Einfaches Beispiel für den Umgang mit vim/gcc/gdb

Version: 1.9(01.10.2018)
Autor: Franz Regensburger

Voraussetzungen:
- Abarbeitung der Anleitung
  10_VirtuelleMaschine-Mint183.pdf
- Rudimentäre Kenntnisse des Editors vim und der C-Programmierung

Einleitung

Diese Datei zeigt an einem Beispiel den Umgang mit der Entwicklungsumgebung vim/gcc/gdb

Schritt 1: Verzeichnis TestDir einrichten

Öffnen Sie eine Shell und legen Sie ein Verzeichnis für das folgende Beispiel an.

```
$ cd
$ mkdir -p ~/TestDir
$ cd ~/TestDir
```

Schritt 2: C-Source-Code eingeben im Vim

Öffnen Sie in einer Shell mit dem Editor vim die Datei loop.c

```
$ vim loop.c
```

und geben Sie folgenden Source-Code ein:
(Mit 'i' in den Insert-Modus wechseln).

```
#include <stdio.h>

int add(int a, int b) {
    return a+b;
}

int main() {
    int i,j;
    for (i = 0; i<10; i++) {
        j = i + 10;
        printf("i=%d j=%d i+j=%d\n",i,j,add(i,j));
    }
    return 0;
}
```

Nach Eingabe des Codes das Speichern nicht vergessen (:w) !

Schritt 3: C-Source-Code kompilieren

Öffnen Sie in einem neuen Fenster eine zweite Shell und wechseln Sie dann ins Verzeichnis TestDir. In der ersten Shell ist immer noch der Vim mit dem Source-Code der Datei loop.c geöffnet.

Hinweis: da wir später den Debugger benutzen wollen, ist die Verwendung eines zweiten Tabs in einem einzigen Terminal-Fenster nicht zu empfehlen. Öffnen Sie ein separates zweites Fenster.

```
$ cd ~/TestDir                (wechselt ins Beispiel-Verzeichnis)
$ ls loop.c                   (sind wir hier auch richtig?)
$ gcc -g -o loop loop.c       (Datei loop.c kompilieren mit Debug-
                               Symbolen '-g', Executable in Datei
                               loop erzeugen)
```

Hinweis:
Mit dem Befehl 'man gcc' erhalten Sie die Manual-Page zum gcc.

Schritt 4: Executable ausprobieren

In der zweiten Shell, in der Sie, hoffentlich erfolgreich, die ausführbare Datei loop (unter Linux/MacOS nur loop ohne Endung .exe) erzeugt haben, starten Sie nun die ausführbare Datei.

```
$ ./loop
```

Ausgabe:

```
i=0 j=10 i+j=10
i=1 j=11 i+j=12
i=2 j=12 i+j=14
i=3 j=13 i+j=16
i=4 j=14 i+j=18
i=5 j=15 i+j=20
i=6 j=16 i+j=22
i=7 j=17 i+j=24
i=8 j=18 i+j=26
i=9 j=19 i+j=28
```

Schritt 5: Programm loop debuggen in gdb

In der zweiten Shell, in der Sie, die ausführbare Datei loop erzeugt und bereits getestet haben, starten Sie nun den Debugger gdb mit der ausführbaren Datei. In der ersten Shell ist immer noch der Vim mit dem Source-Code der Datei loop.c geöffnet.

```
$ gdb -q ./loop        # Hinweis: verwenden Sie für diese Übung nicht den tui-Modus
```

Ausgabe:

```
Reading symbols from /home/lars/TestDir/loop...done.
(gdb)
```

Der GDB ist gestartet, und das Programm loop kann nun mit dem Debugger untersucht werden.

Geben Sie den Befehl ‘run’ ein. Das führt das Programm loop aus.

```
(gdb) run
Starting program: /home/lars/TestDir/loop
i=0 j=10 i+j=10
i=1 j=11 i+j=12
i=2 j=12 i+j=14
i=3 j=13 i+j=16
i=4 j=14 i+j=18
i=5 j=15 i+j=20
i=6 j=16 i+j=22
i=7 j=17 i+j=24
i=8 j=18 i+j=26
i=9 j=19 i+j=28

[Inferior 1 (process 13610) exited normally]

(gdb)
```

Breakpoint für Funktion ‘int add(int a, int b)’ setzen:

```
(gdb) break add
Haltepunkt 1 at 0x80483e7: file loop.c, line 4.
```

Bis zum Breakpoint ausführen:

```
(gdb) run

Starting program: /home/lars/TestDir/loop

Breakpoint 1, add (a=0, b=10) at loop.c:4
4         return a+b;
```

Vergleichen Sie die Ausgabe mit dem Source-Code im ersten Shell-Fenster:
Die Ausführung hält beim ersten Erreichen des Breakpoints:
(Zeile 4 im Source-Code)
Die aktuellen Parameter der Funktion add sind wie folgt belegt:
a=0 und b=10

Programm schrittweise fortsetzen durch mehrfache Eingabe des Kommandos ‘n‘ (für next)

```
(gdb) n
5     }
(gdb) n
i=0 j=10 i+j=10
main () at loop.c:8
8         for (i = 0; i<10; i++) {
```

```
(gdb) n
9             j = i + 10;
(gdb) n
10            printf("i=%d j=%d i+j=%d\n",i,j,add(i,j));
(gdb) n

Breakpoint 1, add (a=1, b=11) at loop.c:4
4         return a+b;
```

Nach mehreren Schritten sind wir wieder beim Breakpoint in Zeile 4 angelangt.

Wir lassen nun das Programm ohne Unterbrechung bis zum nächsten Erreichen des Breakpoints ausführen.

```
(gdb) cont
Continuing.
i=1 j=11 i+j=12

Breakpoint 1, add (a=2, b=12) at loop.c:4
4           return a+b;
(gdb)
```

Nun führen wir nur ein paar Einzelschritte aus, bis wir in Zeile 9 angelangt sind, also direkt vor Ausführung der Anweisung j=i+10.

```
(gdb) n
5     }
(gdb) n
i=2 j=12 i+j=14
main () at loop.c:8
8         for (i = 0; i<10; i++) {
(gdb) n
9             j = i + 10;
(gdb)
```

Jetzt betrachten wir den Inhalt der Variablen i und j mittels Kommando ‘p‘ (print):

```
(gdb) p i
$1 = 3
(gdb) p j
$2 = 12
(gdb)
```

Die Variable j ist noch mit dem Wert 12 belegt, da die Anweisung j = i+10 ja erst ausgeführt werden muss.
Wir stehen direkt davor in Zeile 9.

Wir führen die Zuweisung j = i+10 aus und testen danach die Belegung von j.

```
(gdb) n
10              printf("i=%d j=%d i+j=%d\n",i,j,add(i,j));
(gdb) p j
$3 = 13
(gdb)
```

Wir ändern die Belegung der Variablen i von 3 auf 8
Damit manipulieren wir den normalen Programmablauf!
Achtung: die Eingabe von ‚`p`' vor der Zuweisung ‚`i=8`' am Prompt des gdb ist notwendig.

```
(gdb) p i=8
$4 = 8
(gdb)
```

Wir löschen den Breakpoint für die Funktion add und führen das Programm bis zum Ende aus

```
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x080483e7 in add at loop.c:4
      breakpoint already hit 3 times
(gdb) clear add
Haltepunkt gelöscht 1
(gdb) cont
Continuing.

i=8 j=13 i+j=21
i=9 j=19 i+j=28
[Inferior 1 (process 13620) exited normally]
(gdb)
```

Aufgrund unserer Manipulation der Schleifenvariablen i wurden die Schleifendurchläufe für i=3 bis i=7 übersprungen. Die Variable j und die Summe ändern sich daher dementsprechend.
Vergleichen Sie die Ausgabe mit der Ausgabe eines ungestörten Ablaufs (siehe Ausgabe nach Eingabe des nächsten Befehls).

```
(gdb) run
Starting program: /home/lars/TestDir/loop
i=0 j=10 i+j=10
i=1 j=11 i+j=12
. . .
i=7 j=17 i+j=24
i=8 j=18 i+j=26
i=9 j=19 i+j=28
[Inferior 1 (process 13633) exited normally]
(gdb)
```

Wir verlassen den Debugger
(gdb) q

Viele weitere Informationen zur Bedienung des gdb finden sich im Manual des gdb.

Links für den Download: (zuletzt 01.10.2018)
http://sourceware.org/gdb/download/onlinedocs/gdb.pdf.gz
http://sourceware.org/gdb/download/onlinedocs/refcard.pdf.gz

Die komprimierten Dateien kann man in der Shell mit 'gunzip' entpacken.

Hinweis:

In der virtuellen Maschine ist eine Version des gdb installiert, die auch ein einfaches CLI auf Basis von Curses anbietet, das sogennate TUI (Text User Interface).

Diese Variante des GDB starten sie so:

$ gdbtui -q ./loop