Aufgabenblatt 6
Systemprogrammierung (SoSe 2019)

Institut: Beuth Hochschule für Technik Berlin
Dozent: Prof. Dr. Christian Forler
Url: `https://lms.beuth-hochschule.de/`
Email: `cforler(at)beuth-hochschule.de`

- Lösen Sie die Aufgaben mit einem der folgenden Editoren: **vim** und **emacs**.
- Sämtliche Aufgaben müssen sich mittels einem **Makefile** kompilieren lassen.
- Zu verwendende Compilerflags: `-W -Wall -Wextra -Werror -std=c11`

Aufgabe 1 (4 Punkte) Hello World

a) Legen Sie die drei Dateien `main.c, hello.c` und `bye.c` an.

`hello.c` und `bye.c` enthalten jeweils eine Funktion die einen Namen als char-Zeiger übergeben bekommt und `"Hello, <name>!"` bzw, `"Bye, <name>!"` ausgeben.

Die main-Funktion in `main.c` ruft beiden Funktionen nacheinander auf.

b) Schreiben Sie für `hello.c` und `bye.c` zwei separate Header Files. Achte Sie darauf, Mehrfachinklusion zu vermeiden. Inkludieren Sie diese Header Files.

c) Schreiben Sie ein Makefile welches das Programm baut.

Aufgabe 2 (6 Punkte) Fehlersuche
In den folgenden Makro-Definitionen sind Fehler versteckt, die bei der Benutzung der Makros zu unerwarteten Ergebnissen führen. Geben Sie je ein Beispiel für die Auswirkungen der Fehler an und korrigieren Sie, wenn möglich, die Definition.

a) `#define 2_PI 3.14 + 3.14`

b) `#define MULT(a, b) (a * b)`

c) `#define ADD(a, b) (a) + (b)`

Hinweis: Aufgabenteil b) und c) ist nicht ganz einfach.

Aufgabe 3 (4 Punkte) Makros
Implementieren Sie die folgendes Makros.

a) `SWAP(x,y)`
Das Makro vertauscht `x` mit `y`.

b) `LSB(x)`
Das Makro liefert das LSB (Least Significant Bit) von `x` zurück.

Aufgabe 4 (8 Punkte) Awesome Library
Nehmen Sie an Sie arbeiten gerade an einem Projekt. Dieses Projekt verwendet die Library `awesome` welche in ihrem Header neben einigen Funktionen auch das Makro `__awesome_ver`, das die Version der Library enthält.

```
#pragma once
#include<stdio.h>

#define __awesome_ver 3

static void do_something_awesome(int a) {
  printf("awesome: %d\n", a);
}

static void hello_awesome(char *name) {
  printf("Hello %s, You are awesome.\n", name);
}
```

Den aktuelle Stand des Quellcodes Ihres Projektes sehen Sie hier:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "awesome.h"

int main() {
  puts("starting");
  hello_awesome (getlogin())
  something_awesome(42);
  puts("done");

  return EXIT_SUCCESS;
}
```

a) In der gerade neu erschienen Version 3 der awesome-Library wurde der Name einiger Funktionen geändert. `something_awesome(int a)` wurde in `do_something_awesome(int a)` umbenannt.

Es sollen künftig sowohl die neue, als auch alte Versionen der Library unterstützt werden. Passen Sie den Code so an, dass er mit beiden Versionen der Library kompiliert.

Legen Sie dazu die beiden Unterverzeichnisse `awesome2` und `awesome3` mit den entsprechenden Headerdateien (`awesome.h`) an. Die Datei `awesome3/awesome.h` stellt die Library in Version 3 (siehe oben). Die Datei `awesome2/awesome.h` stellt die Library in Version 2 dar und muss entsprechend angepasst werden.

Schreiben Sie ein Makefile mit dem sich das Projekt mit beiden Versionen der Library bauen lässt.

Hinweise

- *Sie können zwei Targets verwenden (z.B.* `version2` *und* `version3`*).*
- *Mit dem Compilerflag* $-I$ *können Sie einen Pfdad angeben, indem der Compiler nach Headerdatein schaut (z.B. gcc -Iawesome2 -o main main.c).*

b) In ihrem Projekt soll in Zukunft die `hello_awesome()` Methode verwendet werden, das erst ab Version 2 zur Verfügung steht.

Ergänzen Sie den Code so, dass das Übersetzen mit einer aussagekräftigen Fehlermeldung abgebrochen wird, sollte die Library in einer Version kleiner als 2 vorliegen.

c) Immer wieder beschweren sich Benutzer, dass das Programm zu viel Debug-Text (`starting` und `done`) auf die Kommandozeile ausgebe.

Sorgen Sie dafür, dass dies nur noch geschieht, wenn beim Kompilieren das Makro `DEBUG` definiert war

d) Finden Sie heraus, wie sie dem GCC-Compiler Makros als Kommandozeilenoptionen übergeben können und compilieren Sie ihr Programm einmal mit und einmal ohne Debug-Ausgaben.

Aufgabe 5 (8 Punkte) Multi Platform Support
Schreiben Sie ein Programm welches über die Kommandozeilenparameter drei Zahlen einliest und deren Summe berechnet und diese auf dem Bildschirm ausgibt. Ihr Programm soll auf den drei fiktiven Betriebssystemen *Birne X*, *Locked-BSE* und *Doors 10* lauffähig sein. *Birne X* bietet jedoch nur den Datentyp `long` an, um Ganzzahlen zu speichern. `Locked-BSE` hingegen kennt nur den Datentyp `int`, und `Doors 10` kann Ganzzahlen nur in Variablen vom Typ `short` speichern.

Verwenden Sie Präprozessor-Anweisungen in der Art, dass das Programm mit einer entsprechenden `#define`-Anweisung jeweils so übersetzt wird, dass es unter dem jeweiligen Betriebssystem funktioniert. Falls das Programm unter dem nicht unterstützten Betriebssystem `Banana Mac` kompiliert wird, soll eine Fehlermeldung ausgegeben und die Kompilierung abgebrochen werden.