Zusätzliche Unterlagen

Inhaltsverzeichnis

1 Hilfe- und Dokumentationsseiten unter UNIX (man pages)

Man–Pages (nach dem Unix-Kommando `man`, was für englisch manual "Handbuch" steht) sind eine Sammlung von Hilfe- und Dokumentationsseiten unter Unix und verwandten Betriebssystemen. In den *Man-Pages* finden sich die Dokumentationen aller in der Unix–Shell verfügbaren Befehle, aber auch die für die Lehrveranstaltung wichtige Dokumentation aller in der Programmiersprache C vordefinierten Funktionen.

Um die *Man–Pages* aufrufen zu können muss vorerst eine Konsole (Terminal) gestartet werden. Dies kann durch Aufrufen des Software– Schnellzugriffes durch Drücken der Tastenkombination `Alt + F2`, vgl. Abbildung 1, oder durch Auswahl der Konsolenanwendung über die Menüleiste bzw. durch Starten der Konsole über die Verknüpfung am Desktop erfolgen.

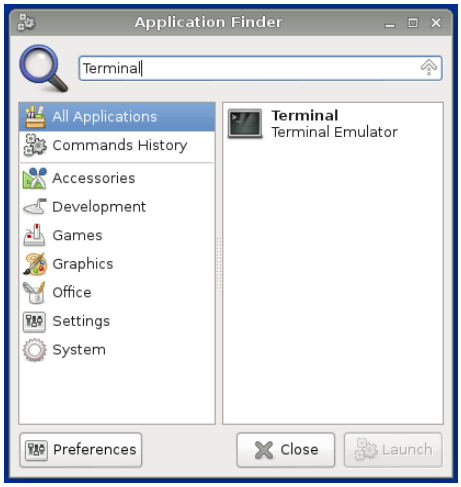

Abbildung 1: Starten der Konsole (des Terminals) mit Hilfe des *Application Finders*, welcher über die Tastenkombination `Alt + F2` aufgerufen werden kann.

Aufgerufen werden die einzelnen *Man-Pages* unter Verwendung des Schlüsselwortes `man` gefolgt vom jeweiligen Befehl, siehe Abbildung 2. Zum Beispiel liefert der Ausdruck `man strstr` die Hilfeseiten zur C-Funktion `strstr` wie in Abbildung 3 dargestellt.

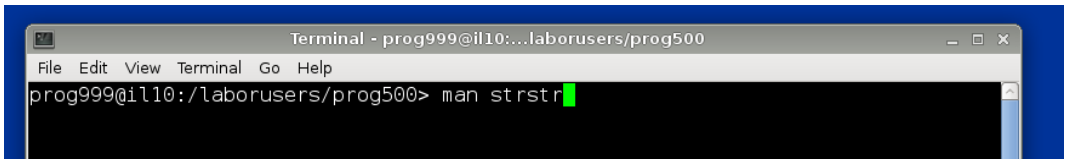

Abbildung 2: Befehl zum Aufrufen der Hilfeseite für `strstr`.

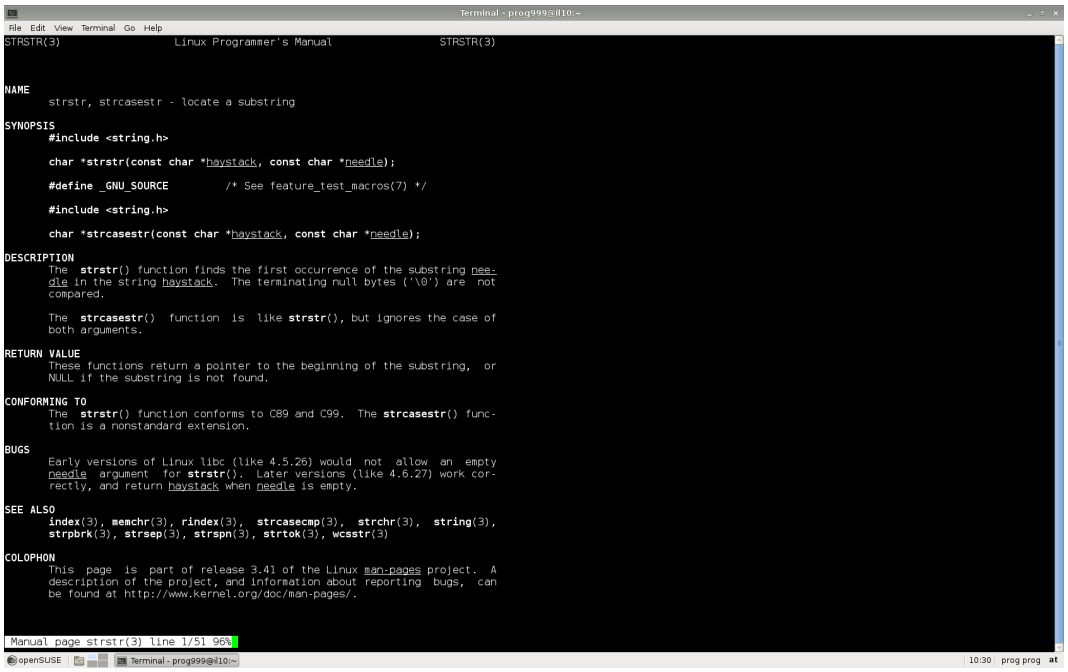

Abbildung 3: Hilfeseite zu `strstr`.

Eine *Man–Page* eines C–Befehls bzw. eines Programms ist üblicherweise in folgende Bereiche unterteilt (in Klammer gesetzte Ausdrücke sind optional):

- **NAME**: Name der Funktion/des Programms gefolgt von einer Kurzbeschreibung.
- **SYNOPSIS**: Hier wird eine vollständige Liste der Parameter und Optionen beschrieben. Ebenfalls findet man hier die notwendige Headerdatei für die betreffende Funktion.
- **DESCRIPTION**: Eine Textbeschreibung der Funktionsweise des Befehls oder der Funktion. (Üblicherweise jedoch nicht der Benutzung, siehe unten.)
- **RETURN VALUE**: Beschreibung der Rückgabewerte der Funktion.
- **CONFORMING TO**: Auflistung der C–Standards, in welcher der Befehl in der jeweiligen Headerdatei verfügbar ist. Für die Laborübung sind all jene Befehle interessant, welche im **C99** Standard verfügbar sind.
- **OPTIONS / FLAGS**: Eine üblicherweise alphabetisch sortierte Liste der Optionen und ihrer zulässigen Argumente.
- **(USAGE)**: Eine Anleitung für den Einsatz der Funktion/des Programmes.
- **EXIT STATUS**: Beschreibung möglicher Return Codes und deren Bedeutung.
- **(EXAMPLES)**: Einige Beispiele zur Benutzung des Befehls/des Programmes.

- **(FILES)**: Falls bestimmte Dateien (etwa Konfigurationsdateien) an der Funktion des Programms beteiligt sind oder durch die Ausführung desselben betroffen sind, so werden sie hier aufgeführt und es wird ihre Funktion beschrieben.
- **SEE ALSO**: Eine Liste ähnlicher oder verwandter Befehle oder Funktionen.

Die *Man–Pages* werden durch drücken der Taste "*q*" (quit) geschlossen und es erscheint die Eingabezeile in der Konsole.

2 Plotten unter UNIX (gnuplot)

Gnuplot ist ein skript- bzw. kommandozeilengesteuertes Computerprogramm zur grafischen Darstellung von Messdaten und mathematischen Funktionen. Aufgerufen wird es über die Unix–Konsole. Hiezu wird eine Konsole geöffnet und über die Eingabe `gnuplot` und anschließendes Drücken der Eingabetaste wird die *Gnuplot* Kommandozeile gestartet, vgl. Abbildung 4.

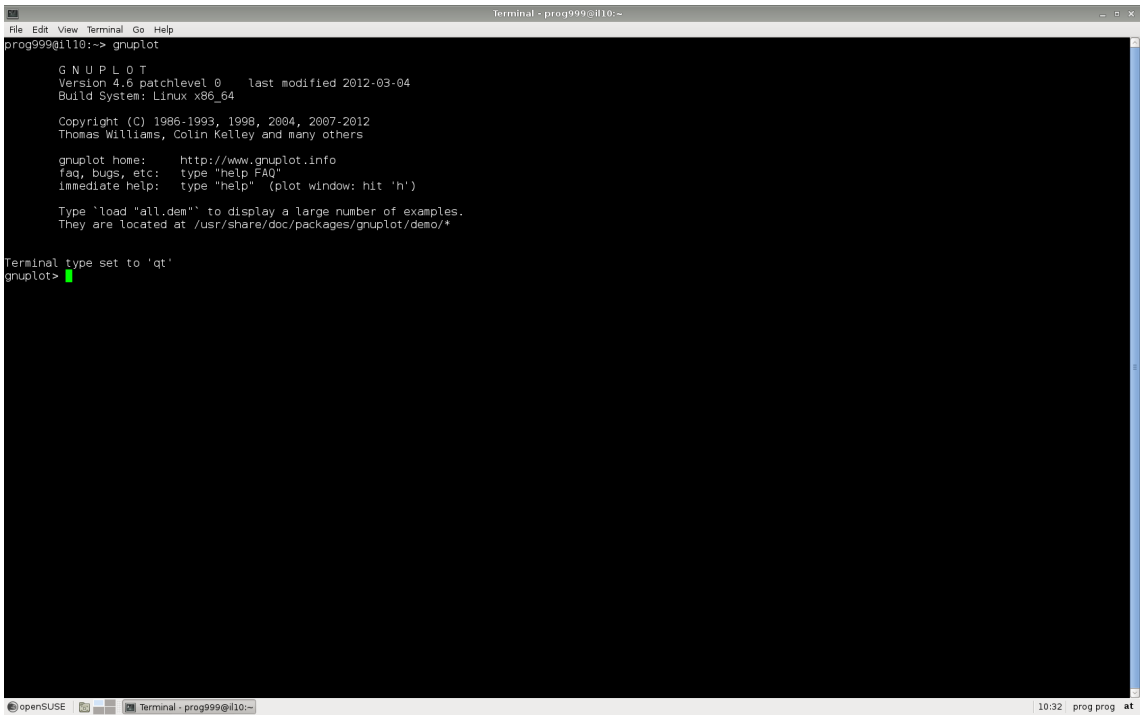

Abbildung 4: Durch Starten der Konsole und Aufrufen des Kommandos `gnuplot` wird die *Gnuplot* Kommandozeileneingabe geöffnet.

In *Gnuplot* können alle gängigen Rechenoperationen durchgeführt werden. Wie in Abbildung 5 dargestellt, kann mittels der Eingabe

```
print 2.0 * pi + sqrt(2) * 4**2
```

das Ergebnis der Rechnung

$$2\pi + \sqrt{2} * 4^2$$

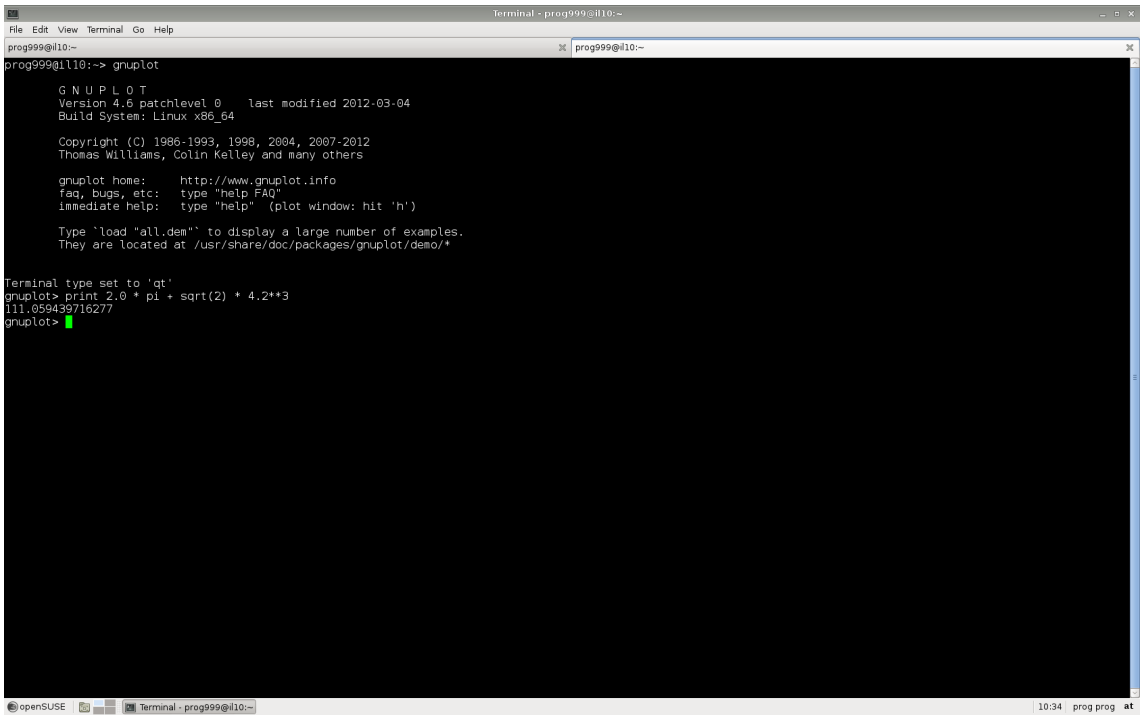

Abbildung 5: Mit *Gnuplot* lassen sich Rechenoperationen durchführen.

ausgegeben werden. Auch sind in *Gnuplot* mathematische Funktionen wie zum Beispiel sin, cos, tan, exp, log, log10, sqrt und viele mehr verfügbar.

Die Graphen verschiedenster Funktionen lassen sich in *Gnuplot* sehr einfach visualisieren. Mit der Eingabe

```
plot sin(x), cos(x), x**2
```

werden die Funktionen

$$f_1(x) = sin(x), \quad f_2(x) = cos(x) \quad \text{und} \quad f_3(x) = x^2$$

graphisch dargestellt, vgl. Abbildung 6.

Außerdem lassen sich in *Gnuplot* auf elegante Weise Inhalte von Textdateien darstellen. Die Testdaten für folgenden Dateiauszug wurden mittels der Funktion

$$f(x) = sin(x)$$

ermittelt. Die einzelnen Spalten der Daten werden durch beliebig viele *white spaces*, also zum Beispiel Leerzeichen, voneinander getrennt. So eine Datei sieht zum Beispiel folgendermaßen aus (die linke Spalte stellt die Stellen (x-Werte) und die rechte Spalte die Funktionswerte $f(x)$ dar):

```
0.000   0.000
0.628   0.588
1.256   0.951
1.884   0.951
2.512   0.589
...     ...
```

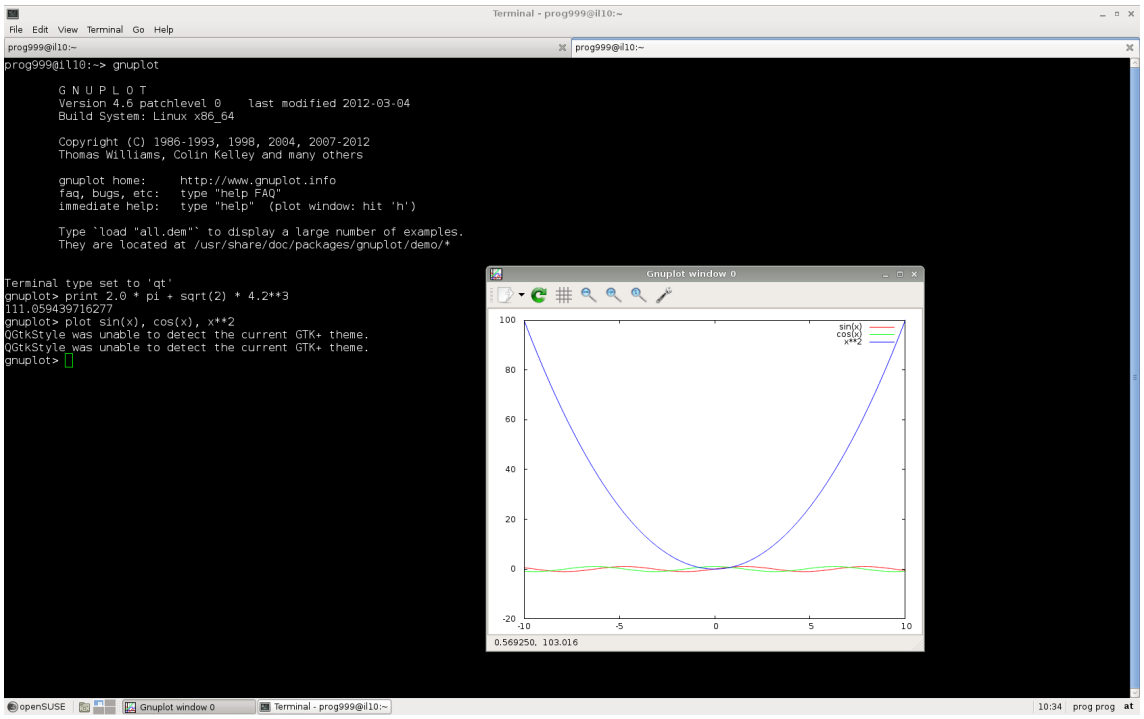

Abbildung 6: Plotten der trigonometrischen Funktionen `sin` und `cos` sowie der Funktion x^2.

Mit der Eingabe

```
plot 'test.dat' using 1:2 with lines
```

wird der Inhalt der Datei `test.dat` geplottet, wobei die x-Werte der ersten Spalte und die y-Werte der zweiten Spalte entnommen werden. Dies wird *Gnuplot* mit Verwendung des Schlüsselwortes `using` mitgeteilt. Eine Darstellung des Graphen findet sich in Abbildung 7

Sehr praktisch ist auch, dass mit der Pfeiltaste nach oben die letzten ausgeführten Befehle in der Kommandozeileneingabe wiederhergestellt werden können. So kann sehr schnell auf die letzten Eingaben erneut zugegriffen werden.

Mit der Eingabe `quit` wird *Gnuplot* beendet und mir der Konsoleneingabe kann fortgefahren werden.

3 Detect Unintended Memory Access (DUMA)

Es ist prinzipbedingt schwierig, alle Fehler, die möglicherweise auftreten können, zu testen und zu überprüfen, ob sich das Programm soweit wie möglich korrekt verhält. Es treten auch nicht alle Fehler mit der gleichen Häufigkeit auf. Leicht zu überprüfende Fehler sind in unserem konkreten Fall beispielsweise eine Division durch Null, wesentlich schwieriger zu überprüfen sind die Fälle, in denen z.B. `malloc()` Null zurückliefert. In diesen Fällen darf das Programm unter keinen Umständen "abstürzen", sondern muss eine definierte Fehlerbehandlung durchführen.

Die Wahrscheinlichkeit, dass `malloc()` in Ihrem Programm während der Übung Null zurückliefert, ist praktisch Null. Dies gilt nicht nur für die Übung, sondern ist auch sonst während der

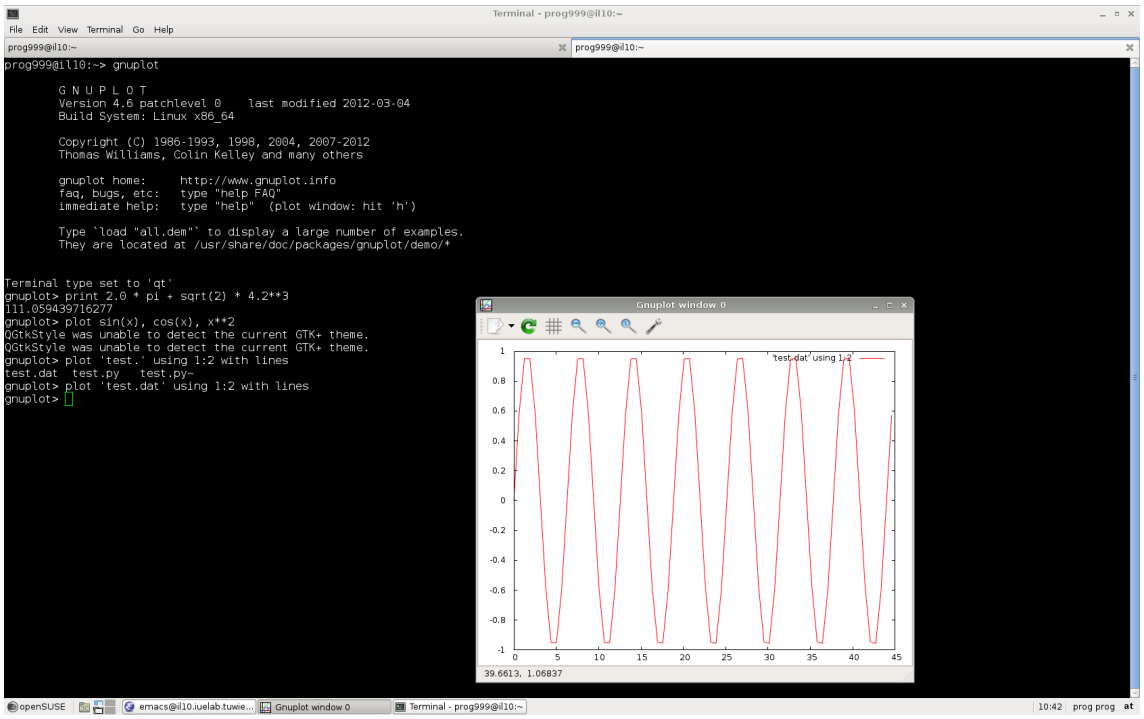

Abbildung 7: Plotten von Textdateiinhalten.

Programmentwicklung ein seltenes Ereignis. Das verleitet viele Programmierer dazu, die Überprüfung des Rückgabewertes nicht sehr ernst zu nehmen. Die Folge davon sind Programme, die funktionieren, solange kleine Datenmengen zu verarbeiten sind, aber abstürzen, sobald die erstellte Software in den produktiven Einsatz gebracht wird.

Während der Übung müssen Sie Ihrem Compiler mitteilen die Bibliothek DUMA zu linken und die Headerdatei `duma.h` inkludieren. Diese Bibliothek ersetzt die Systemfunktionen `malloc()` , `calloc()`, `realloc()` und `free()` durch eigene Varianten, die bei falscher Verwendung eine Fehlermeldung zurückliefern. Für die Übung haben wir DUMA so verändert, dass über eine UNIX Umgebungsvariable die Fehlerwahrscheinlichkeit für die Speicherallozierung eingestellt werden kann. Im Konkreten müssen Sie folgendes tun:

In Code::Blocks im Menü `Project/Build Options` beim Punkt `Linker Settings` unter `Other linkeroptions` die Zeile `-lm -lduma -lpthread` eintragen. Hierbei bezieht sich `-lm` auf die Mathematikbibliothek und die zwei folgenden Einträge auf die DUMA– Bibliothek. Wie Sie sehen, funktioniert das Hinzufügen dieser Bibliothek analog zur Mathematikbiliothek.

Um das Programm mit gesetzter Umgebungsvariable `DUMA_MALLOC_ERROR` zu starten, führen sie die Zeile

```
DUMA_MALLOC_ERROR=50 ./<Programmname>
```

in der Konsole aus. Der Wert der Umgebungsvariable `DUMA_MALLOC_ERROR` (hier:50) gibt die Wahrscheinlichkeit an, dass ein `malloc() 0` zurückliefert, also fehlschlägt. Der Wert 100 entspricht dabei dem Fall, dass immer Null zurückgeliefert wird, der Wert 0 dem Fall, dass kein Fehler erzeugt wird.

Sie bekommen beim Ausführen ihres Programms zusätzlich nützliche Ausgaben von DUMA. Im Anschluss zeigen wir ein Beispielprogramm:

```
long *ptr1;
long *ptr2;
if (!(ptr1 = malloc(10 * sizeof(long))))
{ fprintf(stderr, "Fehler: Speicher konnte nicht angefordert werden
   !\n");
  exit(-1);
}
/* ... */
ptr1 = ptr2;  /* Fehler: Verlust des Speicherbereichs von ptr1 */
/* ... */
free(ptr1);
```

DUMA liefert Ihnen dazu folgende (oder eine sehr ähnliche) Ausgabe:

```
DUMA 2.4.27 (static library)
Copyright (C) 2002-2006 Hayati Ayguen <h_ayguen@web.de>, Procitec GmbH
Copyright (C) 1987-1999 Bruce Perens <bruce@perens.com>

DUMA: ptr=0x2b4d542d1fb0 size=80 alloced from memory_leak.c(9) not freed
DUMA Aborting: DUMA_delFrame(): Found non free'd pointers.
```

Mit der Ausgabe "`Found non free'd pointers`" wird signalisiert, dass nicht alle dynamisch allozierten Speicherbereiche freigegeben worden sind. Diese Meldung darf bei korrekt funktionierenden Programmen nicht auftreten!