

I.1 Überblick

- gdb

I.2 Debuggen mit dem gdb

I.2 Debuggen mit dem gdb

- Programm muß mit der Compileroption `-g` übersetzt werden

```
gcc -g -o hello hello.c
```

- Aufruf des Debuggers mit `gdb <Programmname>`

```
gdb hello
```

- im Debugger kann man u.a.
 - ◆ Breakpoints setzen
 - ◆ das Programm schrittweise abarbeiten
 - ◆ Inhalt Variablen und Speicherinhalte ansehen und modifizieren
- Debugger außerdem zur Analyse von core dumps
 - ◆ Erlauben von core dumps:
z. B. `limit coredumpsize 1024k` oder `limit coredumpsize unlimited`

1 Breakpoints

- Breakpoints:
 - ◆ **b** <Funktionsname>
 - ◆ **b** <Dateiname>:<Zeilennummer>
 - ◆ Beispiel: Breakpoint bei main-Funktion

```
b main
```

- Starten des Programms mit **run** (+ evtl. Befehlszeilenparameter)
- Schrittweise Abarbeitung mit
 - ◆ **s** (step: läuft in Funktionen hinein) bzw.
 - ◆ **n** (next: läuft über Funktionsaufrufe ohne in diese hineinzusteppen)
- Fortsetzen bis zum nächsten Breakpoint mit **c** (continue)
- Breakpoint löschen: **delete** <breakpoint-nummer>

2 Variablen, Stack

- Anzeigen von Variablen mit **p** <variablenname>
- Automatische Anzeige von Variablen bei jedem Programmhalt (Breakpoint, Step, ...) mit **display** <variablenname>
- Setzen von Variablenwerten mit **set** <variablenname>=<wert>
- Ausgabe des Stack-Traces: **bt**
- Navigieren zwischen den Stackframes: **up**, **down**

3 Emacs und gdb

- gdb lässt sich am komfortabelsten im Emacs verwenden
- Aufruf mit "**ESC-x gdb**" und bei der Frage "**Run gdb on file:**" das mit der **-g**-Option übersetzte ausführbare File angeben
- Breakpoints lassen sich (nachdem der gdb gestartet wurde) im Buffer setzen, in welchem das C-File bearbeitet wird: **CTRL-x SPACE**

I.3 Electric Fence

- Speicherprobleme (SIGSEGV!) lassen sich mit der Electric Fence-Bibliothek gut finden:

```
gcc -g -o hello hello.c -L/proj/i4sp/pub/efence -leference
```

- Programm danach im Debugger laufen lassen