



MIKROPROZESSORPRAKTIKUM

WS2015

**Termin1**

C-Programmierung für eingebettete Systeme

Name, Vorname	Matrikelnummer	Anmerkungen
Datum	Raster (z.B. Mi3x)	Testat/Datum

Legende: V:Vorbereitung, D: Durchführung, P: Protokoll/Dokumentation, T: Testat

**Lernziele:**

Mit den folgenden Versuchen sollen Sie die Sprache "C" einmal aus einer anderen Sicht kennen lernen. An ganz einfachen Programmen sollen Sie ermitteln, welchen Code ein Compiler erzeugt, wo welche Variablen abgelegt werden, wie ein *call by value / reference* in ARM Assembler umgesetzt wird.

**Arbeitsverzeichnis:**

Kopieren Sie sich das Verzeichnis /mpsWS2015. Dort stehen im Unterverzeichnis Termin 1 die Dateien *Termin1Aufgabe1.c* als Programmgerüstbeispiel und *makefile* zur Verfügung.

**Aufgabe 1:**

Es gibt folgende elementare Datentypen. Füllen Sie die Tabellen mit den fehlenden Informationen.

Datentyp	Bit	kleinste Zahl Hex / Dez	größte Zahl Hex / Dez	Bemerkungen
char	8	0x80 / -128	0x7F / 127	
short int	16			
int	{16} 32 {64}			hängt vom verwendeten Compiler und Betriebssystem ab
unsigned int	16			
long int	32 {64}			hängt vom verwendeten Compiler und Betriebssystem ab

Zusätzlich gibt es noch die Gleitpunktzahldatentypen

Datentyp	Bit	kleinste darstellbare Zahl	größte darstellbare Zahl
float	32		
double	64	ca. 1,7E-308	ca. 1,7E308
long double	80	ca. 1,2E-4932	ca. 1,2E4932

Legen Sie im C Programm Variablen mit den in der Tabelle gezeigten Datentypen an. Weisen Sie den Variablen Werte zu und betrachten Sie sich die Ergebnisse mit dem Debugger im Speicher.

**Aufgabe 2:**

Geben Sie die folgenden Zahlen in der jeweils anderen Schreibweise des entsprechenden Datentypen an.

Dez	int	float
1	= _____	= _____
-1	= _____	= _____
65535	= <u>0x0000FFFF</u>	= _____
1,024*10 <sup>3</sup>	= _____	= _____
	= _____	= <u>0x44FFFFFF</u>

**Aufgabe 3:**

Legen Sie im C Programm zwei lokale integer Variablen an. Weisen Sie diesen im Code Werte zu, z.B. 0x1 und 0x2. Übersetzen Sie Ihr Programm und schauen Sie sich den erzeugten Code an. Führen Sie das Programm im Simulator aus. Dokumentieren Sie den erzeugten Code.

## Aufgabe 4:

Verwenden Sie nun statt der lokalen Variablen globale Variablen. Was ändert sich? Warum?

## Aufgabe 5:

Legen Sie nun zwei globale und zwei lokale integer Variablen an. Weisen Sie den Variablen Werte zu. Machen Sie nun Zuweisungen der Form "global=lokal". Was stellen Sie fest? Ändern Sie im Makefile die Optimierung und beobachten Sie die Auswirkungen.

```
// Termin1Aufgabe1.c
//*****
// Lösung zu Termin1
// Aufgabe 1
// Namen: _____
// Matr.:  _____
// vom:    _____
int main (void)
{
// ....
    return (0);
}
```

## Aufgabe 6:

Vereinbaren Sie einen Funktionsprototypen für eine Funktion "addition", die 3 integer Parameter erwartet. Rufen Sie diese Funktion im Anschluss an die Zuweisung nach Aufgabe 3 mit den beiden lokalen und einer globalen Variablen auf. Dokumentieren Sie Ihre Beobachtungen. Ändern Sie im Makefile die Optimierung und beobachten und dokumentieren Sie die Auswirkungen.

## Aufgabe 7:

Ändern Sie den Funktionsprototypen so ab, dass er statt der

lokalen Variablen Pointer auf diese Variablen erwartet und rufen Sie die Funktion entsprechend auf. Was ändert sich und warum? Wo liegen die lokalen Variablen nun?

## Aufgabe 8:

Erstellen Sie zu diesem Termin ein Protokoll mit den Lösungen zu den Aufgaben und Ihren Erkenntnissen. Das Protokoll sollen Sie zum nächsten Termin vorlegen können.

```
# makefile
#*****
# Quellendatei
FILE = Termin1Aufgabe1
# Compiler
GCC = arm-eab63-elf-gcc
# Optimierungsstufe
OPTI = 1
```

```
all:
# Erzeugen des Assemblercode aus der C-Datei
$(GCC) -S -O$(OPTI) $(FILE).c
# Übersetzen und binden der Quelldatei
$(GCC) -g -O$(OPTI) $(FILE).s -o $(FILE).elf
```