Rheinisch-Westfälische Technische Hochschule
Institut für Geometrie und Praktische Mathematik

Mathematisches Praktikum (MaPra) — Sommersemester 2011

Prof. Dr. Wolfgang Dahmen — M.Sc. Mathieu Bachmann, Dipl.-Math. Jens Berger, M.Sc. Liang Zhang

Kurzeinführung[1]

1 Eine Sitzung am Rechner

Die Rechner im CIP–Pool (Raum 242 im Hauptgebäude) sind mit dem Betriebssystem Linux ausgestattet.

1.1 Das Einloggen (login)

1. Monitor einschalten, der Computer sollte durchgehend eingeschaltet sein (damit das Anmelden über das Netz möglich ist).
2. Geben Sie als Benutzernamen Ihre Matrikelnummer ein und danach Ihr Passwort.
3. War das Passwort korrekt, so wird die grafische Benutzeroberfläche von Linux gestartet. Und schon kann es losgehen.

1.2 Das Ausloggen (logout)

Wichtig: Wenn Sie Ihre Sitzung beendet haben, dürfen Sie nicht das Betriebssystem herunterfahren oder gar den Rechner abschalten, da ansonsten die Arbeit anderer Benutzer auf Ihrem Rechner empfindlich gestört wird!

1. Beenden Sie die grafische Benutzeroberfläche, indem Sie im Menü (normalerweise am unteren Bildschirmrand) entweder den Menüpunkt *Abmelden* auswählen oder den entsprechenden Knopf drücken. Wenn das Login-Fenster wieder erscheint, sind Sie ausgeloggt.
2. Monitor ausschalten.

1.3 Sicherheit und Passworte

Bitte ändern Sie aus Sicherheitsgründen nach dem ersten Anmelden als erstes ihr Passwort! Wie Sie Ihr Passwort ändern können, erklären Ihnen die Betreuer in der ersten Rechnerübungsstunde.

Bitte beachten Sie folgende Regeln zur Sicherheit von Passworten:

- Das Paßwort muß aus mindestens 6 (empfohlen: 8 und mehr) Zeichen bestehen, wobei Zeichen aus allen der Bereiche
  - Kleinbuchstaben
  - Großbuchstaben
  - Ziffern 0 bis 9
  - Sonderzeichen (%,!,?,...)

  verwendet werden sollen.

[1] Stand: 30. März 2011

- Das Paßwort sollte *kein* Wort enthalten, welches in einem Lexikon vorkommen könnte und darf insbesondere *nicht* den Namen, Geburtstag, Loginnamen oder Autokennzeichen des Anwenders oder einer seiner Angehörigen ganz oder in Teilen enthalten.
- Die Paßwörter, die Sie verwenden, wenn Sie mehrere Rechensysteme benutzen, sollten *verschieden* sein. So sollten Sie zum Beispiel *nicht* ihre TIM-Kennung (Login und Paßwort für CAMPUS Office) verwenden.

Weitere Informationen zur Sicherheit von Passworten und wie man einfach merkbare und trotzdem sichere Paßworte erstellen kann, finden Sie zum Beispiel in [6, 10].

1.4 Aufgabenblätter und Dateien

Die Aufgaben uns Zusatzinformationen zum Mathematischen Praktikum liegen auch online vor und sind im Internet unter

`http://www.igpm.rwth-aachen.de/MaPra`

zu finden. Auf den Rechnern im CIP-Pool können außerdem auch die Webbrowser `firefox` und `konqueror` benutzt werden. Mit Hilfe von

`kpdf ma1.pdf` oder `evince ma1.pdf` oder `acroread ma1.pdf`

(bzw. `ma2` etc. für die folgenden Aufgaben) können die Aufgabenblätter angesehen werden.

2 Rechnernutzung von zu Hause per Internet

Von zu Hause aus sind die Rechner

`mars.mathepool.rwth-aachen.de`
und
`venus.mathepool.rwth-aachen.de`

über das Internet zu erreichen. Sie stellen jeweils das Verbindungsglied zwischen den Arbeitsrechnern im CIP-Pool und der Außenwelt dar.
Die Namen der Rechner, auf die man sich von dort aus (mit den gleichen Zugangsdaten) einloggen kann, lauten:
foster, fuller, gaudi, gehry, gray, gropius, hadid, himmelblau, hollein, johnson, lecorbusier, libera, maki, mansart, moore, niemeyer, otto, pei, piano, rogers, rossi, stirling, tange, venturi und wright

Sollte es Schwierigkeiten bei der Erreichbarkeit der Rechner geben, wenden Sie sich bitte per Email an das Team unserer Administratoren unter der Adresse `admin@igpm.rwth-aachen.de`. Die Administratoren werden versuchen, sich sobald wie möglich um eine Lösung des Problems zu bemühen.

Für den Zugriff auf die Rechner gibt es zwei Möglichkeiten:

2.1 Zugang per NX

Auf dem Rechner läuft ein NX-Server. Mit dieser Software kann ein grafisches Benutzerinterface über das Internet, durch Datenkompression und Caching sogar über schmalbandige Leitungen übertragen werden. Um diesen Dienst nutzen zu können, benötigen Sie einen NX-Client. Dieser steht für alle gängigen Betriebssysteme auf der Webseite `http://www.nomachine.com/` kostenlos zur Verfügung. Eine Dokumentation zur Konfiguration des NX-Clienten findet man unter `http://www.nomachine.com/documents/configuration/pdf/client-guide.pdf`.

Wichtig: Bitte stellen Sie sicher, dass die Einstellung „Enable SSL encryption of all traffic“ in Ihrem NX-Client aktiviert ist.

2.2 Zugang per ssh und scp

Auf dem Rechner läuft außerdem ein ssh-Server. Über diesen kann mittels `ssh` eine Shell übertragen werden und mit `scp` können Dateien vom oder auf den Server kopiert werden. Die Software ist in den meisten Linux-Distributionen vorinstalliert. Ein geeigneter Client für Windows liegt zum Beispiel auf dem `ftp`-Server des Deutschen Forschungsnetzes (DFN) unter der Adresse `ftp://ftp.cert.dfn.de/pub/tools/net/ssh/` bereit.

Um sich mittels `ssh` auf einem der zur Verfügung stehenden Rechner einzuloggen reicht es folgenden Befehl in der Konsole einzugeben:

```
ssh -XY Benutzername@Host
```

Danach kommt nur noch die obligatorische Passwortabfrage und man ist auf „Host“ eingeloggt. Konkret heißt das, falls man sich von außen einloggen will:

1. Verbinden zu mars oder venus, via

   ```
   ssh -XY Benutzername@mars.mathepool.rwth-aachen.de
   ```
   bzw.
   ```
   ssh -XY Benutzername@venus.mathepool.rwth-aachen.de
   ```

2. Weiterverbinden zu einem der 25 verfügbaren Rechner, z.B. via

   ```
   ssh -XY Benutzername@foster
   ```

Informationen zu anderen Möglichkeiten, die `ssh` bietet, bekommt man indem man in der Konsole das Kommando `man ssh` (man für manual=Anleitung) eingibt.

3 Übungen zu Hause bearbeiten

Viele der Übungen benutzen keine betriebssystemspezifischen Besonderheiten. Daher ist es im Prinzip möglich, einige Vorarbeit am Rechner zu Hause zu leisten. Häufig gibt es jedoch Testroutinen, die die von Ihnen ermittelten Ergebnisse testen. Diese sind vorkompiliert und laufen daher nur unter dem jeweiligen Betriebssystem.

- Wer unter Linux (Kernel 2.4.4 und höher) arbeitet, kann zu Hause arbeiten und braucht sich nur die Übungen und Ankreuzaufgaben im Rechnerraum testieren zu lassen.

- Wer unter einer anderen Oberfläche arbeiten möchte, kann sich zum Beispiel einen der folgenden Compiler aus dem Internet besorgen: den Intel C++-Compiler[2] (ICC) den Borland-Compiler[3] (C++-Builder von CodeGear) oder das Microsoft-Studio[4]. Eventuell besteht auch die Möglichkeit, eine Lizenz eines dieser Produkte über das Rechenzentrum der RWTH zu erhalten. Bitte wenden Sie sich dazu an das Rechenzentrum[5].

  Mit diesen Programmen kann der Code aber nur *im Prinzip* getestet werden, da die Testroutinen nicht aufgerufen werden können. Für die reine C++-Programmierung sind diese Tools natürlich geeignet und ausreichend.

Wichtig: Die Abgabe der Aufgabe sollte jeweils möglichst nicht erst am letzten Termin erfolgen, da es sonst erfahrungsgemäß zu großem Andrang bei der Testatabnahme kommt und keine Nachbesserungen mehr möglich sind.

4 Arbeiten mit LINUX

Eines der Fenster auf dem Bildschirm, es ist mit `konsole` oder `xterm` überschrieben, stellt Ihnen eine sogenannte Shell zur Verfügung. Wenn Sie den Shell-Knopf (Symbol: Bildschirm mit Muschel oder Kleiner-Zeichen) in der Menüleiste betätigen, wird eine neue Shell gestartet. Hier können Sie Ihre Befehle eingeben. Einige wichtige sind im Folgenden kurz zusammengefasst.

4.1 Shell-Befehle

`mkdir` *Name*	Erstelle ein Verzeichnis *Name* (make directory).
`rmdir` *Name*	Entferne das (leere) Verzeichnis *Name* (remove directory).
`ls`	Zeige die Dateien im aktuellen Verzeichnis an (list).
`cd` *Name*	Wechsle in das Verzeichnis *Name* (change directory).
`cd ..`	Wechsle in das übergeordnete Verzeichnis.
`cd`	Wechsle ins Home–Verzeichnis.
`cp` *Quelle Ziel*	Kopiere die Datei *Quelle* nach *Ziel* (copy).
`rm` *Name*	Lösche die Datei *Name* (remove). **Vorsicht: Weg ist weg!**
`mv` *Name Verzeichnis*	Verschiebe die Datei *Name* ins Verzeichnis *Verzeichnis* (move).
`mv` *Name1 Name2*	Benenne die Datei *Name1* in *Name2* um.
`mcopy` *Name* `a:`	Kopiere die Datei *Name* auf Diskette.
`mcopy a:`*Name* `.`	Kopiere die Datei *Name* von Diskette ins aktuelle Verzeichnis.
`nedit` *Name*	Editiere (bearbeite) die Datei *Name*.
`man` *Befehlsname*	Zeige Hilfe zum Befehl *Befehlsname* an (manual).
`g++ -o` *Prog Name1 Name2 etc.*	Übersetze die C++–Dateien *Name1 Name2 etc.* und füge sie zum Programm *Prog* zusammen.
`kdbg | gvd | ddd | gdb` *Name*	Debugge das Programm *Name*.

4.2 Optionen

Die meisten der Befehle haben eine ganze Reihe zusätzlicher Optionen, mit denen man ihr Verhalten verändern kann. Die Optionen werden fast immer zwischen dem Befehl und dem nachfolgenden Dateinamen eingefügt. Um sie von Dateinamen zu unterscheiden, beginnen sie mit einem Minuszeichen. Beispielsweise zeigt `ls -l` die Dateien mit zusätzlichen Informationen (Größe, Datum etc.) an. Mit `ls -t` kann man sich die Dateien nach dem Datum sortieren lassen. Optionen, die nur aus einem Buchstaben bestehen, kann man oft

[2] Webseite der Firma Intel: `http://www.intel.com`
[3] Webseite der Firma Borland: `http://www.borland.com`
[4] Webseite der Firma Microsoft: `http://www.microsoft.com`
[5] Webseite des Rechenzentrums der RWTH Aachen: `http://www.rz.rwth-aachen.de`

zusammenfassen: `ls -lt` oder `ls -tl`. Wenn nach einer Option noch eine weitere Angabe (wie zum Beispiel `-o` *Prog* bei `g++`) folgt, dann funktioniert das allerdings nicht.

4.3 Befehle in den Hintergrund schieben

Normalerweise nimmt die Shell erst dann wieder Befehle entgegen, wenn der letzte eingegebene abgearbeitet wurde. Bei Befehlen wie `ls` ist das auch ganz vernünftig. Wenn Sie allerdings den Editor mit `nedit` *Name* gestartet haben, bleibt das Fenster so lange blockiert, bis Sie ihn wieder verlassen haben. Damit das nicht passiert (schließlich wollen Sie ein Programm nach einer Änderung übersetzen und testen, ohne jedesmal den Editor zu verlassen), können Sie stattdessen `nedit` *Name* `&` eingeben. Durch das Und-Zeichen `&` am Ende wird der Befehl in den „Hintergrund“ verschoben, und die Shell ist sofort für den nächsten Befehl bereit.

5 Der Editor `nedit`

Der Editor `nedit` stellt eine Vielzahl an Möglichkeiten zur Bearbeitung eines Textes zur Verfügung. Ein großer Teil von ihnen kann über die Menüleiste am oberen Rand des Fensters aufgerufen werden. Durch Anklicken des Menüpunkts `File` mit der linken Maustaste wird ein kleines Untermenü eröffnet, das zum Beispiel die Punkte `Save` (zum Abspeichern des gerade bearbeiteten Textes) und `Exit` (zum Verlassen von `nedit`) enthält. Viele dieser häufig benutzten Editor–Befehle kann man auch über spezielle Tastenkombinationen wesentlich schneller erreichen. Einige wichtige darunter sind:

<*Ctrl*>–*n*	Neues Editorfenster öffnen.
<*Ctrl*>–*o*	Datei öffnen.
<*Ctrl*>–*s*	Speichern der Datei.
<*Ctrl*>–*w*	Schließen eines Editorfensters.
<*Ctrl*>–*q*	Verlassen des Editors.
<*Ctrl*>–*f*	Suche in der Datei.
<*Ctrl*>–*g*	Weitersuchen.
<*Ctrl*>–*r*	Suchen und ersetzen.
<*Ctrl*>–*l*	Springe zu bestimmter Zeile.
<*Ctrl*>–*c*	Markierten Text in die Zwischenablage kopieren.
<*Ctrl*>–*x*	Markierten Text ausschneiden und in die Zwischenablage schieben.
<*Ctrl*>–*v*	Text aus der Zwischenablage einfügen.
<*Ctrl*>–*z*	Änderungen schrittweise rückgängig machen. Sehr nützlich!

Textregionen können mit der linken Maustaste markiert werden. Dazu drücken Sie am Anfang des zu markierenden Textes die linke Maustaste und fahren, ohne die Taste loszulassen, bis zum Ende der Region. Dort können Sie die Taste loslassen. Um diesen Text zu löschen, drücken Sie <*Backspace*>. Wenn Sie den Text stattdessen kopieren wollen, drücken Sie mittlere Maustaste an der entsprechenden Stelle (wenn Sie die dann die <*Backspace*>–Taste drücken, wird der Text an der ursprünglichen Stelle gelöscht, Sie haben ihn dann also verschoben).

Alternativ zu `nedit` kann auch ein anderer Editor, zum Beispiel `kwrite`, `kate`, `gedit` oder `scite` verwendet werden. Fortgeschrittene können sich einmal `vi` bzw. `vim`[6] oder `xemacs`[7] ansehen.

6 Erste Schritte in C++

Betrachten wir das folgende Programm:

[6] Webseite zu `vim` (Vi IMproved): `http://www.vim.org/`. Eine Schnellübersicht zum `vim` finden Sie unter `http://tnerual.eriogerg.free.fr/vimqrc-ge.pdf`, eine Kurzanleitung unter `http://lug.fh-swf.de/vim/vim-kurzanleitung.pdf`

[7] Webseite zum XEmacs: `http://www.xemacs.org/`. Die grundlegenden und wichtigsten Tastenkombinationen des `xemacs` sind zum Beispiel unter `http://www.cs.dal.ca/studentservices/refcards/xemacs.pdf` zu finden.

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main ()
6 {
7   double d;
8   d = 2.0;
9   cout << sqrt(d) << endl;     // Zahl ausgeben
10  cout << "sqrt(d)" << endl;  // Text ausgeben
11  return 0;
12 }
```

Die ersten beiden Zeilen binden die beiden Header–Dateien `iostream` (für `std::cout` und `std::endl`) und `cmath` (für `sqrt`) ein. Die dritte Zeile bewirkt, dass man in den Namensraum `std` wechselt. Dies hat zur Folge, dass die Befehle `std::cout` und `std::endl` durch `cout` und `endl` aufrufbar werden. Der Text hinter `//` (bis zum Zeilenende) ist ein Kommentar für den Benutzer und hat keinerlei Auswirkungen auf den Programmablauf.

Durch `double d` wird eine Variable `d` vom Typ `double` angelegt, d.h. es wird Speicherplatz reserviert. Damit kann `d` eine Fließkommazahl doppelter Genauigkeit (meist etwa 16 Dezimalstellen) aufnehmen. Das Semikolon `;` signalisiert das Ende eines Befehls und darf nicht vergessen werden. Durch die Zuweisung `d = 2.0` erhält `d` den Wert 2.0.

In der nächsten Programmzeile passiert eine ganze Menge: Mit `cout` werden Daten ausgegeben. Als erstes `sqrt(d)` (die Wurzel von `d`), dazu wird die Funktion `sqrt` aus `cmath` mit dem Argument `d` aufgerufen. Durch `endl` wird die aktuelle Zeile beendet und der Text auf den Bildschirm geschrieben.[8] In der nächsten Zeile steht `sqrt(d)` aber in Anführungszeichen, so dass es als Zeichenkette (string), also nur als eine Folge von Buchstaben, interpretiert wird. Daher wird in dieser Zeile der Text `sqrt(d)` ausgegeben.

Die vier zuletzt unter die Lupe genommenen Programmzeilen werden durch die geschweiften Klammern zu einer Einheit zusammengefasst. Sie bilden gemeinsam den Rumpf der Funktion `main`. Jedes C++-Programm benötigt eine Funktion dieses Namens; sie ist das Hauptprogramm und wird als erstes gestartet. Sie kann weitere Funktionen aufrufen, in unserem Fall beispielsweise die Funktion `sqrt`. Neben dem Rumpf, der festlegt, was eine Funktion macht, braucht sie auch einen Kopf, der festlegt, wie sie mit der Außenwelt in Verbindung tritt. Dieser Kopf ist hier `int main ()`. Er besagt, dass der Funktion keine Daten übergeben werden (denn zwischen den Klammern steht nichts) und dass sie einen ganzzahligen Wert (`int`) zurückgibt (erfolgt in diesem Falle durch `return 0;`). Wir könnten alternativ auch

```
int main (void)
```

schreiben. Die Funktion `sqrt` ist in `cmath` dagegen als

```
double sqrt (double);
```

deklariert: ihr wird eine `double`–Zahl übergeben, und sie gibt eine solche zurück.

7 Übersetzen und Linken

Wenn Sie das Programm aus dem letzten Abschnitt mit Hilfe des Editors eingegeben und unter dem Namen `wurzel.cpp` abgespeichert haben, ist das zunächst nichts weiter als eine Textdatei, mit der der Computer noch nicht viel anfangen kann. Es muss erst noch übersetzt (kompiliert) werden, um in eine für den Computer lesbare Form (Maschinencode) umgewandelt zu werden; dazu dient der Compiler. Wir benutzen hier den

[8] Normalerweise werden Ausgaben erst in einem Puffer zwischengespeichert und (wenn er voll ist) in einem Rutsch geschrieben, was meist effizienter ist als viele einzelne Ausgaben.

C++-Compiler `g++` aus der GNU Compiler Collection[9], die den Bedingungen der GNU General Public License[10] unterliegt, auf vielen Plattformen verfügbar und auch im CIP-Pool installiert ist. Bei anderen Compilern funktioniert es meist ähnlich, jedoch können die Optionen abweichend sein.

Der Compiler erzeugt aus `wurzel.cpp` zunächst den sogenannten Objektcode. Er wird in der Datei `wurzel.o` abgespeichert, wenn man den Compiler in der Shell mit

```
g++ -c wurzel.cpp
```

aufruft. In einem zweiten Schritt wird dieser Code zu einem ausführbaren Programm zusammengefügt. Dieser Vorgang wird auch als Linken bezeichnet und mit dem Aufruf

```
g++ -o wurzel wurzel.o -lm
```

ausgeführt. So wird das Programm `wurzel` erzeugt, das Sie nun mit `./wurzel` oder vielleicht auch `wurzel` starten können (die erste Version funktioniert immer, die zweite nur dann, wenn der sogenannte Suchpfad auch das aktuelle Inhaltsverzeichnis „.“ enthält).

Wegen der Option `-o wurzel` bekommt das Programm den Namen `wurzel`. Beim Linken wird auch der Code für die Ausgabe und zur Berechnung der Wurzel hinzugefügt. Dieser findet sich in sogenannten Bibliotheken (libraries), in denen der entsprechende Objektcode abgelegt ist. Die mathematischen Routinen befinden sich in der Bibliothek `libm`, die durch die Option `-lm` hinzugefügt wird.[11] Weil diese Option sich erst auf den zweiten Schritt der Programmbearbeitung bezieht, steht sie (im Gegensatz zur üblichen Konvention) am Ende der Zeile.

Beide Schritte, Übersetzen und Linken, können Sie auch mit einem Aufruf

```
g++ -o wurzel wurzel.cpp -lm
```

hintereinander durchführen lassen.

Statt einer einzigen Quelldatei mit dem C++–Programm kann man dem Compiler auch mehrere Dateien übergeben, die dann zusammengelinkt werden. Darunter können auch Dateien mit Objektcode sein, bei denen dann der erste Bearbeitungsschritt übersprungen wird. Durch

```
g++ -o prog datei1.o datei2.o -lIGL
```

werden beispielsweise die Objektdateien `datei1.o` und `datei2.o` zusammen mit der IGPM-Grafikbibliothek `libIGL` zu einem ausführbaren Programm `prog` gelinkt.

Dem Compiler können noch weitere Optionen übergeben werden. Die Option `-g` erleichtert die Fehlersuche, indem sie einem Debugger (Fehlersuchprogramm) zusätzliche Informationen mitgibt, die allerdings das Programm aufblähen. Die Optionen `-O1` bis `-O3` optimieren das Programm. Das Übersetzen dauert dann länger, aber der produzierte Code wird (meist) schneller. Die Optionen `-Wall` und `-pedantic` geben zusätzliche Warnungen oder Fehlermeldungen über möglicherweise *gewagte* Programmiervorgänge aus – manchmal eine gute Hilfe.

Mit der Option `-m32` fordern Sie auf 64-Bit-Betriebssystemen (wie im CIP-Pool installiert) den Compiler auf, Maschinencode für 32-Bit-Prozessoren zu generieren. Auf 64-Bit-Betriebssystemen generiert der Compiler automatisch Maschinencode für 64-Bit-Prozessoren, wenn nichts Gegenteiliges angegeben wird. Dieser ist jedoch inkompatibel zu Maschinencode für 32-Bit-Prozessoren, den Sie vielleicht zu Hause benötigen. Daher stellen wir die notwendigen Objektdateien stets in einer Version für 32-Bit-Betriebssysteme und auch in einer Version für 64-Bit-Betriebssysteme zur Verfügung.

Das Buch von Gough [18] führt verständlich in die Welt der Optionen der C- und C++-Compiler der GNU Compiler Collection ein. Die Liste aller Optionen des Compilers `g++` findet man auf den Info-Seiten, die man

[9] Webseite der GNU Compiler Collection: http://gcc.gnu.org/
[10] Die GNU General Public License ist eine Lizenz für freie Software.
[11] Bei neueren Compilern kann man auf `-lm` verzichten, da diese Bibliothek automatisch hinzugefügt wird.

mit `info g++` aufrufen kann oder auch in der Online-Dokumentation auf der Webseite der GNU Compiler Collection.

Noch ein Hinweis: Nicht entmutigen lassen, wenn der Compiler beim Übersetzen Ihres Programmcodes Dutzende von Fehlermeldungen ausgibt! Oft handelt es sich dabei um Folgefehler, die verschwinden, wenn der allererste vom Compiler monierte Fehler (zum Beispiel eine vergessene schließende Klammer oder ein vergessenes Semikolon) beseitigt ist. In einem solchen Fall sollten Sie also in den Fehlermeldungen immer den *zuerst* aufgetretenen Fehler lokalisieren und beheben und anschließend Ihren Code erneut übersetzen.

8 Fehlersuche

In diesem Abschnitt wollen wir lernen, wie man mit einem Debugger arbeitet. Er ist ein wichtiges Werkzeug bei der Fehlersuche in Programmen. Wir behandeln hier zunächst ein triviales Beispiel, das aber die Vorgehensweise gut widerspiegelt. Danach stellen wir die wesentlichen Befehle des Debuggers zusammen.

8.1 Beispielsitzung zur Fehlersuche

Wir wollen eine kleine Variante des „Hello World“-Programms schreiben. Nehmen wir einmal an, dieses würde wie folgt aussehen:

```
1 #include<iostream>
2 using namespace std;
3
4 int j;
5
6 int main()
7 {
8   // This is the world's most famous program!
9   cout << "Hello" << endl;
10  cout << 10/j << endl;
11  cout << "GoodBye" << endl;
12 }; // semicolon is not necessary!
```

Nun wollen wir dieses Programm übersetzen. Geben Sie dazu im Shell-Fenster den Befehl

```
g++ -g -o hello hello.cpp
```

ein. Die Compileroption `-g` ermöglicht es, das Programm mit dem Debugger zu behandeln. So erhalten Sie ein ausführbares Programm `hello`. Starten Sie es, indem Sie in der Shell `hello` aufrufen. Sie erhalten folgende Fehlermeldung:

```
Hello
Floating point exception (core dumped)
```

Wir wollen nun mit Hilfe des Debuggers dem Fehler auf die Spur kommen. Rufen Sie dazu mit dem Befehl

```
kdbg hello
```

den Debugger auf. Sie erhalten ein Haupt- und ein Programmausgabefenster.

Im Hauptfenster erscheint nun der Quellcode des obigen Programms. Setzen Sie nun mit der rechten Maustaste einen Haltepunkt (breakpoint) in der Zeile

```
cout << "Hello" << endl;
```

und starten Sie das Programm mit `Ausführung | Ausführen`. Damit wird das Programm bis zu dem gesetzten Haltepunkt ausgeführt und dort angehalten. Ein grüner Pfeil erscheint und zeigt die aktuelle Position im Quellcodefenster an.

Jetzt benutzen Sie den Menüpunkt `Ausführung | In Funktion` oder das entsprechende Symbol, dadurch springt der Debugger zur nächsten Programmzeile und gibt den Text `Hello` im Ausgabefesnter aus. Der grüne Pfeil springt in die nächste Zeile. Ein erneutes `Ausführung | In Funktion` bewirkt die Ausgabe der obigen Fehlermeldung in der Statuszeile des Debuggers `kdbg`. Das Programm ist also in der Zeile

```
cout << 10/j << endl;
```

abgestürzt.

Wählen Sie nun `Ansicht | Ausdrücke` und geben Sie links oben im Fenster `Ausdrücke` die Variable `j` ein. Ihnen wird nun der Wert für `j` angezeigt: er beträgt Null! Offensichtlich versuchen wir in dieser Zeile, durch Null zu dividieren, was natürlich nicht erlaubt ist.

Ersetzen Sie nun Zeile 4 zum Beispiel durch

```
int j=1;
```

und übersetzen und starten Sie das Programm. Nun erscheint auf dem Bildschirm die Ausgabe

```
Hello
10
Good Bye
```

Natürlich hätte dieses Problem auch anders gefunden und behoben werden können.

8.2 Der Debugger `kdbg`

Wir wollen nun die wesentlichen Befehle für den grafischen Debugger `kdbg` (KDE Debugger) erklären. Natürlich können Sie auch jeden anderen Debugger[12] benutzen. Wie bereits gesehen, wird `kdbg` mit dem Befehl `kdbg` *Programmname* aufgerufen.

Die folgende Tabelle enthält eine Zusammenstellung einiger wichtiger Befehle des `kdbg`. Es versteht sich von selbst, dass diese Liste nicht vollständig sein kann.

kdbg-Befehle	
`Haltepunkt | Setzen/entfernen`	Stoppt das Programm in der Zeile des Haltepunkts, nachdem es gestartet wurde.
`Haltepunkt | Temporären Haltepunkt setzen`	Stoppt das Programm nur beim nächsten Durchlauf an dieser Stelle.
`Ausführung | Zum Cursor`	Führt das bereits gestartete Programm bis zu dieser Programmzeile aus.
`Ansicht | Ausdrücke`	Öffnet ein Fenster, in dem die Werte der Variablen angezeigt werden können.
`Ausführung | Ausführen`	Programmstart im Kommandofenster. Ist kein Haltepunkt gesetzt, so läuft das Programm solange, bis es durch einen Fehler abstürzt oder normal beendet wird. Ansonsten wird die Programmausführung am Haltepunkt gestoppt.
Ausführen | In Funktion	Fortsetzen eines gestoppten Programms nur für einen Programmschritt. Der Debugger springt dabei auch in die nächste Funktion, falls Debugginginformationen verfügbar sind.

[12] zum Beispiel `gvd`, `ddd` oder `gdb`

Ausführen | Über Funktion	Fortsetzen eines gestoppten Programms nur für einen Programmschritt. Erfolgt im nächsten Schritt ein Funktionsaufruf, so überspringt der Debugger überspringt dabei die Einzelschritte der Funktion.
`Ausführung | Argumente`	Eingabe von Kommandozeilenargumenten für das Programm.

Da der Debugger `kdbg` eine vollständige grafische Oberfläche besitzt, sollten weitere Befehle selbsterklärend sein. Im Zweifelsfall finden Sie oben rechts in der Menüleiste eine Online-Hilfe.

9 Datentypen in C++

Elementare Datentypen	
Fließkommazahlen (floating–point numbers)	
`float`	`float x; x=3.0;` (meist 4 Bytes)
`double`	`double y; y=-1.5e4;` (meist 8 Bytes)
`long double`	`long double z; z=y;` (meist 8–16 Bytes)
ganze Zahlen (integers)	
`short int, (un)signed short int`	`short int i; i=1;` (meist 1 oder 2 Bytes)
`int, (un)signed int`	`int j; j=-3;` (meist 2 oder 4 Bytes)
`long int, (un)signed long int`	`long int k; k=123;` (meist 4 oder 8 Bytes)
Buchstaben (characters)	
`char, (un)signed char`	`char c; c='f';` (meist 1, selten 2 Bytes)
Logische Werte (booleans)	
`bool`	`bool b; b=true; b=false;` (meist wie `int`)
Aufzählung (enumeration)	

`enum`	`enum Antwort {Ja,Nein}; Antwort Ant;`
	`enum {Ja,Nein} Ant; Ant=Ja;`
	`enum Antwort {Ja=1,Nein=0};`

Nichts	
`void`	nur in zusammengesetzten Typen Zeiger: `void *Ptr`, Funktion: `void main ()`

Zusammengesetzte Datentypen	
Felder (arrays)	

Datentyp Name`[`*Konstante*`]`	`int Zahlen[6];`
	`Zahlen[0]=3; Zahlen[5]=1;`
	`double Vektor[42];`

Zeiger (pointers)	

Datentyp `*`*Name*	`int *Zeiger;`
	`Zeiger=new int[10]; Zeiger[9]=1;`
	`delete[] Zeiger;`
	`Zeiger=new int; *Zeiger=1;`
	`delete Zeiger;`

Datensätze (structures)	
`struct` *Name*	`struct Paar;`

`struct` *Name* `{`*Definition*`};`	`struct Paar { int i; double x; };`
	`Paar IntDbl;`
	`IntDbl.i=3; IntDbl.x=6.0;`

Funktionen (functions)	

Datentyp Name`(`*Datenliste*`)`	`void main ()`
	`double Maximum ( double, double )`
	`void Vertausche ( double &, double & )`

Datentyp Name`(`*Datenliste*`) {`*Definition*`}`	`void Plus1 ( int &i ) { i++; }`
Klassen (classes)	
`class` *Name*	`class Vektor;`
`class` *Name* `{`*Definition*`};`	(structs sind Spezialfälle von Klassen)

10 Operatoren in C++

Operator	Erklärung	Beispiel
Vorzeichen		
`+ -`		`+a -3`
Arithmetik		
`+ - * /`		`1+3 a*b`
`%`	Rest der Division (modulo)	`5%3`
Zuweisung		
`=`		`a=1 b=x`
`+= -= *= /= %=`	Operation und Zuweisung	`a+=1`
Inkrement, Dekrement		
`++ --` (Präfix)	vorher erhöhen/erniedrigen	`++a --b`
`++ --` (Postfix)	nachher erhöhen/erniedrigen	`a++ b--`
Vergleich		
`== !=`	gleich, ungleich	`a==1 b!=x`
`< > <= >=`		`a<1 b>=x`
Logik		
`&& || !`	und, oder, nicht	`a&&b a||b !x`
Bits		
`& | ^ ~`	und, oder, xor, Komplement	`a&b a|b a^b ~x`
`<< >>`	nach links/rechts verschieben	`a<<2 b>>x`
`&= |= ^= <<= >>=`	mit Zuweisung	`a&=b a<<=3`
Ein-/Ausgabe		
`<< >>`	Ausgabe, Eingabe	`cout<<a cin>>b`
Zeiger (Pointer)		
`new`	Speicher belegen	`new int; new int[5];`
`delete delete[]`	Speicher löschen	`delete p; delete[] p;`
Adresse, Zeiger, Member		
`&`	Adresse einer Variablen	`&a`
`*`	Dereferenz (darauf zeigt der Zeiger)	`*p`
`. .*`	Memberauswahl	`MyStruct.x MyStruct.*p`
`-> ->*`	Memberauswahl	`p->x p1->*p2`
Klammern		
`[]`	Index	`a[3]`
`()`	Funktionsaufruf	`sin(x)`
`()`	Typumwandlung	`(double)3; double(3)`
Sichtbarkeit (Scope)		
`::`	globale Variable	`::x`
`::`	Klassenmember	`MyClass::x`
Sonstiges		
`,`	Sequenz (liefert den Wert von `b`)	`a,b`
`?:`	wenn `a`, dann `b`, sonst `c`	`a?b:c`
`sizeof`	Größe	`sizeof(int); sizeof a;`
`throw`	Ausnahmebehandlung	`throw Ausnahme;`

11 Kontrollstrukturen in C++

Hintereinanderausführung
stat1 stat2 ...
`a=3; cout << a;`

Klammerung
`{` *stat1 stat2 ...* `}`
fasst mehrere Befehle zu einer Einheit zusammen

if-Anweisung
`if (`*expr*`)` *stat*
`if (i<0) cout << "Zahl ist negativ!" << endl;`

if-else-Anweisung
`if (`*expr*`)` *stat1* `else` *stat2*
`if (a<0) b=sqrt(-a); else b=sqrt(a);`

switch-Anweisung
`switch (`*expr*`) { case` *const1*`:` *stat1 ...* `case` *constn*`:` *statn* `}`
`switch (i) { case 0: a=5; break; case 1: a=8; break; case 2: a=1; }`

switch-Anweisung mit default
`switch (`*expr*`) { case` *const1*`:` *stat1 ...* `case` *constn*`:` *statn* `default:` *stat* `}`
`switch (i) { case 0: a=5; break; case 1: a=8; break; default: a=1; }`

while-Schleife
`while (`*expr*`)` *stat*
`while (i>0) { i--; j*=2; }`

do-Schleife
`do` *stat* `while (`*expr*`);`
`do { i--; j*=2; } while (i>0);`

for-Schleife
`for (`*stat_init*`;` *expr*`;` *stat_inc*`)` *stat*
`for ( i=0; i<n; i++ ) cout << x[i];`

break-Anweisung
`break;`
verlässt eine Schleife

continue-Anweisung
`continue;`
bricht einen Schleifendurchgang ab

return-Anweisung
`return` *expr*`;`
verlässt eine Unterroutine und gibt den Wert *expr* zurück

12 Literaturverzeichnis

Literatur zur Numerischen Mathematik

[1] Dahmen, W. und A. Reusken: *Numerische Mathematik für Ingenieure und Naturwissenschaftler*. Springer Verlag, Heidelberg, 2. Auflage, 2008.

[2] Deuflhard, P. und A. Hohmann: *Numerische Mathematik I*. de Gruyter, Berlin, 4. Auflage, 2008.

[3] Schwarz, H.-R. und N. Köckler: *Numerische Mathematik*. B.G. Teubner, Wiesbaden, 5. Auflage, 2004.

[4] Stoer, J.: *Numerische Mathematik I*. Springer Verlag, Heidelberg, 9. Auflage, 2004.

[5] Stoer, J. und R. Bulirsch: *Numerische Mathematik II*. Springer Verlag, Heidelberg, 5. Auflage, 2005.

Literatur zu C++

[6] *Regeln bei der Vergabe von Paßwörtern im Rechenzentrum der RWTH Aachen*. `http://www.rz.rwth-aachen.de/aw/cms/rz/Themen/unsere_dienste/kommunikation/sicherheit/warum_sicherheit_/~rhy/Regeln_bei_der_Vergabe_von_Passwoertern`.

[7] Davis, S. R.: *C++ für Dummies*. mitp, 2000.

[8] Erlenkötter, H.: *C++. Objektorientiertes Programmieren von Anfang an*. Rowohlt Taschenbuch, Reinbek, 8. Auflage, 2000. `http://www.erlenkoetter.de/html/c.htm`.

[9] Josuttis, N.: *Objektorientiertes Programmieren in C++*. Addison-Wesley, Bonn, 2001. `http://www.josuttis.com/cppbuch/`.

[10] Kruse, C.: *Die sichere Passwort-Wahl*. `http://aktuell.de.selfhtml.org/artikel/gedanken/passwort/index.htm`.

[11] Lippman, S. B. und J. Lajoie: *C++ Primer*. mitp, Bonn, 2002. `http://www.awprofessional.com/cpp_primer`.

[12] Prata, S.: *C++ Primer Plus*. SAMS Publishing, Indianapolis, IN, 4. Auflage, 2001.

[13] Stroustrup, B.: *Die C++-Programmiersprache*. Addison-Wesley, Bonn, 4. Auflage, 2000. Deutsche Übersetzung der Special Edition, `http://www.research.att.com/~bs/3rd.html`.

Online-Referenzen

[14] *C++ FAQ Lite*. `http://www.dietmar-kuehl.de/mirror/c++-faq/`.

[15] *C++-Kurs der MATSE-Ausbildung im Rechenzentrum der RWTH Aachen*. `http://www.rz.rwth-aachen.de/ca/k/qxm/lang/de/`.

[16] *C/C++-Referenz*. `http://www.cppreference.com/index.html`.

[17] *Dokumentation der C++ Standard Template Library (STL)*. `http://www.sgi.com/tech/stl/`.

[18] Gough, B. J.: *An Introduction to GCC - for the GNU Compilers gcc and g++*. A network theory manual. Network Theory Ltd., Bristol, 2005. `http://www.network-theory.co.uk/docs/gccintro/`.