PROGC Lab01: Erste Programme in C

Inhalt

1 Einführung

In diesem Praktikum lernen Sie die grundlegenden Handgriffe um ein eigenes einfaches Programm in C zu schreiben, zu kompilieren und zu testen.

2 Lernziele

- Sie können ein C Programm in einem Text Editor schreiben.
- Sie können den geschriebene C Programm Code in ein ausführbares Programm übersetzen und ausführen.
- Sie wissen wie die vorgegebene Test Umgebung angestossen wird und wie die Resultate zu interpretieren sind.
- Sie können ein einfaches Programm schreiben welches Integer Arithmetik anwendet.

2.1 Allgemeine Hinweise

Sie sind ermuntert in Gruppen zu arbeiten und sich auszutauschen. Beachten Sie aber bitte, dass der gesamte Praktikumsstoff ebenfalls Teil der Semester End Prüfung ist.

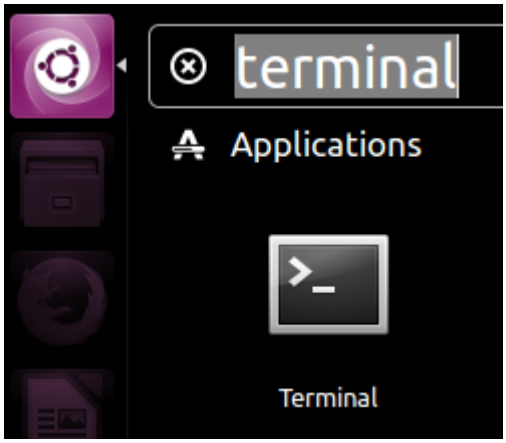
Es ist essential dass jeder sich „die Hände selber dreckig“ macht, d.h. die Praktika sollten von jedem selber geschrieben werden. Mit Fehler machen lernt man am effizientesten – eine Sprache kann man nicht nur theoretisch lernen!

3 Hintergrund Informationen

3.1 Die Bash Shell

Das Kommandozeilen Interface bildet die sogenannte Bash Shell.

Auf dem ***Ubuntu Desktop*** wählen sie dazu das ***Suche*** Icon und tippen ***terminal*** ein. Klicken Sie dann auf das erscheinende Terminal Programm Icon.



Tipp: Damit Sie dieses Programm immer direkt verfügbar haben, können sie es auch am Launcher anheften.

Im geöffneten Terminal läuft nun die Bash Shell und wartet auf Ihre Eingaben.

3.2 Nützliche Shell Kommandos

Einige nützliche Kommandos sind unten aufgelistet.

Tipp: Fast alle Kommandos haben ein **`--help`** Argument oder zumindest ein **`-h`** Argument für eine knappe Übersicht über die Möglichkeiten.

Siehe auch http://cli.learncodethehardway.org/bash_cheat_sheet.pdf

Kommando	Beschreibung	Beispiel
`man`	Hilfe anfordern. Die man-pages sind in Sektionen aufgeteilt. Die wichtigsten sind 1 (Executable programs and shell commands) und 3 (Library calls).	**`man man`** **`man 1 ls`** **`man 3 printf`**
`ls`	Listet Directory Inhalte.	**`ls -l`** **`ls -lta`**
`cd`	Ist ein in Bash eingebautes Kommando um im Directory Baum zu navigieren. **Tipp:** **`man cd`** ist erfolglos. Mit **`man bash`** kommen sie weiter, müssen aber in der riesigen man-page weit hinunter scrollen.	**`cd lab01-hello-world`** **`cd ..`**
`gcc`	Der Gnu C Compiler. Das erstellte Programm kann dann z.B. folgendermassen ausgeführt werden: **`./myprogram`**.	**`gcc -o myprogram main.c`**

Kommando	Beschreibung	Beispiel
make	Build Utility um inkrementell Programme zu erstellen. Es bestimmt die Teile welche neu gebildet werden müssen. Ein entsprechendes **Makefile** definiert die Projekt Struktur und die Abhängigkeiten unter den Files. Der Compiler wird dann durch das **make** Utility bei Bedarf mit den erforderlichen Argumenten angestossen.	**make clean** **make default** **make test** **make install** **make doc** **make -n | more** **make -p | more**
find	Sucht in einem Directory Baum nach Files. Die einfachste Anwendung ist, alle Files einfach aufzulisten (**find** Aufruf ohne Argumente). Eine andere ist, nach gewissen Files zu suchen, siehe Beispiel nebenan.	**find** **find lab01 -name '*.c'**
grep	Durchsucht den Inhalt von (Text-) Files und listet die Zeilen auf welche zum Suchmuster passen.	**grep -Hni assert tests/*.c**
less **more**	**man less**: *"[…] the opposite of more […] but has many more features […]"* Seitenweise durch Text Files navigieren.	**less main.c** **more tests.c**
cat	Darstellen des (Text-) File Inhalts auf Standard Out. Im Gegensatz zu **more**/**less** wird nicht nach jeder Seite ein User Input erwartet.	**cat */*.c**

3.3 Praktikums Projekte und deren Makefiles

Die Praktika sind in unabhängige Directories unterteilt. Z.B. **lab01-hello-world**.

Ein solches Praktikumsprojekt besteht immer aus derselben Struktur.

```
.
./doc
./tests
./tests/tests.c
./src
./src/main.c
./mainpage.dox
./bin
./Makefile
```

Der Kern bildet das **Makefile**. Damit können Sie folgende so genannte Targets bilden:

Make Targets	Beschreibung
make clean	Löscht alle generierten Files und Directories.
make (oder **make default**)	Buildet das Programm.
make test	Buildet das Test Programm und lässt es laufen.
make doc	Generiert aus den Sourcen HTML Dokumentation.

Tipp: Im Directory wo alle Praktika enthalten sind gibt es ein Makefile welches eine knappe Anleitung gibt wie man die Praktika ausführt:

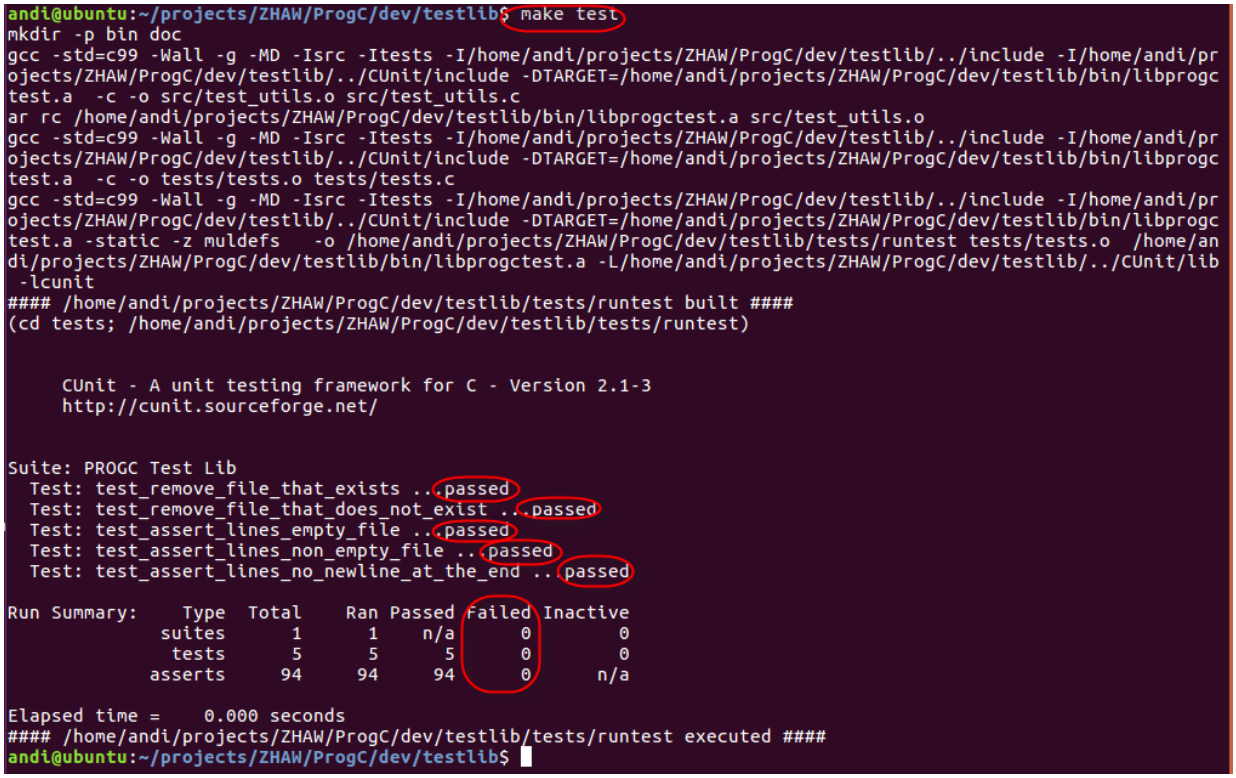
```
andi@ubuntu:~/projects/ZHAW/ProgC/dev$ make
**** PROGC Labs ****
lab01-hello-world
lab01-integer-arithmetic
lab01-point-and-line
lab01-struct
lab02-calculator
lab03-modular
lab04-array
lab05-pointer
lab06-pointer-arithmetic
lab07-make-file
lab51-control-structures
lab52-date-entry
lab53-marks
lab54-sort-words
lab55-linked-list
testlib

**** Prerequisites ****
1. Change into the testlib directory
       cd testlib
2. Build and install the library, e.g.
       make clean
       make default
       make test
       make install
       make doc
   Caution: make sure the tests, installation and documentation does not produce any error.
3. View the produced documentation, e.g.
       firefox doc/index.html

**** How to build and run a lab? ****
1. Change into the respective directory, e.g.
       cd lab01-hello-world
2. Build the lab, e.g.
       make
   The resulting executable is located in the bin folder.
3. Build and run the tests, e.g.
       make test
4. Build the HTML documentation from the sources, e.g.
       make doc
   The produced HTML documentation is located in the doc folder. Open the index.html file in a HTML browser.
Notes:
- You may cleanup the builds, e.g.
       make clean
andi@ubuntu:~/projects/ZHAW/ProgC/dev$
```

3.4 Tests

In jedem Praktikum Projekt gibt es ein Test Programm. Dieses enthält die minimalen Tests welches ein Projekt erfolgreich erfüllen muss. Bei Bedarf werden Sie zusätzliche Tests in dieses Programm integrieren.

Die Tests werden zu Beginn alle brechen. Ihre Aufgabe ist es, das Praktikumsprogramm so zu implementieren dass die Tests alle den Status „passed“ haben ohne den Test-Code oder deren Stimulus und erwarteten Resultat Daten zu manipulieren.

Die Tests werden via das **`make`** Utility gebildet und ausgeführt. Der Test Output sollte selbsterklärend sein. Bei Bedarf kann natürlich im File tests/tests.c nachgeschaut werden was genau getestet wird. Anhand der Aufgabenstellung sollte aber der Grund für das Brechen eines Tests ersichtlich sein.



3.5 Verwendete zusätzliche Sprach Elemente

Sprach Element	Beschreibung

`int main(int argc, char *argv[])`
`{`
`  ...`
`}`
	`argc`: Anzahl Einträge in **`argv`**.
	`argv`: Array von Command Line Argumenten.
	`argv[0]`: wie das Programm gestartet wurde
	`argv[1]`: erstes Argument
	…
	`argv[argc-1]`: letztes Argument

`#include <stdio.h>`
`...`
`(void)printf(...);`
	Siehe **`man 3 printf`**.
	`(void)` vor dem Funktionsaufruf **dokumentiert** dass der von der Funktion zurückgegebene Wert verworfen wird.

`#include <stdlib.h>`
`int main(...)`
`{`
`  ...`
`  return EXIT_SUCCESS;`
`  // return EXIT_FAILURE;`
`}`
	Standardisierte Rückgabe Werte für die **`main`** Funktion.
	Alternativ könnte 0 als erfolgreiche Terminierung und alle anderen Werte als Fehler Code zurückgegeben werden.

Sprach Element	Beschreibung

`if (condA) {`
`  ...`
`} else if (condB) {`
`  ...`
`} else {`
`  ...`
`}`
	Verkettetes if-else-if-else Konstrukt.
	Zu beachten ist dass das else-if zwei separate Statements sind. Dies ist äquivalent zu
	`if (condA) {`
	`  ...`
	`} else {`
	`  if (condB) {`
	`  ...`
	`  } else {`
	`  ...`
	`  }`
	`}`

`int rappen = 0;`
`int res = sscanf(argv[1]`
`  , "%d"`
`  , &rappen`
`  );`
`if (res != 1) {`
`  // Fehler Behandlung...`
`  // ...`
`}`
	Siehe **`man 3 sscanf`**.
	Die Funktion **`sscanf`** gibt die Anzahl erfolgreich erkannte Argumente zurück. Unbedingt prüfen und angemessen darauf reagieren.
	Der gelesene Wert wird in **`rappen`** gespeichert, dazu müssen Sie die Adresse der Variablen übergeben. Mehr Details dazu werden später erklärt.

`(void)printf("%02d", betrag%100);`	Siehe **`man 3 printf`**. **`%d`** gibt eine Zahl dezimal aus. **`%2d`** ist wie **`%d`**, aber min. 2 Stellen, mit führendem Space für ein-ziffrige Zahlen. **`%02d`** ist wie **`%2d`**, aber mit führender Null.

4 Aufgaben

4.1 Erweitertes Hello World

Ergänzen Sie in **`lab01-hello-world/src/main.c`** den Code so dass die Tests erfolgreich durchlaufen.

1) Schreiben Sie als erste Output Zeile **`"Program name: "`** gefolgt vom Inhalt von **`argv[0]`** und einem Newline Zeichen (**`\n`**).
2) Wenn **keine** Argumente angegeben sind, schreiben Sie danach **`"Hello World!\n"`**.
3) Sonst wenn **ein** Argument angegeben wird, schreiben Sie anstelle von **`World!`** das gegebene Argument heraus, d.h. **`"Hello `*`was-auch-immer-sie-angegeben-haben`*`\n"`**.
4) Schliesslich geben Sie den Programm Erfolgs-Status zurück: falls mehr als ein Argument gegeben wird, geben Sie **`EXIT_FAILURE`** zurück, andernfalls **`EXIT_SUCCESS`**.

4.2 Integer Arithmetik

Ergänzen Sie in **`lab01-integer-arithmetic/src/main.c`** den Code so dass die Tests erfolgreich durchlaufen.

1) Schreiben Sie ein Programm das als einziges Kommando Zeilen Argument eine Integer Zahl übernimmt welche als Rappen Wert zu interpretieren ist, z.B. das Argument **`895`** entspricht **`CHF8.95`**.

2) Dieses Argument soll in eine **`int`** Variable gespeichert werden (siehe **`sscanf`**) und eventuell auftretende Konvertierungsfehler behandelt werden. Bei Eingabefehlern soll das Programm eine sinnvolle Fehlermeldung ausgeben, und mittels **`return 255;`** terminieren.

3) Die eingegebenen Rappen sollen auf die Geld-Münzen CHF5.00, CHF2.00, CHF1.00, CHF0.50, CHF0.20, CHF0.10, CHF0.05 aufgeteilt werden so dass immer die grösstmöglichen Münzen verwendet werden. Der Output für obigen Rappen Betrag soll folgendermassen aussehen:

```
andi@ubuntu:~/projects/ZHAW/ProgC/dev/lab01-integer-arithmetic/bin$ ./main 895
CHF 8.95:
- 1 x 5.00
- 1 x 2.00
- 1 x 1.00
- 1 x 0.50
- 2 x 0.20
- 0 x 0.10
- 1 x 0.05
Kein Rest
andi@ubuntu:~/projects/ZHAW/ProgC/dev/lab01-integer-arithmetic/bin$
```

Und im Fall eines Restes:

```
andi@ubuntu:~/projects/ZHAW/ProgC/dev/lab01-integer-arithmetic/bin$ ./main 897
CHF 8.97:
- 1 x 5.00
- 1 x 2.00
- 1 x 1.00
- 1 x 0.50
- 2 x 0.20
- 0 x 0.10
- 1 x 0.05
Rest = 0.02
andi@ubuntu:~/projects/ZHAW/ProgC/dev/lab01-integer-arithmetic/bin$
```

4) Das Main Programm soll den Rest zurückgeben (d.h. wenn kein Rest: 0).
Der Return Wert des letzten Kommandos kann mit folgendem Kommando in der Bash Shell abgefragt werden.
(Achtung: das **`echo`** Kommando hat auch einen Return Wert ;-))

```
echo $?
```