Tools zur Programmierung mit C

Software Entwicklung 1

Annette Bieniusa, Mathias Weber, Peter Zeller

Inhaltsverzeichnis

1 Compiler

Es gibt viele verschiedene C-Compiler. Unter Linux werden vor allem die GCC und Clang Compiler benutzt. Unter Windows ist auch der Visual C++ Compiler weit verbreitet.
In dieser Vorlesung verwenden wir den Clang Compiler, unter Windows empfehlen wir den GCC Compiler via MinGW zu installieren. Prinzipiell kann aber auch jeder andere Compiler verwendet werden, welcher den C99 Standard unterstützt.

1.1 Installation

1.1.1 Installation unter Linux

Unter den meisten Linux-System ist der Clang Compiler über einen Paketmanager installierbar. Zum Beispiel unter Ubuntu mit:

```
$ sudo apt-get install clang
```

1.1.2 Installation unter Mac

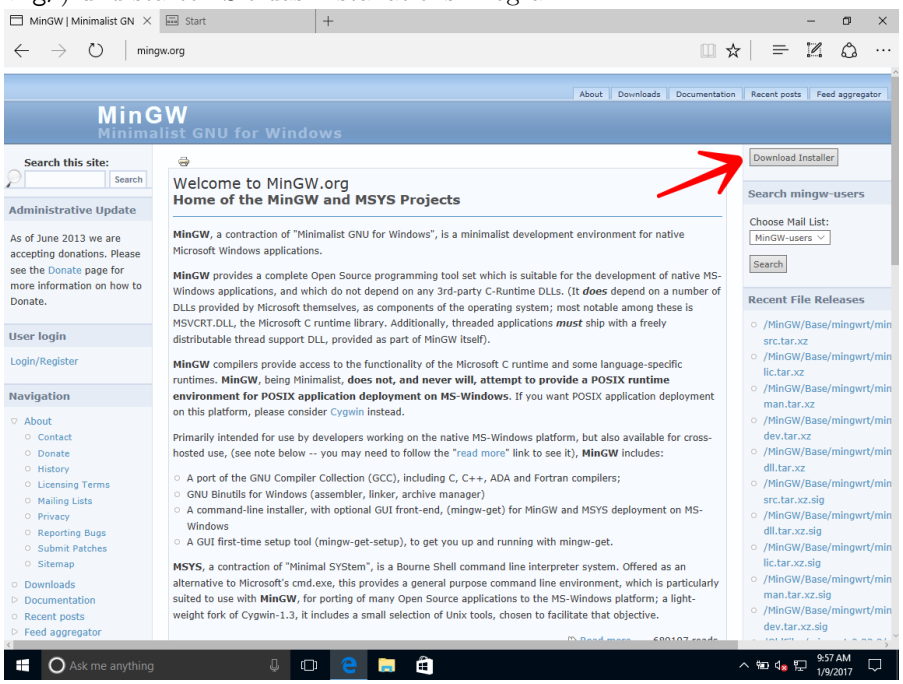
Unter Mac lässt sich Clang über Xcode installieren (Xcode lässt sich über den App Store installieren). Öffnen Sie ein Terminal-Fenster und geben Sie den folgenden Befehl ein:

```
$ xcode-select --install
```

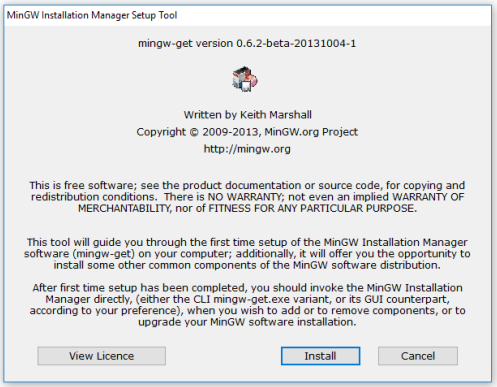
Danach öffnet sich ein Fenster, welches anbietet die Entwickler-Tools zu installieren.

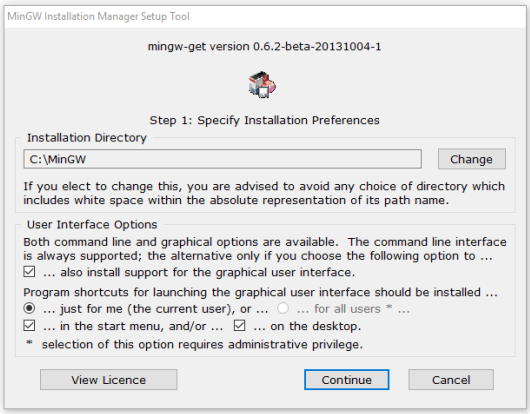
1.1.3 Installation unter Windows

Laden Sie sich MinGW herunter ("Download Installer" unter `http://www.mingw.org/`) und starten Sie das Installations-Programm.

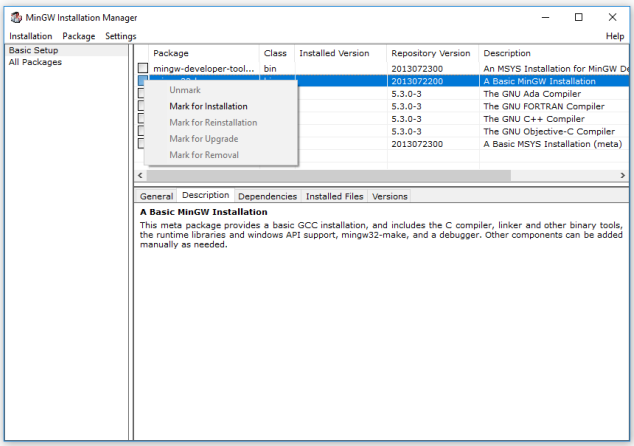


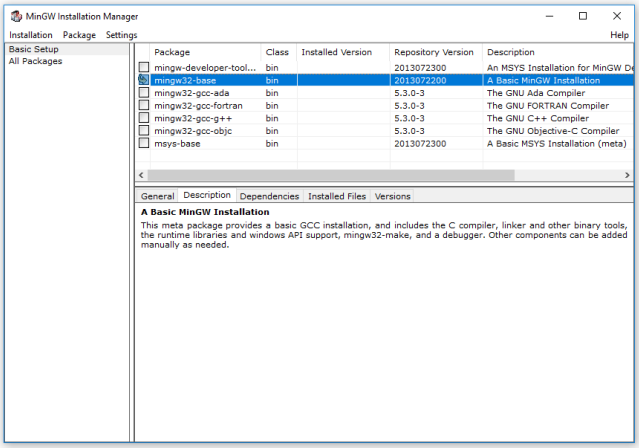
Belassen Sie die Installation bei den Standard-Einstellungen.



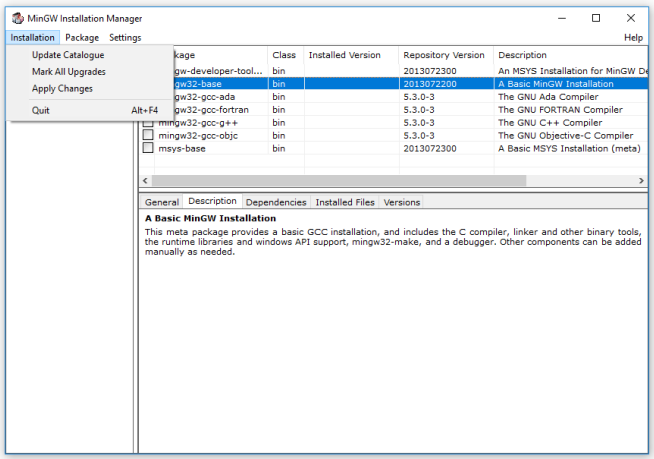


Nach der Installation öffnet sich der MinGW Installations Manager. Dieses Programm kann auch über das Windows-Startmenü erneut aufgerufen werden, zum Beispiel um später noch weitere Pakete zu installieren.
Verwenden Sie den Installations Manager um das das Paket `mingw32-base` zu installieren.

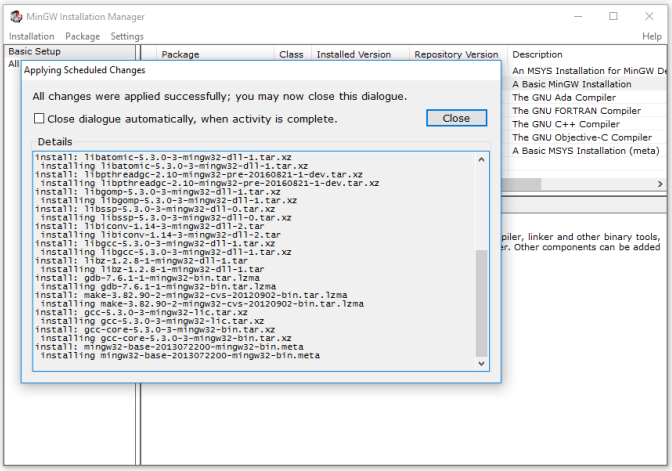




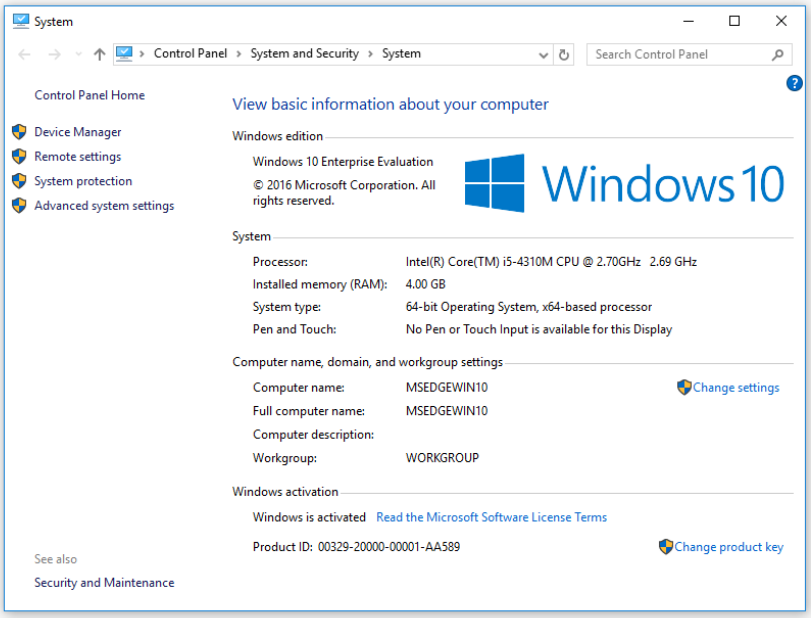
Wählen Sie “Apply Changes” um die Installation zu starten:



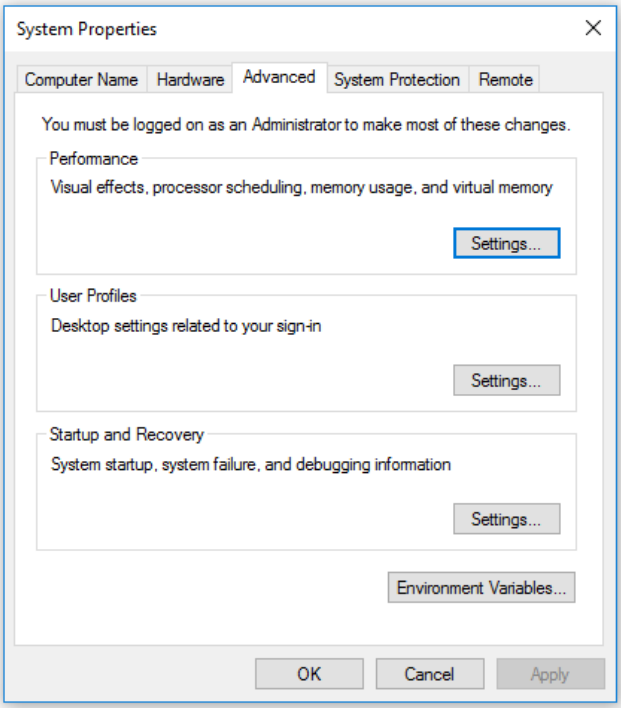
Nach Abschluss der Installation erscheint der folgende Dialog:



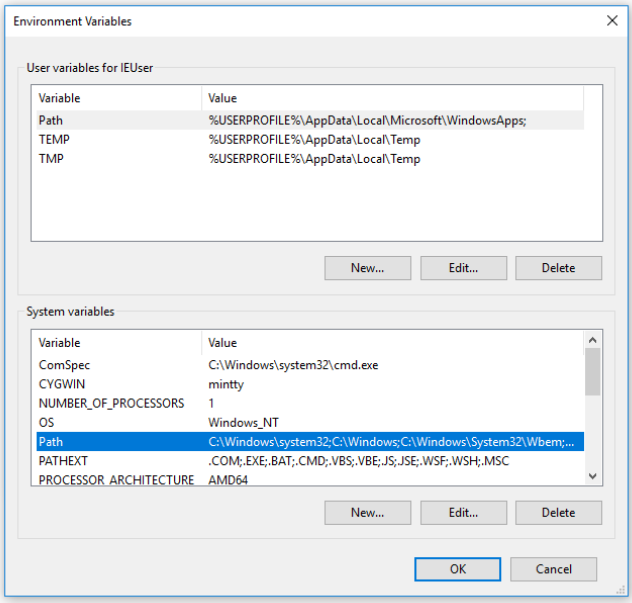
Damit Der installierte Compiler vom Terminal aus benutzt werden kann und von anderen Programmen gefunden wird, muss er in die Umgebungs-Variable `PATH` eingetragen werden.
Öffnen sie dazu mit der Tastenkombination Windows+Pause die Systemeinstellungen (alternativ können Sie auch nach “System” suchen):



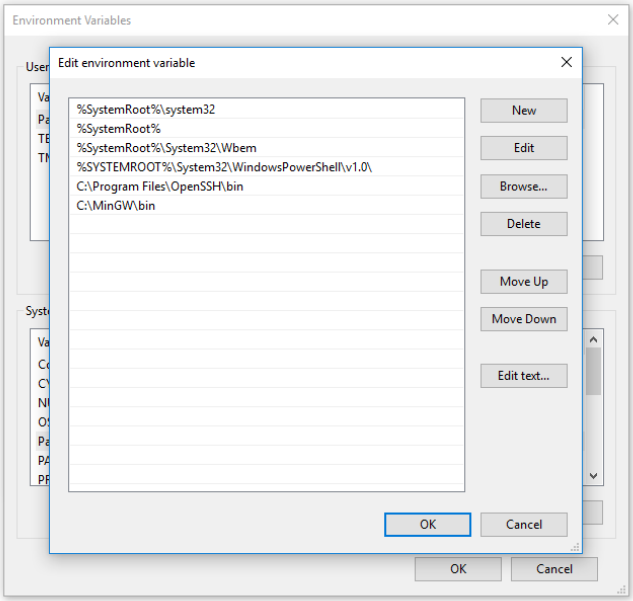
Wählen Sie dort “Advanced system settings” aus:



Gehen Sie dann zu “Environment Variables...”:



Wählen Sie hier unter “System variables” den Eintrag der Variablen “Path” aus und fügen Sie den Pfad “`C:\MinGW\bin`” hinzu. Wenn der Eintrag noch nicht existiert, erstellen Sie ihn neu.

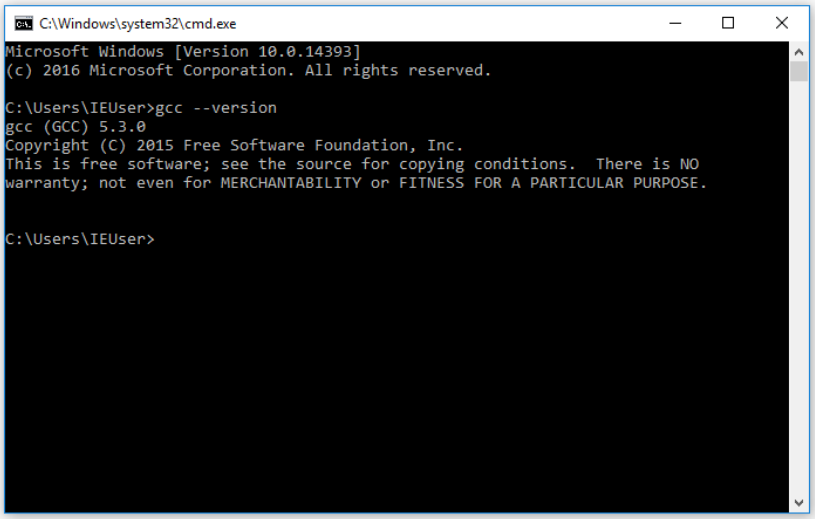


In älteren Windows-Versionen müssen die Einträge manuell in ein kleines Textfeld

durch Semikolons getrennt eingetragen werden.
Nach dem Ändern des Pfads muss das System eventuell neu gestartet werden, damit alle anderen Anwendungen den Compiler korrekt finden.

1.2 Testen der Installation unter Windows

Öffnen Sie ein Terminal und geben Sie den Befehl `gcc --version` ein. Es sollte dann die Versionsnummer angezeigt werden, zum Beispiel:



1.3 Testen der Installation unter Mac/Linux

Öffnen Sie ein Terminal und geben Sie den Befehl `clang --version` ein. Es sollte dann die Versionsnummer angezeigt werden, zum Beispiel:

```
C:\>clang --version
clang version 3.9.1 (branches/release_39)
Target: x86_64-pc-windows-msvc
Thread model: posix
InstalledDir: C:\Program Files\LLVM\bin
```

1.4 Verwendung des Compilers

Die folgende Anleitung bezieht sich auf den Clang Compiler. Für GCC kann in der Regel der Befehl `clang` durch `gcc` ersetzt werden.
In der einfachsten Variante wird dem Compiler einfach der Name der Quelldatei übergeben:

```
$ clang hello.c
```

In diesem Fall wird eine ausführbare Datei `a.out` (bzw. `a.exe`) erstellt. Mit der Option `-o` lässt sich ein anderer Name für die Ausgabe-Datei (output) angeben:

```
$ clang hello.c -o hello
```

Des weiteren ist es noch sinnvoll weitere Warnungen zu aktivieren, um mögliche Fehler bereits vor dem Ausführen des Programms zu bemerken. Mit der Option `-Wall` (Warnings: all) lässt sich eine Reihe von Warnungen aktivieren. Mit den Optionen `-Wextra` und `-Weverything` können noch weitere Warnungen aktiviert werden.
Die Option `-g` speichert Debug-Informationen im Code ab, so dass dieser besser mit einem Debugger (siehe unten) verwendet werden kann.
Die Option `-fsanitize=address` fügt dem Code Anweisungen zu, die undefinierte Speicherzugriffe erkennen und melden. Dies kann zum Finden von Fehlern hilfreich sein.
Das folgende Beispiel zeigt einen Aufruf des Compilers mit allen Warnungen und den anderen zuvor genannten Optionen:

```
$ clang hello.c -o hello -g -Weverything -fsanitize=address
```

2 Debugger

Ein Debugger ist ein Programm, mit dem sich die Ausführung eines Programms steuern und beobachten lässt. Damit können Fehler im Programm gefunden und das Verhalten eines Programms genauer verstanden werden.

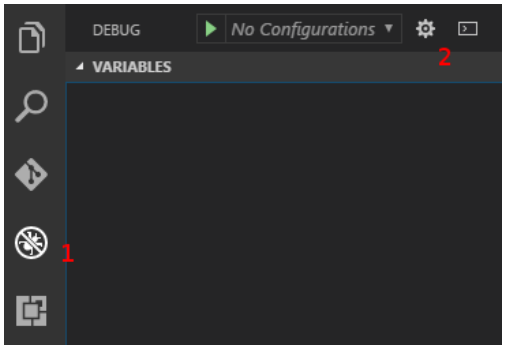
2.1 Debugger: Visual Studio Code

Visual Studio Code ist ein Editor, welcher auf `https://code.visualstudio.com/` heruntergeladen werden kann.
Um den Debugger für C zu benutzen muss eine Erweiterung installiert werden. Installieren Sie sich unter Linux/Mac die Erweiterung C/C++ von Microsoft [1] und unter Windows die Erweiterung Native Debug [2].

2.1.1 Einrichten des Debuggers

Wechseln Sie zur Debug-Ansicht (1) und erstellen Sie mit einem Klick auf das Zahnrad (2) eine `launch.json`-Konfigurations-Datei. Es öffnet sich ein Dialog. Wählen Sie unter Linux/Mac die Option “C++ (GDB/LLDB)” und unter Windows die Option “GDB”.

[1] `https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools`
[2] `https://marketplace.visualstudio.com/items?itemName=webfreak.debug`



In der `launch.json`-Datei müssen Sie noch das Programm eintragen.

Unter Linux/Mac ändern Sie den Eintrag `"program"` zu `"${workspaceRoot}/main"`, wobei `main` der Name der ausführbaren, kompilierten Datei ist.
Beispiel:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "C++ Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceRoot}/main",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceRoot}",
            "environment": [],
            "externalConsole": true,
...
```

Unter Windows ändern Sie den Eintrag `"target"` zur ausführbaren, kompilierten Datei. Fügen Sie außerdem einen neuen Eintrag `"terminal": "cmd"` hinzu, so dass die Ausgabe des Programms in einem neuen Terminal-Fenster angezeigt wird.
Beispiel:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Debug",
            "type": "gdb",
            "request": "launch",
            "target": "main.exe",
            "terminal": "cmd",
            "cwd": "${workspaceRoot}"
```

```
        }
    ]
}
```

2.1.2 Verwenden des Debuggers

Vor dem Verwenden des Debuggers müssen zuerst sogenannte Breakpoints festgelegt werden. Dies sind Programmstellen, an denen die Programm-Ausführung angehalten und die Kontrolle an den Debugger übergeben wird.
Ein Breakpoint wird durch einen Klick links neben den Zeilennummern hinzugefügt und entfernt.

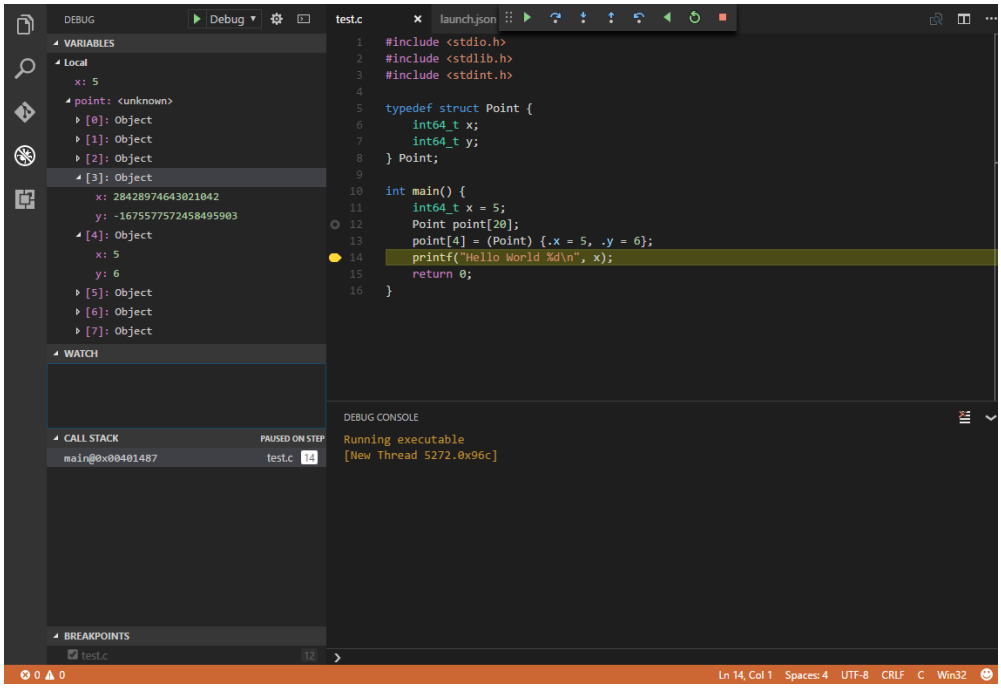
Vor dem Starten des Debuggers muss zuerst das Programm mit der Option `-g` kompiliert werden, zum Beispiel:

```
$ clang main.c -g -o main
```

oder unter Windows mit GCC:

```
$ gcc main.c -g -o main.exe
```

Danach kann der Debugger über den grünen Start-Pfeil in der Debug-Ansicht oder mit der entsprechenden Tastenkombination (Standardeinstellung ist `F5`) gestartet werden. Die Ausführung hält dann beim ersten Breakpoint an. Die aktuelle Zeile wird jeweils hervorgehoben.



Mit den Steuer-Elementen am oberen Bildschirm-Rand lässt sich die Ausführung (schrittweise) fortsetzen. Es gibt Aktionen um die Ausführung bis zum nächsten Breakpoint fortzusetzen ("Continue"), die aktuelle Zeile auszuführen ("Step Over"), in die Funktion in der aktuellen Zeile zu gehen ("Step Into"), oder die aktuelle Funktion zu verlassen ("Step Out"). Des weiteren gibt es bei einem Rechtsklick im Quelltext noch ein Menü, in dem man die Aktion "Debug: Run to Cursor" auswählen kann, welche das Programm bis zur ausgewählten Zeile ausführt und dann erneut anhält.
Zu den Aktionen gibt es auch jeweils Tastenkombination, mit denen sich der Debugger schneller bedienen lässt.
In der linken Ansicht findet sich der Zustand der aktuellen lokalen Variablen, welche durch einen Doppelklick auf den Wert auch geändert werden können. Der Wert von Variablen wird auch angezeigt, wenn die Maus über eine Variable im Quelltext bewegt wird.
Der Abschnitt "Watch" darunter kann verwendet werden, um den Wert von Ausdrücken zu beobachten. Dort lassen sich eigene Ausdrücke angeben, die dann ausgewertet und angezeigt werden. Damit lassen sich zum Beispiel auch globale Variablen beobachten.
Der Abschnitt "Call Stack" zeigt die aktuell aktiven Funktionsaufrufe. Dort kann auch ein anderer aktiver Aufruf ausgewählt werden, um dessen lokale Variablen zu sehen.

3 Valgrind

Valgrind (`http://valgrind.org/`) ist ein Programm, mit dem Fehler in Programmen zur Laufzeit gefunden werden können.

3.1 Installation

Unter Linux kann Valgrind in der Regel über den Paket-Manager installiert werden, unter Mac via Homebrew.
Windows wird von Valgrind nicht unterstützt.

3.2 Benutzung

Beim Starten des Programms wird der Befehl `valgrind` vorangestellt. Als Beispiel betrachten wir das folgende Programm `main.c`, welches einige Fehler enthält:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[]) {
5      int *ar = malloc(20*sizeof(int));
6      ar[20] = 42;
7      printf("ar[5] = %d\n", ar[5]);
8      return 0;
9  }
```

Das Programm kann wie folgt kompiliert und mit Valgrind ausgeführt werden:

```
$ clang -g main.c -o main
$ valgrind ./main
```

Die Ausführung führt zu mehreren Warnungen, welche im folgenden kurz erkärt werden.

```
==28300== Invalid write of size 4
==28300==    at 0x4005A7: main (main.c:6)
==28300==  Address 0x5203090 is 0 bytes after a block of size 80 alloc'd
==28300==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==28300==    by 0x400594: main (main.c:5)
```

Die Warnung bezieht sich auf Zeile 6 in `main.c` und besagt, dass dort eine ungültige Schreib-Operation der Größe 4 Bytes erfolgt, und zwar auf den Speicherbereich, welcher 80 Bytes groß ist und in Zeile 5 von main.c angefordert wurde. In der tat schreiben wir in Zeile 6 an Array-Index 20, der maximale zulässige Index wäre aber 19.

```
==28300== Conditional jump or move depends on uninitialised value(s)
==28300==    at 0x4E87B83: vfprintf (vfprintf.c:1631)
==28300==    by 0x4E8F898: printf (printf.c:33)
==28300==    by 0x4005BB: main (main.c:7)
```

Diese Warnung bezieht sich auf den Zugriff `ar[5]` in Zeile 7 von `main.c`. Dort lesen wir einen Eintrag des Arrays, der noch nicht initialisiert wurde.

```
==28300== HEAP SUMMARY:
==28300==     in use at exit: 80 bytes in 1 blocks
==28300==   total heap usage: 2 allocs, 1 frees, 1,104 bytes allocated
==28300==
==28300== LEAK SUMMARY:
==28300==    definitely lost: 80 bytes in 1 blocks
==28300==    indirectly lost: 0 bytes in 0 blocks
==28300==      possibly lost: 0 bytes in 0 blocks
==28300==    still reachable: 0 bytes in 0 blocks
==28300==         suppressed: 0 bytes in 0 blocks
==28300== Rerun with --leak-check=full to see details of leaked memory
==28300==
==28300== For counts of detected and suppressed errors, rerun with: -v
==28300== Use --track-origins=yes to see where uninitialised values come from
==28300== ERROR SUMMARY: 9 errors from 9 contexts (suppressed: 0 from 0)
```

Am Ende der Ausgabe werden Informationen über den verbrauchten Speicher angezeigt. Die Ausgabe gibt an, dass wir 80 Bytes an Speicher verloren haben. In der Tat wurde der Speicher, welcher in Zeile 5 angefordert wurde, danach nicht mehr durch einen Aufruf von `free` freigegeben.
Valgrind kann mit der Option `--leak-check=full` ausgeführt werden, um mehr Informationen darüber zu erhalten, wo der Speicher angefordert wurde:

```
valgrind --leak-check=full ./main
...
==28756== 80 bytes in 1 blocks are definitely lost in loss record 1 of 1
==28756==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==28756==    by 0x400594: main (main.c:5)
```