Institut für Informatik
Daniel Neugebauer

Wintersemester 2019/20
Tag 7

Hardwarenahe Programmierung

Gruppe 6 (Stefan)

In dieser Übung nehmen Sie eine high-level-Perspektive ein und beschäftigen sich mit dem C-Kompilierprozess und dessen technischen Grundlagen wie der Repräsentation von unterschiedlichen Datentypen. Damit zusammenhängend arbeiten Sie weiter mit `make` *und dem C-Debugger* `gdb`.

Keine Angst – es sind diesmal sehr viele Aufgaben, die meisten davon sind aber sehr schnell zu lösen.

Aufgabe 1 *Datentypen und Casts*

Betrachten Sie folgende drei C-Codeschnipsel und beantworten Sie für jeden davon schriftlich: Welchen Wert erhält `x` und warum?

(a)
```
int a = 3;
int b = 2;
float x = a / b;
```

(b)
```
int a = 3;
float b = 2;
float x = a / b;
```

(c)
```
float a = 3;
int b = 2;
float x = a / b;
```

Aufgabe 2 *Der Compiler*

Beantworten Sie die folgenden Aufgaben schriftlich. Sie können bei Bedarf für Ihre Antworten online recherchieren oder sich mit Ihren Nachbarn austauschen.

(a) Aus welchen 4 Stufen besteht der C-Kompiliervorgang? Beschreiben Sie für jede der 4 Stufen kurz, was der Compiler dort macht.

(b) Sie haben bereits die Compilerdirektive `#include` kennengelernt. Auf welcher der 4 Stufen wird sie verarbeitet? Begründen Sie, warum diese Direktive nicht in den anderen Stufen sinnvoll verarbeitet werden kann.

Aufgabe 3 *Endianess*

Schreiben Sie ein Programm `endianess`, das die Art der Bytereihenfolge (Endianess) auf Ihrem Computer ermittelt.

Sie können sich bei Bedarf z.B. auf `https://de.wikipedia.org/wiki/Byte-Reihenfolge` über das Konzept von Bytereihenfolge/Endianess informieren.

Beispielaufruf:

`./endianess`

Ausgabe: "`0x1234 ist 0x34 0x12 - dieser Computer benutzt little-endian.`"

Aufgabe 4 *Datentypen I*

Wir haben Ihnen unter *longint.c* ein unvollständiges Programm bereitgestellt, das Benutzern ausgeben soll, wie viele Bytes der Datentyp `long` zur Speicherung benötigt, und was der kleinste bzw. der größte darstellbare Wert dieses Datentyps ist.

Beispielaufrufe:

`./longint`

Ausgabe: "*Der Datentyp longint benötigt TODO Bytes Speicherplatz.*"

`./longint -m`

Ausgabe: "*Der Datentyp longint benötigt TODO Bytes Speicherplatz.*
Die kleinste darstellbare Zahl beträgt dann TODO."

`./longint -M`

Ausgabe: "*Der Datentyp longint benötigt TODO Bytes Speicherplatz.*
Die größte darstellbare Zahl beträgt dann TODO."

Vervollständigen Sie das Programm. Ersetzen Sie dazu im Programm die Platzhalter "TODO" und "42" und fügen Sie fehlenden Code hinzu.

Aufgabe 5 *Datentypen II*

Wir haben Ihnen unter *typen1.c* wieder ein unvollständiges Programm bereitgestellt, das vom Benutzer eine ganze Zahl einliest und ausgeben soll, welchen Wert eine `char`, `int` oder `double`-Variable annimmt, wenn dieser Wert darin gespeichert wird. Verwandeln Sie den eingegebenen String aus `argv` zuerst in einen `long long`-Wert mithilfe der Funktion `atoll()`.

Beispielaufruf:

```
./fassungsvermoegen 5608973456345
```

Ausgabe:

```
char      : -39
int       : -253832231
double    : 5608973456345.000000
```

Ersetzen Sie dazu wieder im Programm "TODO" und "42" geeignet, und ergänzen Sie ggf. zusätzlich benötigten Code.

Hinweis: Ihr Programm wird beim Kompilieren Warnungen anzeigen, das ist diesmal OK.

Aufgabe 6 *Floss your teeth and make backups!*

Schreiben Sie ein Makefile-Rezept, das eine Sicherheitskopie aller Quell- und Headerdateien aus dem aktuellen Verzeichnis in eine zipdatei packt.

Aufgabe 7 *Make it again, Sam*

Wie kann man make dazu zwingen, beim nächsten Kompilieren alles neu zu kompilieren, und nicht alte Kompilate wiederzuverwenden?

(a) Überlegen Sie sich zunächst, wie Sie dies per Hand (also ohne Makefile) tun würden. Sie können Ihre Idee mit Ihren Nachbarn oder Ihrem Tutor besprechen, bevor Sie weitermachen.

(b) Schreiben Sie nun ein Makefile-Rezept, das diesen Vorgang automatisiert. Nennen Sie Ihr Target `clean`.

Aufgabe 8 *GDB*

In dieser Aufgabe sollen Sie den GNU-Debugger *gdb* benutzen, um in einem bereits kompilierten C-Programm (zu dem Ihnen der Quellcode zunächst nicht zur Verfügung steht) den Inhalt einer geheimen Passwort-Variable herauszufinden.

Bei Bedarf finden Sie Links zu hilfreichen Online-Quellen zu *gdb* im Lernmodul zu Tag 7.

(a) Starten Sie das von uns vorgegebene Programm *gdb_2* mit folgendem Befehl:

gdb gdb_2

Finden Sie nun mit Hilfe von *gdb* heraus, welcher Wert in der Variablen `pwd` steht!
Hinweis: Es ist Absicht, dass das Programm zunächst fehlerhaft ist und in eine Endlosschleife läuft – das sollte Sie nicht aufhalten!

(b) Nutzen Sie den Wert von `pwd` als Passwort, um die verschlüsselte Datei `gdb_2.c.gpg` mittels *gpg* zu entschlüsseln, um so an den C-Quellcode des Programms zu gelangen. Sie können dies mit folgendem Befehl tun:

```
gpg -d gdb_2.c.gpg > gdb_2.c
```

(c) Jetzt, da Sie den Quellcode des Programms haben, finden und beheben Sie (mit Hilfe von *gdb*) die beiden Fehler im Programm, welche dazu führen, dass das Programm nicht richtig terminiert.