin;
using std::endl;

int main()
[...]
```

# Lösung in C++

`polyxx.cc` (lines 18 to end):

```
[...]
int main()
{
  // create polynomial and read coefficients
  cout << "Please enter polynomial coefficients (c0, c1, c2):" << endl;
  mymath::DPoly2 p;
  cin >> p(0) >> p(1) >> p(2);
  // report polynomial to stdout
  cout << p;
  // check for complex roots and report
  if (p.iscomplex()) { cout << "Polynomial has complex roots" << endl; }
  // report roots and catch exception
  try {
    cout << "First root: " << p.root1() << endl;
    cout << "Second root: " << p.root2() << endl;
  }
  catch (mymath::DPoly2::NotOfDegree2)
  {
    cout << "The coefficient of order two is too small: " << p(2) << endl;
  }
}

/* ----- END OF polyxx.cc ----- */
```

# Lösung in C++

## Ergebnis

```
thof@lizzy:~> g++ -Wall -c -o polynomxx.o polynomxx.cc
thof@lizzy:~> g++ -Wall -c -o polyxx.o polyxx.cc
thof@lizzy:~> g++ -o polyxx polyxx.o polynomxx.o
thof@lizzy:~> echo 1. 2. 3. | polyxx
Please enter polynomial coefficients (c0, c1, c2):
f(x) = 3 * x^2 + 2 * x^1 + 1
Polynomial has complex roots
First root: (-0.333333,0.471405)
Second root: (-0.333333,-0.471405)
thof@lizzy:~> echo 0. -2. 4. | polyxx
Please enter polynomial coefficients (c0, c1, c2):
f(x) = 4 * x^2 + -2 * x^1 + 0
First root: (0.5,0)
Second root: (0,-0)
thof@lizzy:~> echo 1. 2. 0. | polyxx
Please enter polynomial coefficients (c0, c1, c2):
f(x) = 0 * x^2 + 2 * x^1 + 1
ERROR: effective degree of polynomial is less than 2!
       c(2) = 0
The coefficient of order two is too small: 0
```

# Wozu sind Kommentare gut?

Beispiel für unlesbaren Quelltext: Ein anderes *hello world* Programm.

```
thof@lizzy:~> gcc -o helloworld helloworld.c
thof@lizzy:~> helloworld 48.3 8.3 | fold
              !!!!!!!!!!! !!!            !!!   !!!!!!!
! !!!!!!!!!!!!!!!!  !!!!! !    !      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 !!!!!!!!!!!!!!!!!! !!!!          !  !! !!!!!!!!!!!!!!!!!!!!!!!!!!!!
          !!!!!!!!!!!!!            !"!!!!! !!!!!!!!!!!!!!!! !!    !
             !!!!!!!!!            !! !   !!!!!!!!!!!!!!!!!!!  !
     !        !!!!  !             !!!!!!!!!!!!!!!!!!!!!!!!!
                !!!!!            !!!!!!!!!!!!!   !!!   !!! !
                     !!!!!         !!!!!!!!!!      !     ! !  !
                     !!!!!!!!          !!!!!                    !!
                      !!!!!!            !!!! !             !!!!!
                       !!!!              !!                !!!!!!!!
                       !!                                   !!  !!     !
                       !
```

Als Parameter werden die geographische Breite und Länge übergeben.

# Wozu sind Kommentare gut?

`helloworld.c` (lines 29 to 44):

```
[...]

#include <stdlib.h>
#include <stdio.h>

           main(l
      ,a,n,d)char**a;{
  for(d=atoi(a[1])/10*80-
 atoi(a[2])/5-596;n="@NKA\
CLCCGZAAQBEAADAFaISADJABBA^\
SNLGAQABDAXIMBAACTBATAHDBAN\
ZcEMMCCCCAAhEIJFAEAAABAfHJE\
TBdFLDAANEfDNBPHdBcBBBEA_AL\
 H E L L O,    W O R L D! "
   [l++-3];)for(;n-->64;)
      putchar(!d+++33^
           l&1);}
[...]
```

Andere Beispiele: `http://www.ioccc.org/`

# Wozu sind Kommentare gut?

## Quellcode `hello_world_comm.c`:

```
/* this is hello_world_comm.c
 * ------------------------------------------------------------------------
 * $Id$
 *
 * Hello world with comments (example)
 *
 * Copyright (c) 2006 by Thomas Forbriger (BFO Schiltach)
 *
 * REVISIONS and CHANGES
 *  - 23/05/2006   V1.0   Thomas Forbriger
 *
 * ========================================================================
 */

/* read prototype for printf */
#include <stdio.h>

/* main program
 * ------------ */
int main()
{
  /* output greetings to the terminal */
  printf("Hello World!\n");
  /* indicate success */
  return(0);
} /* end of main() */

/* ----- END OF hello_world_comm.c ----- */
```

# Wozu sind Kommentare gut?

Kommentare sollen

- anderen helfen zu verstehen, was das Programm macht
- einem selber helfen den Algorithmus nachzuvollziehen (nach einem Jahr Pause nicht selbstverständlich)
- den Sinn der Befehle zusammenfassen, nicht einfach die Befehle in anderen Worten wiederholen

Im richtigen Umfang zu kommentieren ist eine Kunst.

- Zu viele Kommentare: Der eigentliche Code wird verschleiert und unleserlich.
- Zu wenig Kommentare: Der Sinn komplexer Algorithmen ist nur nach aufwändiger Code-Analyse zu rekonstruieren.
- Im Zweifelsfall: Lieber einen Kommentar mehr.

# Rundungsfehler

Aus *Numerical Recipes* (Kapitel 5.6):

Die Lösung von

$$ax^2 + bx + c = 0 \tag{1}$$

erhält man mit

$$x_{1,2} = \frac{-b \mp \sqrt{b^2 - 4ac}}{2a}, \tag{2}$$

$$x_{1,2} = \frac{2c}{-b \pm \sqrt{b^2 - 4ac}} \tag{3}$$

oder

$$x_1 = \frac{q}{a} \quad \text{und} \quad x_2 = \frac{c}{q} \quad \text{mit} \tag{4}$$

$$q = -\frac{1}{2}\left(b + \text{sign}(b)\sqrt{b^2 - 4ac}\right). \tag{5}$$

Mathematisch sind diese Lösungen identisch, numerisch nicht.

# Rundungsfehler

## Quellcode `roundoff.f`:

```
c this is <roundoff.f>
c ----------------------------------------------------
c $Id$
c three ways to find the zeroes of a polynomial
c ====================================================
c
      program roundoff
      real a, b, c
      complex q, x1, x2, fac
      parameter(fac=(4.,0.))
      read(5, *) c, b, a
      print 50, a, b, c
      x1 = (-b-sqrt(b**2-fac*a*c))/(2.*a)
      x2 = (-b+sqrt(b**2-fac*a*c))/(2.*a)
      print 51, 1, x1, x2
      x1 = 2.*c/(-b+sqrt(b**2-fac*a*c))
      x2 = 2.*c/(-b-sqrt(b**2-fac*a*c))
      print 51, 2, x1, x2
      q=-0.5*(b+sign(1.,b)*sqrt(b**2-fac*a*c))
      x1=q/a
      x2=c/q
      print 51, 3, x1, x2
      stop
   50 format('The roots of ',f8.4,' *x**2 + ',f8.4,' *x + ',
     &  f8.4,' = 0 are:')
   51 format('Method #',i1,': x1=(',2e11.3,') x2=(',2e11.3,')')
      end
c
c ----- END OF roundoff.f -----
```

# Rundungsfehler

```
thof@lizzy:~> gfortran -Wall -o roundoff roundoff.f
thof@lizzy:~> echo 1. 2. 3. | roundoff
The roots of   3.0000 *x**2 +   2.0000 *x +   1.0000 = 0 are:
Method #1: x1=( -0.333E+00 -0.471E+00) x2=( -0.333E+00  0.471E+00)
Method #2: x1=( -0.333E+00 -0.471E+00) x2=( -0.333E+00  0.471E+00)
Method #3: x1=( -0.333E+00 -0.471E+00) x2=( -0.333E+00  0.471E+00)
thof@lizzy:~> echo 1.e-2 20. 1.e-2 | roundoff
The roots of   0.0100 *x**2 +  20.0000 *x +   0.0100 = 0 are:
Method #1: x1=( -0.200E+04  0.000E+00) x2=( -0.477E-03  0.000E+00)
Method #2: x1=( -0.210E+04 -0.000E+00) x2=( -0.500E-03 -0.000E+00)
Method #3: x1=( -0.200E+04  0.000E+00) x2=( -0.500E-03 -0.000E+00)
```

# Rundungsfehler

## Achtung Gefahr!

Kritisch sind kleine Differenzen großer Größen, wie in

$$-b + \sqrt{b^2 - 4ac} \tag{1}$$

für $b > 0$ und $b^2 \gg |4ac|$ oder möglicherweise in

$$\det(\mathbf{A}) = a_{11}a_{22} - a_{12}a_{21} \tag{2}$$

für $|\det(\mathbf{A})| \ll |a_{11}a_{22}|$ oder bei der Division komplexer Zahlen

$$\frac{a+ib}{c+id} = \frac{ac+bd}{c^2+d^2} + i\frac{bc-ad}{c^2+d^2}. \tag{3}$$

# Den Code optimieren

Die Zeit, die ein Computer benötigt, um ein Programm auszuführen hängt stark vom gewählten Programmablauf ab. Die Zeit kann verkürzt werden indem

- mehrfach benötigte Zwischenergebnisse vorab berechnet und gespeichert werden,
- Arrayelemente in einer Reihenfolge angesprochen werden, die von der Hardware besonders effizient ausgeführt werden kann,
- mehrere Schleifen über ein Array zu einer zusammengefasst werden,
- ...

Statt sich selber darum zu kümmern, sollte man es dem Compiler überlassen, den Algorithmus zu optimieren. Dazu bieten alle Compiler die Option `-O`. Mit einem Parameter können mehrere Stufen der Optimierung gewählt werden.

# 64-Bit Prozessoren

Die neueste Generation von PC-Prozessoren verwenden 64 Bit große Register. Das hat auch Auswirkungen auf die Software:

- Software, die davon profitieren will, muss entsprechend compiliert werden (siehe `gcc`-Optionen `-m32` und `-m64`).
- Für 32-Bit Prozessoren compilierte Programme laufen auch auf 64-Bit Prozessoren, falls alle verwendeten, dynamischen Bibliotheken auch als 32-Bit Variante vorliegen.
- Die tatsächliche Größe von Datentypen kann sich ändern. `long int` ist 32 Bit groß auf 32 Bit Plattformen und 64 Bit groß auf 64 Bit Plattformen. Das gleiche gilt für Pointer. Die anderen gängigen Typen behalten ihre Größe. Im Zweifelsfall kann dies mit dem `sizeof` Operator geprüft werden. Relevant ist das bei binärer Daten Ein- und Ausgabe.

# Fortran nach C Konverter (`f2c`)

Mit dem Programm `f2c` kann ein Fortran Programm in C Quellcode umgewandelt werden.

Um diesen dann zu übersetzen und zu linken werden die Header-Datei `f2c.h` und die Bibliotheken `libf2c.a` und `libm.a` benötigt.

Dieses Programm ist insbesondere hilfreich, um Fortran-Code in C Programme zu integrieren oder ihn von dort aufzurufen. Es gibt keine allgemein gültig definierte Schnittstelle zwischen Fortran und C Binärcode. Der `gfortran` Compiler bietet einige spezielle Optionen, die dafür sorgen, dass der ausgegebene Code kompatibel zum von `f2c` generierten Code ist.

# Fortran nach C Konverter (`f2c`)

## Ein Beispiel

```
thof@lizzy:~> f2c polynomf.f
polynomf.f:
   printpolynomial:
   root:
thof@lizzy:~> f2c polyf.f
polyf.f:
   MAIN polyf:
thof@lizzy:~> gcc -c -o polyf2c.o polyf.c
thof@lizzy:~> gcc -c -o polynomf2c.o polynomf.c
thof@lizzy:~> gcc -o polyf2c -lf2c -lm polyf2c.o polynomf2c.o
thof@lizzy:~> echo 1. 2. 3. | polyf2c
Binary file (standard input) matches
```

# Wichtige Bibliotheken

| | |
|---|---|
| `libm.a` | mathematische Funktionen für C Code. |
| STL | Standard Template Library für C++ Code |
| `libstdc++.a` | Standard Library für C++, enthält unter Linux die STL |
| LAPACK | Funktionen zur Lösung linearer Gleichungssysteme, etc. |
| GSL | GNU Scientific Library mit zahlreichen Modulen für die wissenschaftliche Programmierung |
| IMSL | kommerzielle numerische Bibliothek, die inzwischen durch LAPACK (und andere) ersetzt werden kann |
| PGPLOT | graphische Plotfunktionen für Fortran mit vielen verschiedenen Ausgabeformaten |

# Informationsquellen

## man pages

Zu allen wichtigen Funktionen der C Standard-Bibliothek gibt es sogenannte man-pages. Ein Beispiel:

```
thof@lizzy:~> man 3 printf 2>&1 | head -17
PRINTF(3)                  Linux Programmer's Manual                 PRINTF(3)



NAME
       printf,  fprintf,  sprintf,  snprintf,  vprintf,  vfprintf,  vsprintf,
       vsnprintf - formatted output conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);

       #include <stdarg.h>
```

# Informationsquellen

## weitere

Auf SuSE Linux Systemen liegt Dokumenation zu installierter Software unter `/usr/share/doc/packages`

Unter Linux wird die umfangreichere Dokumenation nicht mehr als man pages, sondern als *info* Dokumente bereitgestellt. Diese können mit den Kommandos `info`, `pinfo` oder `tkinfo` gelesen werden. Die Dokumentation des `gcc`-Compilers ruft man mit

```
info gcc
```

auf.

Auf der Homepage zur Vorlesung werden weitere Hinweise auf Internet-Resourcen gegeben.

# Übungsaufgabe

## Aufgabestellung

Schreiben Sie ein Programm, das eine 2x2 Matrix einliest und deren Eigenwerte berechnet. Verwenden Sie dazu die Programme zur Bearbeitung von Polynomen, die in der Vorlesung vorgestellt wurden. Wählen Sie dazu eine der Programmiersprachen C, C++ oder Fortran 77.

# Übungsaufgabe

## Zur Erinnerung

Gegeben sei eine Matrix

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}. \tag{4}$$

Die Eigenwertgleichung lautet

$$\mathbf{A}\vec{x} = \lambda\vec{x}. \tag{5}$$

Nicht-triviale Lösungen (also Lösungen für $\vec{x} \neq \vec{0}$) existieren für

$$\det(\mathbf{A} - \mathbf{E}\lambda) = 0, \tag{6}$$

wobei $\mathbf{E}$ die Einheitsmatrix ist.

# Übungsaufgabe

## Zur Erinnerung

Die Determinante ist eine Funktion des Eigenwertes $\lambda$ und kann mit

$$f(\lambda) = \det(\mathbf{A} - \mathbf{E}\lambda) \tag{4}$$
$$= (a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21} \tag{5}$$
$$= \lambda^2 - \lambda(a_{11} + a_{22}) + a_{11}a_{22} - a_{21}a_{12} \tag{6}$$

als Polynom zweiten Grades geschrieben werden.

# Danksagung

Diese Präsentation wurde mit der LaTeX Beamer Klasse von Till Tantau erstellt:

`http://latex-beamer.sourceforge.net`