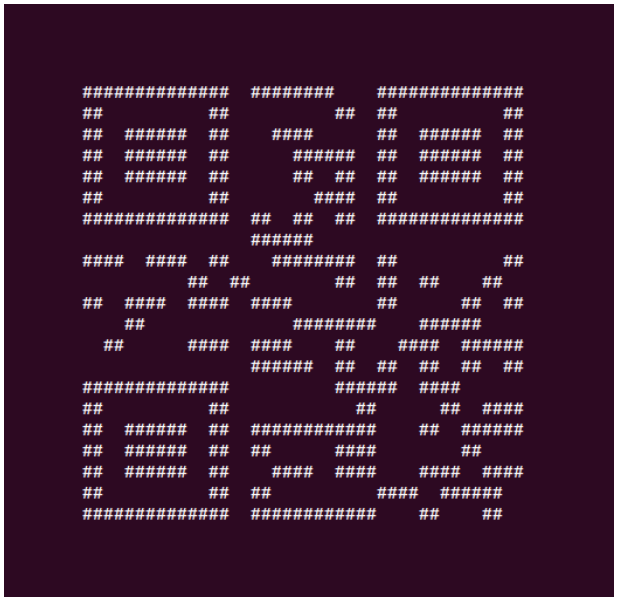
SNP Praktikum: Erste Schritte mit C Programmieren



1 Übersicht

In der ersten Aufgabe lernen Sie ein einfaches Programm zu schreiben und zu testen. Damit machen Sie sich mit der Arbeitsumgebung und einfachen C Konstrukten vertraut.

In der zweiten Aufgabe verfestigen und erweitern Sie obiges Wissen und obige Fähigkeiten.

2 Lernziele

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

- 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 Input von der Kommandozeile entgegennimmt, verarbeitet und auf Standard Output formatiert ausgibt.
- Sie können ein einfaches Programm schreiben welches Standard Input liest, verarbeitet und auf Standard Output formatiert ausgibt.
- Sie können ein eigenes Programm in einem Shell Script via Pipe einbinden

Die Bewertung dieses Praktikums ist am Ende angegeben.

Die Code Beispiele liegen im **`git`** Repository **`snp-lab-code`**.

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 essentiell, 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 Getting Started

3.1 Praktikums Projekte und deren Makefiles

Die Praktika sind in unabhängige Directorys unterteilt. Z.B. **`P02_QR_Code`**

Ein solches Praktikumsprojekt besteht immer aus derselben Struktur.

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

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

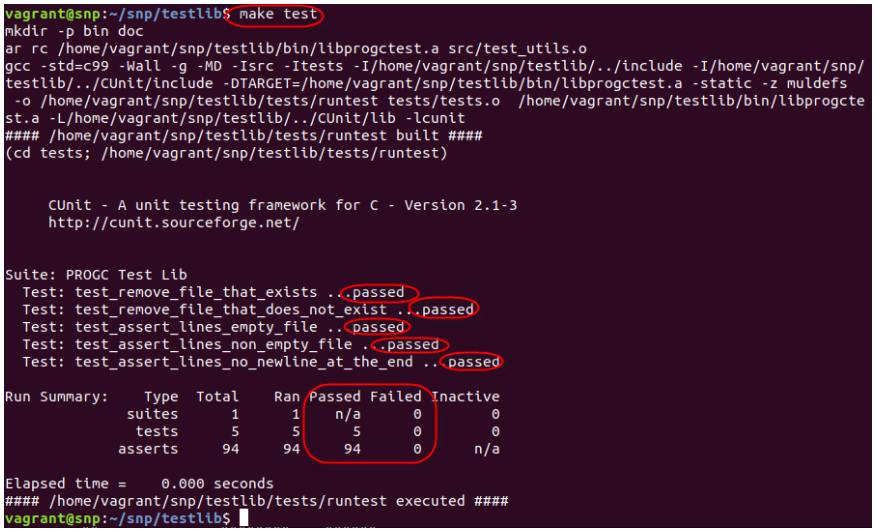
Make Targets	Beschreibung
`make clean`	Löscht alle generierten Files und Directorys.
`make` (oder **`make default`**)	Bildet das Programm.
`make test`	Bildet das Test Programm und lässt es laufen.
`make doc`	Generiert aus den Sourcen HTML Dokumentation.

3.2 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 fehlschlagen. 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.



4 Aufgabe 1: Bit-Manipulationen

4.1 Teilaufgabe Arithmetik

Schreiben Sie ein Programm in C, das als Argument eine unsigned Integer-Zahl im Bereich von 0 bis 255 akzeptiert. Speichern Sie die Zahl in einer 8-bit Variablen. Auf dieser Zahl soll Ihr Programm einige Operationen ausführen.

Prüfen Sie in Ihrem Programm zuerst, ob der Eingabewert gültig ist. Geben Sie andernfalls eine Fehlermeldung mit einer kurzen Anleitung Ihres Programms aus. Geben Sie in diesem Fall **`EXIT_FAILURE`** zurück.

Das Programm soll den Input einmal als unsigned Zahl und einmal als signed Zahl ausgeben. Weiter gibt das Programm das Resultat einer Addition des Input-Werts und 255 aus (unsigned). Mit dem wrap-around erreichen Sie damit die Subtraktion um eins. Bilden Sie in Ihrem Programm auch das Einer- und das Zweierkomplement und geben Sie jeweils das Resultat als unsigned-Wert aus.

Lassen Sie Ihr Programm auch gleich die hexadezimale Darstellung ausgeben.

Eine Beispiel-Ausgabe:

```
unsigned:   13 (0x0d)
  signed:   13 (0x0d)
    +255:   12 (0x0c)
   one's:  -14 (0xfffffff2)
   two's:  -13 (0xfffffff3)
```

Hinweise

- Im Praktikumsrahmen gibt es ein vorgegebenes **`main.c`** File welches erweitert werden soll. Das **`Makefile`** und die Test Files existieren auch schon.

- Benutzen Sie für jeden Ausgabewert eine eigene 8-bit Variable. Wenn Sie die Ausgabewerte direkt im **`printf`**-Statement berechnen, werden Sie mit hoher Wahrscheinlichkeit Ausgaben von 32-bit Werten sehen.
- Damit Ihr Programm den Unit-Test besteht, muss ihr Output aufs Zeichen genau der Vorgabe entsprechen. Verwenden Sie deshalb für die dezimale Darstellung eine Breite von 4 Zeichen.

4.1 Teilaufgabe Bit-Manipulationen

Erweitern Sie ihr Programm so, dass es als optionales zweites Argument eine Integer-Zahl im Bereich von 0 bis 7 akzeptiert. Ihr Programm soll auch das zweite Argument prüfen. Wenn es nicht im erlaubten Bereich liegt, soll Ihr Programm eine Fehlermeldung und den Rückgabewert **`EXIT_FAILURE`** zurückgeben.

Wenn der Anwender Ihres Programms ein zweites Argument mitgibt, soll Ihr Programm nicht den Output von der ersten Teilaufgabe machen, sondern den Output wie unten beschreiben.

Das zweite Argument gibt an, auf welchem Bit des ersten Arguments die folgenden Manipulationen durchgeführt werden sollen:

- Prüfen, ob das Bit gesetzt ist oder nicht.
- Löschen des Bits, ohne die restlichen Bits zu verändern.
- Setzen des Bits, ohne die restlichen Bits zu verändern.
- Invertieren des Bits, ohne die restlichen Bits zu verändern.

Ihr Programm soll die Ergebnisse jeweils in dezimaler, hexadezimaler und oktaler (führende Null) Darstellung ausgeben. Das soll beispielsweise für den Input **`131 2`** so aussehen:

```
                dec   hex   oct
your input    : 131   0x83  0203
bit 2 is not set.
bit 2 cleared: 131   0x83  0203
bit 2 set    : 135   0x87  0207
bit 2 flipped: 135   0x87  0207
```

Auch hier gilt: der Test prüft die Ausgabe pedantisch genau, d.h. jedes abweichende Zeichen in der Ausgabe wird als Fehler interpretiert.

Hinweise

- Im Praktikumsrahmen gibt es ein vorgegebenes **`main.c`** File welches erweitert werden soll. Das **`Makefile`** und die Test Files existieren auch schon.
- Da Sie in C keine einzelnen Bits ansteuern können, müssen Sie einen Weg finden, die notwendigen Masken dem zweiten Argument entsprechend dynamisch zu erzeugen. D.h. wie kommt man vom zweiten Argument für einen Wert 5 zur Maske 0x20 (das Bit Nummer 5 hat den Wert 1, die übrigen Bits den Wert 0).
- **Tipp:** eine Möglichkeit ist, den Shift-Left Operator zu verwenden.

5 Aufgabe 2: QR-Code auf dem Terminal ausgeben

5.1 Teilaufgabe QR Code mittels Schwarzer/Weisser Hintergrundfarbe

Schreiben Sie ein Programm in C, welches von Standard Input ein Text File zeichenweise liest: wo ein Space Zeichen erkannt wurde soll ein Space mit weisser Hintergrundfarbe ausgeben werden, ansonsten ein Space mit Schwarzer Hintergrundfarbe.

Die Steuerung der Farben geschieht via ANSI Terminal Control Codes. Damit wird die Ausgabe auf einem Terminal modifiziert (wie z.B. Hintergrund Farbe, etc.).

Für Interessierte: siehe **`man console_codes`** und suchen Sie darin nach **`ECMA-48 SGR`**.

Ausschnitt aus **`man console_codes`**:

```
$ man console_codes
...
The ECMA-48 SGR sequence ESC [ parameters m sets display attributes.
...
param  result
0      reset all attributes to their defaults
...
40     set black background
...
47     set white background
...
```

Z.B. **`"\033[40m "`** hat folgende Bedeutung

- **`\033[`** Beginn der Kontrollsequenz (ASCII code für **`ESC`**, gefolgt von **`[`**)
- **`40`** Kontrollsequenz Parameter (40 = Schwarzer Hintergrund)
- **`m`** Ende der Kontrollsequenz
- Space normales Zeichen (hier ein Space) ausgeben

Programm Funktionalität

Der Input soll folgendermassen verarbeitet werden:

1. Den String **`"\033[0m\n"`** auf Standard Output ausgeben. Dies entspricht der ASCII Code Sequenz **`27`**, **`91`**, **`48`**, **`109`**, **`10`**. Damit werden die Console Attribute zurückgesetzt und ein New-Line Zeichen ausgegeben.
2. Aus dem Input für jedes gelesene Zeichen folgenden Output generieren
    a. Wenn das gelesene Zeichen New-Line (**`'\n'`**) ist, derselben Output wie bei 1.
    b. Sonst, wenn das gelesene Zeichen ein Space (**`' '`**) ist, **`"\033[47m "`** ausgeben, d.h. ein Space in Weiss. Dies entspricht der ASCII Code Sequenz **`27`**, **`91`**, **`52`**, **`55`**, **`109`**, **`32`**.
    c. Sonst, ein Space in Schwarz ausgeben (**`"\033[40m "`** bzw. die ASCII Code Sequenz **`27`**, **`91`**, **`52`**, **`48`**, **`109`**, **`32`**).
3. Zum Abschluss wieder denselben Output ausgeben wie bei 1.

Das Programm soll immer **`EXIT_SUCCESS`** als Exit Code zurückgeben.

Prüfen Sie die Funktionalität mit **`make test`**. Ihr Programm muss exakt den von den Tests erwarteten Output generieren.

Was ist mit dem QR Code aus **`tests/snp.input`** gegeben (z.B. scannen Sie den Output mit Ihrem Mobile Phone)?

Hinweise

- Im Praktikumsrahmen gibt es ein vorgegebenes **`main.c`** File welches erweitert werden soll. Das **`Makefile`** und die Test Files existieren auch schon.
- Falls Sie die einzelnen Bytes (ASCII Codes) Ihres Outputs genauer anschauen wollen, können Sie folgende Pipe aufrufen: **`bin/term-qr-code | od -a`**

5.2 Teilaufgabe Bash Script

Schreiben Sie ein Bash Script welches via das **`qrencode`** Paket das aktuelle Datum und die Zeit in einen QR Code verpackt und mittels des Programms aus der ersten Teilaufgabe auf dem Terminal ausgibt. Bitte Prüfen Sie die Ausgabe indem Sie den QR Code z.B. mit Ihrem Mobile Phone scannen.

Hinweise

- Um das **`qrencode`** Paket zu installieren müssen Sie zu Beginn einmal folgendes auf einer Bash Shell ausführen (dabei werden Sie nach Ihrem Passwort gefragt):

  `sudo apt install qrencode`
- Wenn das Paket installiert ist, können Sie das Programm **`qrencode`** verwenden, um Text vom Standard Input auf Standard Output als QR-Code in Form von ASCII-Art zu generieren, welcher dann als Input für Ihr obiges Programm dienen kann (siehe auch die Bilder zuoberst in diesem Dokument):

  `qrencode -t ASCII -o -`
- Das aktuelle Datum mit Zeit wird mittels **`date`** Kommando abgefragt.

6 Bewertung

Die gegebenenfalls gestellten Theorieaufgaben und der funktionierende Programmcode müssen der Praktikumsbetreuung gezeigt werden. Die Lösungen müssen mündlich erklärt werden.

Aufgabe	Kriterium	Gewicht
1	Sie können das funktionierende Programm inklusive funktionierende Tests demonstrieren und erklären.	
	Teilaufgabe Arithmetik	1/4
	Teilaufgabe Bit Manipulation	1/4
2	Sie können das funktionierende Programm inklusive funktionierende Tests demonstrieren und erklären.	
	Teilaufgabe QR Code mittels Schwarzer/Weisser Hintergrundfarbe	1/4
	Teilaufgabe Bash Script	1/4

7 Anhang

7.1 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 P02_QR_Code`** **`cd ..`**
`gcc`	Der Gnu C Compiler. Das erstellte Programm kann dann z.B. folgendermassen ausgeführt werden: **`./myprogram`**.	**`gcc -o myprogram main.c`**
`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 P02 -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`**

7.2 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

Sprach Element	Beschreibung

`#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.

`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 zahl = 0;`
`int res = sscanf(argv[1]`
`  , "%d"`
`  , &zahl`
`  );`
`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 **`zahl`** 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.