Aufgaben für die Konsultationen – Programmierübungen in C

Hendrik Fehr

2. Mai 2024

1 Einrichten der Umgebung

1.1 Verwendete Software

Die verwendete Software ist Git, ein C-Compiler und *Code Composer Studio* als IDE[1]:

Git siehe vorherige Konsultation

C-Compiler Um C-Programme auf dem eigenen Rechner zu kompilieren und laufen zu lassen verwenden wir den C-Compiler der *Gnu Compiler Collection* gcc.gnu.org. Die Installation unter Linux ist einfach, während unter Windows mehrere Schritte notwendig sind. Nach erfolgreicher Installation lässt sich der Compiler im Terminal durch `$ gcc` von Hand starten. Wenn man eine IDE verwendet, wird der Compiler bzw. der *Build Process* durch die IDE aufgerufen. Dazu ruft die IDE den Compiler entweder direkt mit absolutem Pfad auf, oder die IDE verlässt sich darauf, dass der Compiler-Ordner im Suchpfad der Umgebung eingestellt ist.

Debugger

IDE Möglichkeiten:

- (empfohlen) *Code Composer Studio:* www.ti.com/tool/download/CCSTUDIO/11.2.0.00007
- *Eclipse IDE for C/C++ Developers:* www.eclipse.org/downloads/packages
- *Code::Blocks:* www.codeblocks.org Code::Blocks stellt u. a. einen Installer bereit, der zusätzlich MinGW-w64 enthält und installiert. Dieser heißt codeblocks-20.03mingw-setup.exe, wobei die Version ggf. abweicht. Alternativ kann das Archiv codeblocks-20.03mingw-nosetup.zip genutzt werden, wobei man sich aber selbst um die Einstellungen der Pfade kümmern muss.
- *Visual Studio Code:* code.visualstudio.com

[1] *Integrated Development Environment* (Integrierte Entwicklungsumgebung)

1.2 Installation von gcc und gdb auf dem eigenen Rechner (Linux)

Meistens sind gcc und gdb schon installiert und lassen sich in einem Terminal starten. Zum Testen kann man sich die Version der Programme ausgeben lassen:

```
$ gcc --version
$ gdb --version
```

Wenn der jeweilige Befehl nicht gefunden wird, im Paketmanager nach gcc bzw. gdb suchen und installieren.

1.3 Installation von gcc und gdb auf dem eigenen Rechner (Windows)

Es gibt mehrere Ports von gcc und gdb für Windows: Die Nutzung von w64devkit reicht für unsere Zwecke aus.

w64devkit (empfohlen) Eine MinGW-w64-Bereitstellung als komprimiertes Archiv. Entpacken des `w64devkit-1.21.0.zip` von github.com/skeeto/w64devkit/releases z. B. nach `c:/w64devkit`.

MinGW Ein relativ altes Projekt, die Bereitstellung kann aber noch installiert werden: Herunterladen und Ausführen der mingw-get-setup.exe von osdn.net/projects/-mingw. Es öffnet sich der MinGW-Installation-Manager. Darin unter *Basic Setup* die Pakete `mingw32-base-bin` und `mingw32-gcc-g++-bin` auswählen sowie unter *MSYS Base System* das Paket `msys-coreutils-bin` auswählen. Anschließend im Menü *Installation* die Aktion *Apply Changes* ausführen und im folgenden Dialog auf *Apply* klicken. Nach dem Einspielen der Änderungen die Meldungen kontrollieren und mit *Close* schließen.

MSYS2 Das MSYS2-Projekt stellt eine Paketverwaltung bereit, weshalb die Installation etwas komplexer ist aber deutlich mehr Optionen bietet. Zunächst der Installationsanleitung von www.msys2.org folgen, d. h. den Installer msys2-x86_64-20240113.exe ausführen. Die Version des Installers kann abweichen. Die Paketverwaltung geschieht dann über den Befehl `pacman` mit dem die Pakete `mingw-w64-x86_64-gcc` und `mingw-w64-x86_64-gdb` eingespielt werden müssen. Das geschieht in einem MSYS2-Terminal mit dem Befehl:

```
$ pacman -S mingw-w64-x86_64-gcc mingw-w64-x86_64-gdb
```

Nun stehen gcc und gdb in der MSYS2-MINGW64-Konsole zur Verfügung.

Achtung, der Paketmanager von MSYS2 erlaubt auch die Installation von Git. Dieses MSYS2-Git unterscheidet sich aber in Details von dem *Git for Windows* und eine parallele Installation sollte vermieden werden.

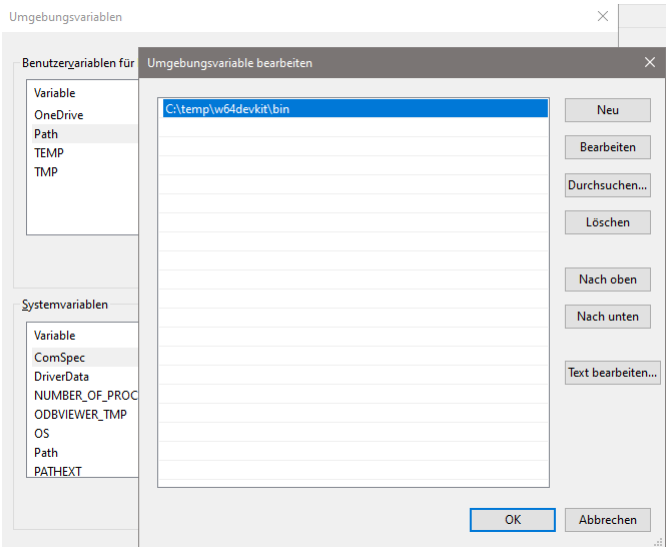


Abbildung 1: Dialog zur Einstellung des Suchpfads unter Windows. Bestehende Einträge sollten erhalten bleiben. Es muss der Ordner angegeben werden, indem sich `gcc.exe` und `gdb.exe` befinden.

1.4 Software-Pfade einrichten

Damit der Compiler und der Debugger gestartet werden können, müssen Sie in den Suchpfad aufgenommen werden. Unter Windows geschieht das unter Start, *Umgebungsvariablen für dieses Konto bearbeiten*, wo man die Benutzervariablen bearbeiten kann. Die Variable *Pfad* muss einen neuen Eintrag erhalten, bestehende Einträge nicht ändern. In den Pools sind folgende Bereitstellungen vorhanden:

GHB1005 Das w64devkit steht unter `c:/temp/w64devkit` bereit, der Suchpfad lautet `c:\temp\w64devkit\bin`.

MH125 und Selbstlernraum MinGW liegt unter `c:/mingw`, als Suchpfad muss der Eintrag `c:\mingw\bin` angegeben werden, weil sich in diesem Verzeichnis `gcc.exe` und `gdb.exe` befinden.

Abbildung 1 zeigt den Dialog mit dem Eintrag für den Pool GHB1005.

2 Arbeitsweise bei der Lösung der Aufgaben

Die geschriebenen Programme sollen mittels Git versioniert werden. Am einfachsten erstellen Sie für jede Aufgabe ein eigenes Repository. Mehrere kleine Aufgaben können sich ein Repository teilen. Dabei müssen Sie die Quelldateien und die zugehörigen Einstellungen so organisieren, dass keine Konflikte entstehen. Das großflächige Auskommentieren von Quelltext sollte lediglich zum Testen genutzt werden, nicht zur Organisation.

3 Aufgaben erster Teil

3.1 Manuelle Übersetzung ohne IDE

Aufgabe 1: Testen der Umgebung und des Compilers

Clonen Sie das Projekt gitlab.tu-ilmenau.de/FakEI/ees-lse/lehre/msp1/material-msp-1/msp_c_beispiel und übersetzen die darin befindliche Quelldatei mit dem Befehl:

```
$ gcc -std=c99 -o hallo_welt hallo_welt.c
```

Bei erfolgreichem Durchlauf erscheint keine Meldung und Sie können das erstellte Programm starten:

```
$ ./hallo_welt
Hallo Welt!
Variable i hat den Wert 5.
```

Wenn der Suchpfad falsch eingestellt ist, erscheint die Meldung

```
Der Befehl "gcc" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.
```

in der Eingabeaufforderung von Windows, oder

```
bash: gcc: command not found
```

wenn Sie in der Bash-Umgebung arbeiten. In diesem Fall gehen Sie zu Abschnitt 1.4 und kontrollieren die Einstellungen.

Das w64devkit kann ohne Konfiguration des Suchpfades genutzt werden, starten Sie dazu die `w64devkit.exe` aus dem w64devkit-Verzeichnis. Jetzt müssen Sie durch Nutzung von `cd` in das richtige Verzeichnis wechseln. Das sieht dann zum Beispiel so aus:

```
~ $ cd c:/Users/name1234/Documents/material/msp_c_beispiel
```

Allerdings: Zur Nutzung einer IDE wie z. B. *Code Composer Studio* muss der Suchpfad korrekt eingestellt sein.

Aufgabe 2: Starten eines Programms im Debugger

Bevor Sie das Programm aus der vorherigen Aufgabe im Debugger starten, sollte es erneut übersetzt werden, aber diesmal mit der Zeile:

```
$ gcc -g -std=c99 -o hallo_welt hallo_welt.c
```

Mit der Option `-g` erzeugt der Compiler zusätzliche Informationen, die dem Debugger bei der Anzeige des zugehörigen Quelltexts und der Variablen helfen. Das Programm

kann wie gewohnt gestartet werden, oder im Debugger. Der Debugger ist ein interaktives Programm und eine Session kann man an dem Prompt `(gdb)` erkennen, der Befehl `quit` oder `q` beendet die Sitzung und schließt den Debugger wieder. Lassen Sie das Programm schrittweise durchlaufen. Verwenden Sie dazu die Befehle `break`, `run` und `next` bzw. deren Abkürzungen `b`, `r` und `n` wie in dem folgenden Mitschnitt zu sehen:

```
$ gdb hallo_welt
Reading symbols from hallo_welt...
(gdb) b main
Breakpoint 1 at 0x1400013c1: file hallo_welt.c, line 21.
(gdb) r
Starting program: ...\hallo_welt.exe

Breakpoint 1, main () at hallo_welt.c:21
21          printf("Hallo Welt!\n");
(gdb) n
Hallo Welt!
27          int i=5;
(gdb) n
28          printf("Variable i hat den Wert %d.\n",i);
(gdb) n
Variable i hat den Wert 5.
29          return 0;
(gdb) n
30      }
(gdb) n
0x00007ff75a1b12c4 in __tmainCRTStartup ()
(gdb) n
Single stepping until exit from function __tmainCRTStartup,
which has no line number information.
[Inferior 1 (process 10664) exited normally]
(gdb)
```

3.2 Mehrere Quelltextdateien

Erweitern Sie das Projekt aus Aufgabe 1 um eine Source- und eine Headerdatei, beispielsweisemit den Namen `produkt.c` und `produkt.h`. Implementieren Sie in der Source-Datei z. B. die Funktion zur Berechnung des Produkts und rufen Sie sie aus der `main`-Funktion auf. Übersetzen Sie das Programm mit folgenden Schritten:

```
$ gcc -std=c99 -g -c -o produkt.o produkt.c
$ gcc -std=c99 -g -c -o hallo_welt.o hallo_welt.c
$ gcc -std=c99 -g -o hallo_welt produkt.o hallo_welt.o
```

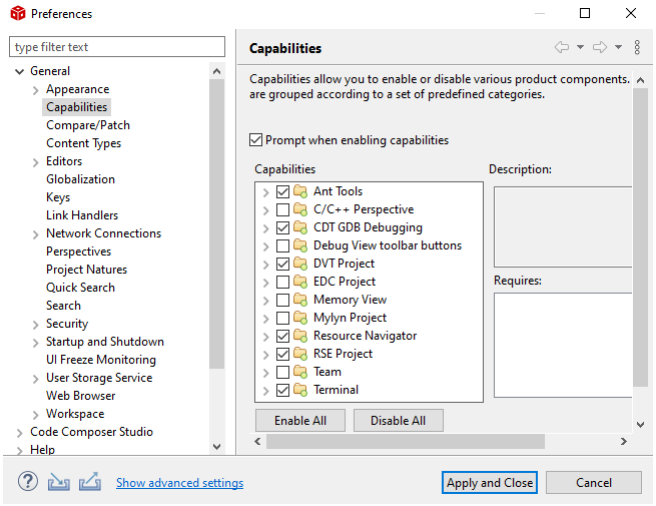


Abbildung 2: Einstellungen im Capabilities-Dialog von *Code Composer Studio*.

Recherchieren Sie die Bedeutung des Schalters `-c`, visualisieren Sie die Abhängigkeiten der Source- und Headerdateien mit einem Diagramm. Führen Sie den Präprozessor aus und erklären Sie den Aufbau der erzeugten Datei:

```
$ gcc -std=c99 -g -E -o produkt.txt produkt.c
```

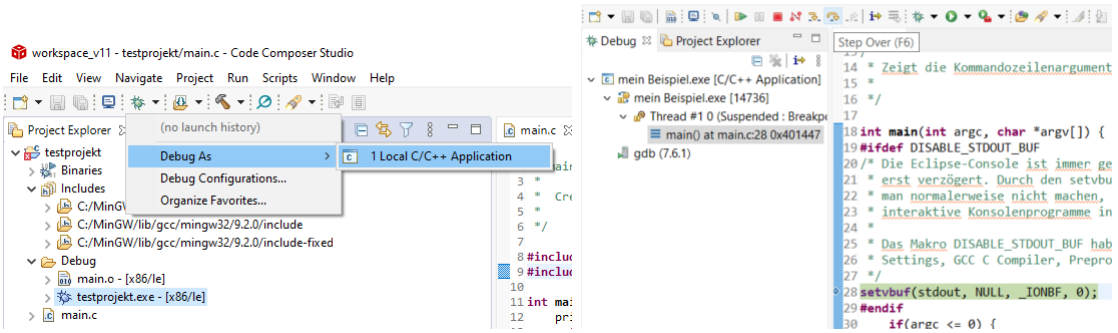
Ignorieren Sie dabei die Zeilen von `produkt.txt` die mit `#` starten.

3.3 Verwenden von *Code Composer Studio*

Aktivieren Sie unter *Window*, *Preferences*, *General*, *Capabilities* den Eintrag *CDT DGB Debugging* und den Eintrag *C/C++ Perspective*. Andere Einstellungen sollten so bleiben, wie vorgefunden. Der Dialog sieht dann z. B. wie in Abbildung 2 aus. Durch diese Einstellungen können sogenannte lokale C-Programme entwickelt und im Debugger gestartet werden, andernfalls erwartet *Code Composer Studio* einen Mikrocontroller als Zielhardware.

Aufgabe 3: Ausgangsprojekt

Clonen Sie das Projekt gitlab.tu-ilmenau.de/FakEI/ees-lse/lehre/msp1/material-msp-1/msp_1_git_eclipse. Jetzt wechseln Sie mit *Code Composer Studio* den *Workspace*, z. B. mit *File*, *Switch Workspace* auf das `workspace`-Verzeichnis im Git-Repository. Danach gehen Sie auf *File*, *Open Projects From Filesystem…* und klicken auf *Directory…*, wo Sie den `prj_1`-Ordner in dem Git-Repository suchen und auswählen. Es erscheint eine Übersicht der gefundenen Projekte und Klick auf *Finish* importiert das Projekt. Im *Project Explorer* können Sie den erfolgreichen Import erkennen. Ändern Sie am besten gleich den Projektnamen und erstellen mit Hilfe von *Git-Bash* ein Commit. Beim Umbenennen wird nur die `prj_1/.cprojekt`-Datei geändert:



(a) Starten einer Debug-Sitzung (b) Debug-Sitzung

Abbildung 3: Bildschirmansichten einer Debugger-Sitzung in *Code Composer Studio*: (a) Starten einer Sitzung des ausgewählten Programms `testprojekt.exe`, (b) nach dem Starten bleibt die Sitzung an dem Default-*breakpoint* (Haltepunkt) stehen und kann mit Einzelschritten *(Step Over (F6))* durchlaufen werden.

```
index 2983ee5..b7b39db 100644
--- a/prj_1/.project
+++ b/prj_1/.project
@@ -1,6 +1,6 @@
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
-       <name>Projekt_umbenennen</name>
+       <name>mein Beispiel</name>
<comment></comment>
<projects>
</projects>
```

Wechseln Sie in die *C/C++ Perspective* und übersetzen das Projekt mit Klick auf das Hammer-Symbol. Markieren Sie `mein Beispiel.exe` im Debug-Ordner und führen *Debug As*, *local C/C++ Application* aus. Das sieht etwa so aus, wie in Abbildung 3 dargestellt.

Aufgabe 4: Formatierte Ausgabe

Zur formatierten Ausgabe steht die `printf()`-Funktion aus der Standardbibliothek zur Verfügung, die wir im Rahmen der Übungen verwenden wollen. Für neue Projekte sollten Sie unbedingt eine sichere Bibliothek für Zeichenketten verwenden, welche die Probleme der variadischen Funktionen aus der `printf()`-Familie nicht hat.

(a) Geben Sie Beispielwerte von `double` aus:

```
1 double real_1 = 2.25;
2 double real_2 = 5/3;
```

```
3 double real_3 = (double)5/3;
4 double real_4 = 5.0/3;
```

(b) Geben Sie Beispielwerte von `int`, `unsigned int` und deren `long`-Versionen aus. Über die Formatanweisungen von `printf()` wird die Art der Ausgabe beeinflusst und der Datentyp der Argumente kommuniziert. Erzeugen Sie korrekte und fehlerhafte Ausgaben durch passende und nicht passende Formatanweisungen.

Frage: Warum kann man mit `float` und `double` keine fehlerhafte Ausgabe erzeugen?

Aufgabe 5: Eigenschaften von `sizeof`

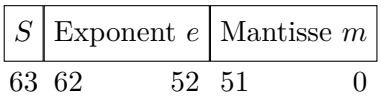
Mit `sizeof Objekt` kann zur Compilezeit die Größe von `Objekt` bestimmt werden. Mit `sizeof(Typ)` kann der Platzbedarf einer Variable des angegebenen Typs ermittelt werden. In beiden Fällen wird ein Integer des Typs `size_t` erzeugt.

(a) Zeigen Sie den Unterschied zwischen `sizeof Objekt` und `sizeof(Typ)` mit Hilfe eines Arrays (automatischer Speicher).

(b) Definieren Sie ein Präprozessor-Makro, dass die Anzahl der Elemente eines automatisch allozierten Arrays bestimmt. Nutzen Sie die Division von zwei `sizeof`-Anwendungen. Warum muss man vorsichtig sein?

Aufgabe 6: Bitweise Darstellung von Fließkommawerten

In IEEE754 werden Fließkommadarstellungen spezifiziert. Das 64-Bit-Basisformat (zur Basis 2) hat 53 binäre Stellen für die Mantisse (52 davon werden gespeichert) und neun binäre Stellen für den Exponent. Das Vorzeichen S wird im MSB[2] gespeichert. Das lässt sich wie folgt darstellen:



Für die Lösung dieser Aufgabe und für ähnliche Probleme bietet sich der Standard-Header `<stdint.h>` an. Der Standard-Header `<stdint.h>`[3] stellt Integertypen mit garantierten Größen bereit. Damit kann man portable Programme schreiben. Beispielsweise hat der in `<stdint.h>` definierte Typ `int8_t` auf jeder Hardware und mit jedem Kompiler exakt acht Bit.

(a) Definieren Sie einen Typ, der den Zugriff auf S, e und m gestattet. Verwenden Sie `union`, `struct` und Bitfelder.

(b) Schreiben Sie eine Funktion `void double_part_print(double val)`. Diese gibt das Vorzeichen, den wirksamen Exponent, d. h. $e - 1023$ als dezimale Zahl und die Mantisse m als Hexadezimale Zahl aus.

[2] *Most Significant Bit* (höchstwertiges Bit)
[3] Weiterführende Informationen beispielsweise unter en.m.wikibooks.org/wiki/C_Programming/stdint.h.

(c) Schreiben Sie eine Funktion `void double_bin_print(double val)`. Diese gibt jedes der 64 Bits von links nach rechts aus, wobei die drei Bestandteile mit `"|"` getrennt werden.

(d) Schreiben Sie eine Funktion `void double_part_print_by_mask(double val)` mit der gleichen Funktion wie `void double_part_print(double val)` aber unter Verwendung von logischen Operationen mit Bitmasken, also ohne Bitfelder und Unions.

In dieser Aufgabe ist es bequem und sinnvoll, die Masken zur Basis 2 im Quelltext zu nutzen. Der C-Kompiler von GCC unterstützt die Notation von Konstanten zur Basis 2. Die erlaubten Ziffern sind `0` und `1`, nach dem Prefix `0b`. Das ist eine der sog. GNU-Erweiterungen: gcc.gnu.org/onlinedocs/gcc/C-Extensions.html#C-Extensions.

```
1 /* The following statements are identical: */
2 i =       42;
3 i =     0x2a;
4 i =      052;
5 i = 0b101010;
```