

Inhalt:

1. Debuggen mit DDD: Ein Beispiel.....	1
2. Programme ausführen.....	3
3. Daten untersuchen.....	5
4. Der Stack	6
5. Programmzustand verändern	7

1. Debuggen mit DDD: Ein Beispiel

Wir betrachten einen leistungsfähigen Debugger, den GNU-Debugger (GDB). GDB ist ein interaktives Programm, welches über Befehle mittels der *Kommandozeile* gesteuert wird. Um die Bedienung von GDB zu vereinfachen, stehen *graphische Benutzeroberflächen* sog. IDEs zur Verfügung.

Wir möchten hier die komfortable Oberfläche für den GDB, den GNU *Data Display Debugger* oder kurz DDD, betrachten. Gegeben sei ein kleines Beispielprogramm, an dem wir die Funktionalität des DDD erläutern.

```
/* sample.c - einfaches Testprogramm */

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

static void shell_sort(int a[], int size)
{
    int i, j;
    int h = 1;
    do
    {
        h = h * 3 + 1;
    } while (h <= size);
    do
    {
        h /= 3;
        for (i = h; i < size; i++)
        {
            int v = a[i];
            for (j=i; j>=h && a[j-h]>v; j-=h)
                a[j] = a[j - h];
            if (i != j)
                a[j] = v;
        }
    } while (h != 1);
}
```

```

int main(int argc, char *argv[])
{
    int *a;
    int i;
    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);
    shell_sort(a, argc);
    for (i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");
    free(a);
    return 0;
}

```

Dieses Programm sollte eigentlich die übergebenen Argumente sortieren und ausdrucken wie im Beispiel:

```

$ ./sample 8 7 5 4 1 3
1 3 4 5 7 8

```

Leider tut es dies nicht, wie unten zu sehen ist:

```

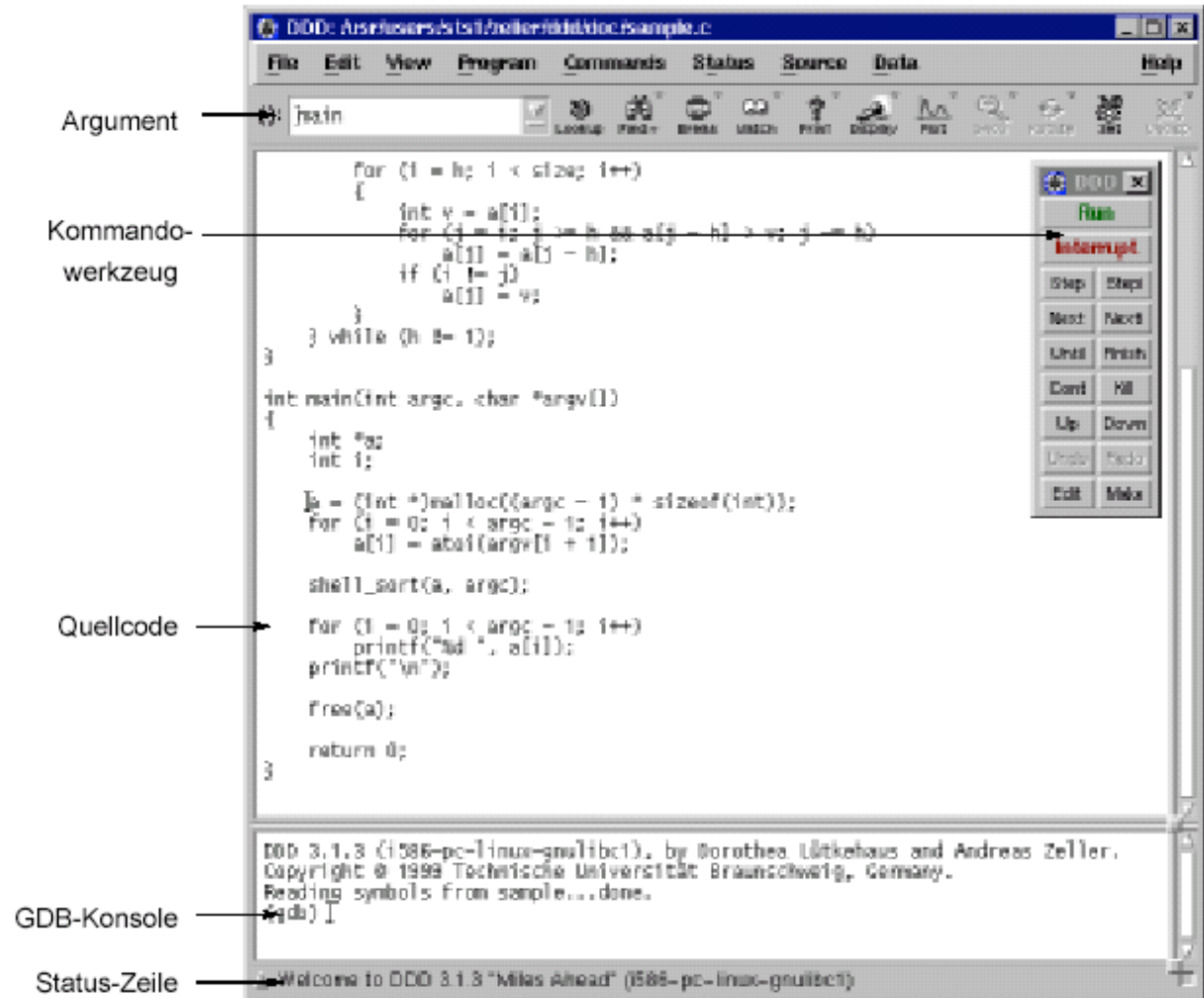
$ ./sample 8000 7000 5000 1000 4000
1000 1913 4000 5000 7000

```

2. Programme ausführen

Mit DDD untersuchen wir nun unser Beispielprogramm `sample`. Zunächst müssen Debugging-Informationen beim compilieren erstellt werden. Dies geschieht beim Übersetzen mittels folgendem Kommandoparameter (-g):

```
$ gcc -g -o sample sample.c
```



Nun kann man DDD auf das Programm loslassen.

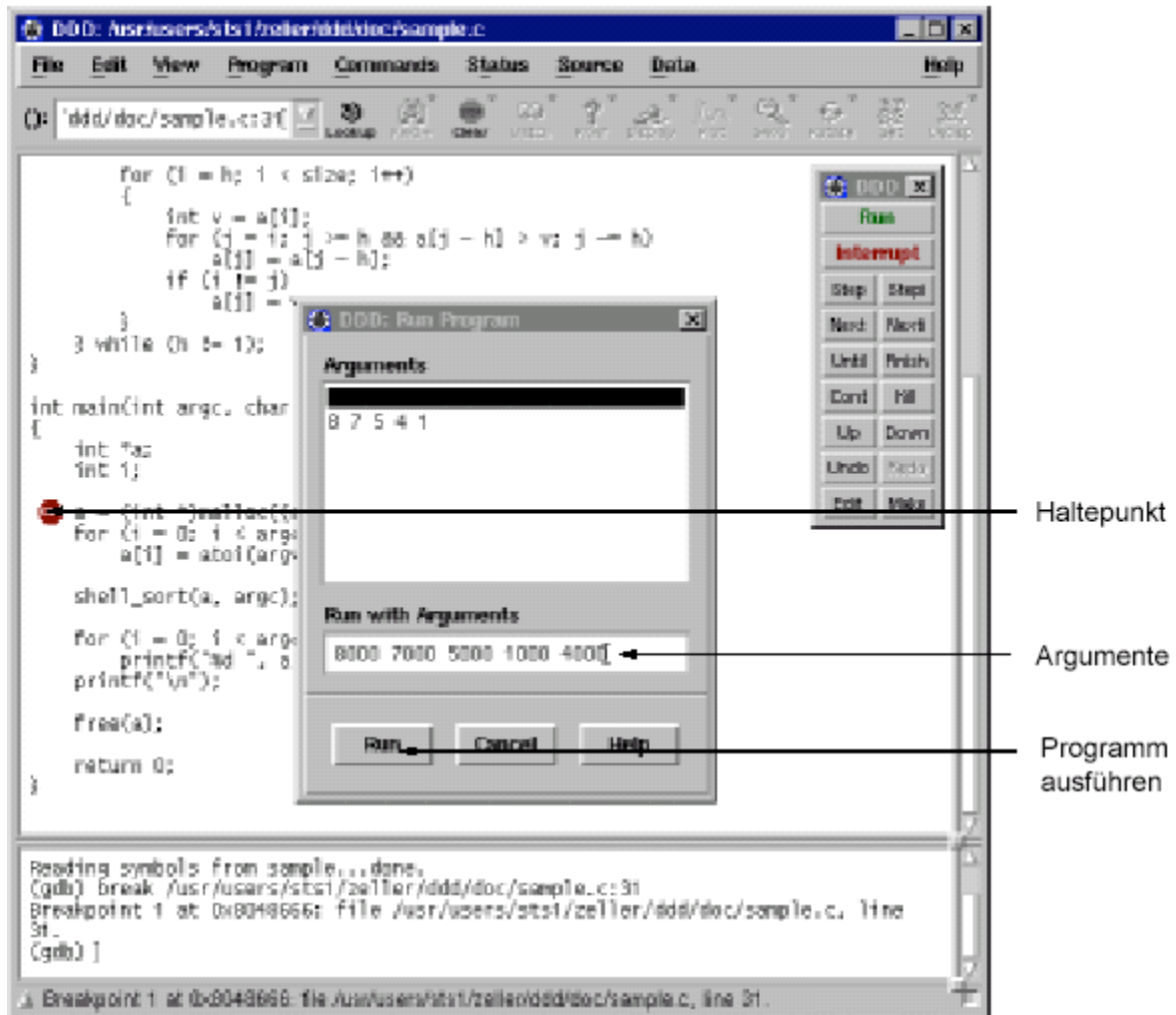
```
$ ddd sample
```

Die obige Abbildung zeigt die DDD-Oberfläche. Innerhalb von *Quellcode* steht unser Programm `sample`.

Die *GDB-Konsole* (am unteren Rand) zeigt die DDD-Version und die GDB-Eingabeaufforderung (`gdb`).

Nun können wir das Programm unter DDD (bzw. GDB) ausführen. GDB kennt mehrere Möglichkeiten, *Abbruchbedingungen* anzugeben. Die einfachste ist, das Programm anzuhalten, mittels eines *Haltepunkts*.

Man setzt einfach an jene Stelle einen Haltepunkt, an dem man das Programm stoppen möchte.



Durch Klicken auf *Break* erhalten wir einen Haltepunkt an der Position, wo sich der Cursor befindet. Nun müssen wir das Programm starten. Mit *Program_Run* führen wir das Programm aus (und übergeben im erscheinenden Dialog die Argumente). Abschließend betätigen wir den Button *Run* um das Programm zu starten.
 Nach einem kurzen Moment hält das Programm am gesetzten Haltepunkt an.

Anhand der Konsolenmeldungen können wir den aktuellen Zustand des GDB erkennen.

```
gdb) break sample.c:31
Breakpoint 1 at 0x8048666: file sample.c, line 31.
```

Nun geben wir folgende Anweisung innerhalb der GDB-Konsole ein:

```
gdb) run 8000 7000 5000 1000 4000
Starting program: sample 8000 7000 5000 1000 4000
Breakpoint 1, main (argc=6, argv=0xbffff918)
at sample.c:31
gdb)
```


Mit *Next* geht es zur nächsten Anweisung:

```
for (i = 0; i < argc - 1; i++)
    a[i] = atoi(argv[i + 1]);
```

Durch mehrmaliges Betätigen von *Next* können wir sehen, wie sich die Werte von *a* ändern. Weiter geht es mit *Next*:

```
for (i = 0; i < argc - 1; i++)
    printf("%d ", a[i]);
```

Dabei fällt uns auf, das *a* den Wert 1000, 1913, 4000, 5000, 7000 hat. Dies lässt uns vermuten, das innerhalb der Funktion `shell_sort` irgendetwas nicht stimmt.

4. Der Stack

Nun löschen wir den alten Haltepunkt (auf den Haltepunkt klicken, gefolgt von *Clear*) und setzen dafür einen Neuen neben dem Aufruf von `shell_sort`. Anschließend starten wir das Programm erneut.

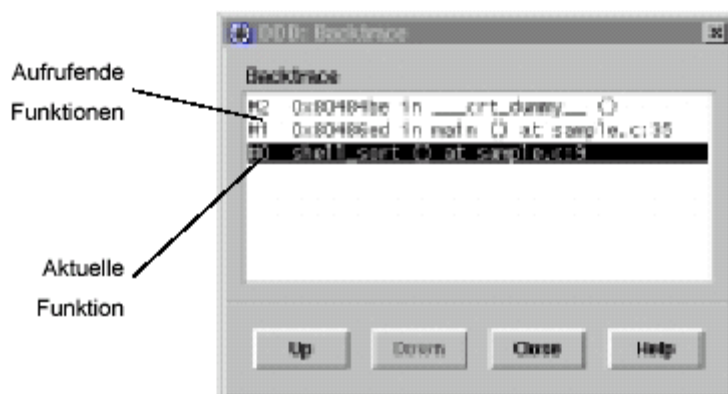
Dieses Mal wollen wir uns ansehen, was `shell_sort` eigentlich tut. Mit *Step* springen wir in die Funktion hinein und landen bei der ersten Anweisung innerhalb der Funktion.

```
int i, j;
```

während GDB uns mitteilt, wo wir gerade sind:

```
gdb) step
shell_sort (a=0x8049878, size=6) at sample.c:9
gdb)
```

Diese Ausgabe ist ein Teil des *Stacks*, auf dem Funktionen während der Ausführung mit ihren Argumenten und ihre lokalen Variablen abgelegt werden. Mit *Status_Backtrace* können wir den gesamten Stack betrachten (siehe Abbildung).



Die Debugger GDB und DDD zeigen nur die lokalen Variablen einer Funktion an. Die lokale Variable *a* wird nur angezeigt, wenn auch ihre zugehörige Funktion `main` im Stack ausgewählt ist. Bevor wir die Anweisungen von `shell_sort` betrachten, überprüfen wir noch die übergebenen Argumente der Funktion.

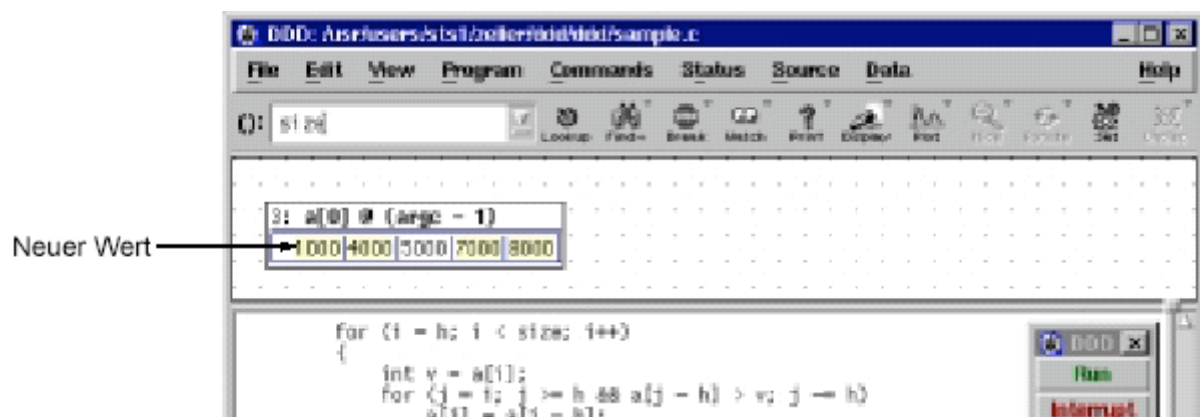
Geben wir also folgendes Kommando innerhalb der GDB-Konsole ein:

```
gdb) print a[0] @ size
$4 = {8000, 7000, 5000, 1000, 4000, 1913}
gdb)
```

Was sehen wir zu unserer Überraschung? Irgendwie ist der zusätzliche Wert 1913 mit eingefloßen! Der Grund ist ja recht simpel. Die Feldgröße ist *um 1 zu groß* – deshalb ist 1913 ein zufälliger Wert aus dem Speicher.

5. Programmzustand verändern

Korrigieren wir den falschen `size`-Wert per Hand, indem wir `size` im Quellcode auswählen und auf *Set* klicken.



Im Dialog können wir nun den falschen Wert 6 durch den korrekten Wert 5 ersetzen. Mit *Finish* beenden wir die Ausführung von `shell sort`:

```
gdb) set variable size = 5
gdb) finish
Run till exit from #0
shell sort (a=0x8049878, size=5)
at sample.c:9
0x80486ed in main (argc=6, argv=0xbffff918)
at sample.c:35
```

Super, es hat geklappt! Nun hat `a` den korrekten Wert. Mittels *Cont* setzen wir das Programm fort und sehen, daß `sample` die Werte tatsächlich ausgibt:

```
gdb) cont
1000 4000 5000 7000 8000
Program exited normally.
```

Nun können wir den Fehler beheben. Korrigieren wir nun das falsche 2. Argument

```
shell sort(a, argc);
```

in das korrekte

```
shell sort(a, argc - 1);
```

Nach der Neuübersetzung

```
$ gcc -g -o sample sample.c
```

können wir in DDD das Programm neu starten und sehen, ob wir erfolgreich waren:

```
gdb) run
'sample' has changed; re-reading symbols.
Reading in symbols . . . done.
Starting program: sample 8000 7000 5000 1000 4000
1000 4000 5000 7000 8000
Program exited normally.
```

Damit haben wir den vermeintlichen Fehler gefunden und können unsere DDD-Sitzung beenden.