Universität Augsburg
Institut für Informatik

**Übung zur Vorlesung Informatik 1** WiSe 2023/2024
Fakultät für Angewandte Informatik
Institut für Informatik
Prof. Dr. J. Hähner, P. Schalk, V. Schmid, S. Lutz, J. Linne 24.01.2024

# Betreutes Programmieren 11

Heute sollen Sie lernen, C-Programme zu debuggen. Mithilfe des GNU Debuggers **GDB** können kompilierte C-Programme Schritt für Schritt durchlaufen werden. Auf diese Weise kann der konkrete Programmablauf nachvollzogen und Programmfehler gefunden werden. Ganz konkret können beispielsweise

- der *Programmablauf gesteuert* (mittels sog. Breakpoints)
- der *Speicher ausgelesen* (mittels sog. Watchpoints)
- und *Variablen modifiziert* werden.

Nachfolgende Kurzübersicht zum GNU Debugger listet eine Auswahl der wichtigsten Befehle auf, die für das heutige Arbeitsblatt relevant sind. Eine vollständige Übersicht ist der Online-Dokumentation[1] zu entnehmen.

### Kurzübersicht GDB/LLDB

1. C-Programm `main.c` kompilieren (z.B. zu `program.exe`):
   `gcc main.c -o program -g`
   - *Der Parameter -g aktiviert die Debugging Symbole, die das Debugging erheblich vereinfachen.*
2. Debugger starten:

   **GDP**: `gdp program`
   **LLDB**: `lldb program`
3. Debugger beenden:

   `quit`
4. Programm mit Kommandozeilenparametern `param1` und `param2` starten:

   `run param1 param2`
5. Laufendes Programm beenden:

   `kill`

[1] https://sourceware.org/gdb/current/onlinedocs/gdb/

6. Übersicht über aktuellen Programmzustand abrufen:

   **GDB**: `info program`
   **LLDB**: `pr status`

7. *Breakpoints:* Programm in Zeile 80 der Quellcode-Datei `main.c` anhalten:

   **GDB**: `break main.c:80`
   **LLDB**: `b main.c:80`

8. *Breakpoints:* Programm bei Aufruf der Funktion `foo` anhalten:

   **GDB**: `break foo`
   **LLDB**: `b -n foo`

9. *Breakpoints:* Programm in Zeile 80 der Quellcode-Datei `main.c` anhalten, falls der Wert der Variable $x$ kleiner als 80 ist:

   **GDB**: `break main.c:80 if x < 50`
   **LLDB**: `br s -f main.c -l 80 -c 'x < 50'`

10. Angehaltenes Programm bis zum nächsten Breakpoint forsetzen:

    `continue`

11. Angehaltenes Programm Zeile für Zeile ausführen, wobei bei Funktionsaufrufen die zugehörigen Funktionen ebenfalls Zeile für Zeile ausgeführt werden:

    `step`

12. Angehaltenes Programm Zeile für Zeile ausführen, wobei bei Funktionsaufrufen die zugehörigen Funktionen **nicht** ebenfalls Zeile für Zeile ausgeführt werden:

    `next`

    *Tipp: Drücken der Enter-Taste führt den zuletzt verwendeten Befehl nochmal aus.*

13. Angehaltenes Programm führt die aktuelle Funktion komplett aus:

    `finish`

14. *Watchpoints*: Programm bei Änderung des Werts der Variable $x$ anhalten:

    **GDB**: `watch x`
    **LLDB**: `w s v x`

    *Hinweis: Das Programm muss zuerst an einer passenden Stelle angehalten werden.*

*Falls $x$ zum Beispiel eine lokale Variable der Funktion* `main` *ist, kann der Watchpoint auf $x$ erst bei Betreten dieser Funktion gesetzt werden. es ist also ein passender Breakpoint zu setzen.*

15. Übersicht über alle Break-/Watchpoints ausgeben lassen:

    **GDB**: `info breakpoints`
    **GDB**: `info watchpoints`
    **LLDB**: `b`
    **LLDB**: `w l`

16. Breakpoints entfernen:

    **GDB**:
    ```
    delete /* entfernt alle gesetzten Breakpoints */
    clear foo /* entfernt Breakpoint bei Funktionseintritt von foo */
    clear main.c:80 /* deaktiviert an Zeile 80 in main.c gesetzte Breakpoints */
    ```
    **LLDB**: `b`
    ```
    br del /* entfernt alle gesetzten Breakpoints */
    br del _ID_ /* entfernt Breakpoint mit der Nummer _ID_ */
    ```

17. Breakpoints (de-)aktivieren:

    **GDB**:
    ```
    disable p /* deaktiviert Breakpoint p */
    enable p /* aktiviert Breakpoint p */
    ```
    **LLDB**:
    ```
    br dis _ID_ /* deaktiviert Breakpoint mit der Nummer _ID_ */
    br en _ID_ /* aktiviert Breakpoint mit der Nummer _ID_ */
    w dis _ID_ /* deaktiviert Watchpoint mit der Nummer _ID_ */
    w en _ID_ /* aktiviert Watchpoint mit der Nummer _ID_ */
    ```

18. Wert der Variable $x$ ausgeben lassen:

    ```
    print x
    ```

    *Hinweis: Programm muss zuvor an passender Stelle angehalten werden.*

19. Array `arr` der Länge 4 mittels Dereferenzierung ausgeben lassen:

    **GDB**: `print *arr@4`
    **LLDB**: `parray 4 arr`

20. Wert der Variable $x$ auf 5 setzen:

    **GDB**: `set x = 5`
    **LLDB**: `expr x = 5`

    *Hinweis: Programm muss zuvor an passender Stelle angehalten werden.*

21. Backtrace ausgeben lassen:

    **GDB**: `backtrace`
    **LLDB**: `bt`

    *Hinweis: Nützlich zum Beispiel nach einem Programmabsturz, um herauszufinden, an welcher Stelle das Programm abestürzt ist.*

22. Informationen über (den Typ) des Ausdrucks `bar` abrufen:

    **GDB**: `explore bar`
    **LLDB**: `frame variable -T bar`

Für viele Befehle existieren Abkürzungen, welche Sie selbst nachschlagen können.

## Aufgabe 11 ** *(Debugging)*

(a)

In dieser Teilaufgabe sollen Sie verschiedene GDB-Befehle anhand des Beispielprogramms `bp10a.c` nachvollziehen. Nachfolgende Befehle sollen Sie bei der Abgabe live ausführen und erklären können. Die notwendigen Befehle entnehmen Sie der o.g. Kurzübersicht.

1. Kompilieren Sie die Datei `bp10a.c` mit Debugging-Symbolen.

2. Starten Sie den Debugger mit dem erzeugten Maschinencode.

3. Starten Sie einen Programmdurchlauf im Debugger und erklären Sie die Ausgabe.

4. Setzen Sie einen Breakpoint in der Funktion `search_unsorted`, so dass der Programmablauf zu Beginn jeder Iteration unterbrochen wird.

5. Lassen Sie sich in jedem Schleifendurchlauf den aktuellen Wert von `a[i]` ausgeben.
   *Tipp: In print-Befehlen kann Feldindizierung und Pointer-Schreibweise wie in C-Code verwendet werden.*

6. Löschen Sie den gesetzten Breakpoint.

7. Setzen Sie einen Watchpoint auf die lokale Variable max in der Funktion laufen Sie das Programm komplett und erklären Sie die Ausgabe des GDB bei jeder Unterbrechung.

8. Löschen Sie alle Breakpoints und setzen Sie einen neuen Breakpoint bei Aufruf der Funktion `array_swap`. Starten Sie das Programm neu.

9. Lassen Sie sich bei der ersten Programmunterbrechung einen Backtrace ausgeben, erklären Sie die Ausgabe und brechen Sie die Programmausführung komplett ab.

10. Lassen Sie sich eine Übersicht über alle momentan gesetzten Breakpoints ausgeben.

11. Deaktivieren Sie alle vorhandenen Breakpoints und setzen Sie einen neuen Breakpoint auf die Funktion `array_print`.

12. Starten Sie einen neuen Programmdurchlauf und durchlaufen Sie Ihr Programm ab Breakpoint Zeile für Zeile, wobei Sie...

    - Funktionsaufrufe als eine Zeile abarbeiten.
    - Funktionsaufrufe ebenfalls Zeile für Zeile abarbeiten.

    Erklären Sie den Unterschied zwischen beiden Varianten und beschleunigen Sie den Programmablauf nach wenigen Durchläufen durch gezielte Verwendung von `finish`.

13. Löschen Sie alle Breakpoints. Setzen Sie einen bedingten Breakpoint in Zeile 17, der nur verwendet wird, falls der Wert der Variable `suchzahl` größer als 10 ist.

14. Starten Sie einen neuen Programmdurchlauf und setzen Sie den Wert der Variable `suchzahl` am Breakpoint auf 3. Lassen Sie das Programm nun bis zum Ende durchlaufen.

15. Geschafft! Beenden Sie schließlich den Debugger.

(b)

In das beigefügte Programm `bp10b.c` haben sich leider zwei Fehler eingeschlichen. Finden und beheben Sie diese Fehler, indem Sie Ihr Programm debuggen! Für die Abgabe ist es entscheidend, dass Sie zeigen können, wie Sie den Debugger verwendet haben.

*Tipps*:

- *Beobachten Sie die Werte von* `c` *und* `in` *in der Funktion* `read_string`
- *Testen Sie Ihr Programm mit der Zeichenkette* `HalloWelt`

(c)

In das beigefügte Programm `bp10c.c` haben sich leider fünf Fehler eingeschlichen. Finden und beheben Sie diese Fehler, indem Sie Ihr Programm debuggen! Für die Abgabe ist es entscheidend, dass Sie zeigen können, wie Sie den Debugger verwendet haben.

*Tipps*:

- *Es ist nicht das Ziel, das Programm neu zu schreiben. Es sind lediglich vier kleine Änderungen notwendig.*
- *Beobachten Sie die Werte der Zählvariablen in* `pretty_print`
- *Beispielhafter Programmaufruf mit zugehöriger Ausgabe:*

Aufruf:

```
program 123
```

Ausgabe:

```
Anzahl Kommandozeilenparameter: 1
Als Parameter eingegebene Zahl: 123
program.exe
123
```