# 1. Übung zu TEI2_CPR

Nur für die Aufgaben auf diesem Blatt: Bitte Linux starten. Wenn Sie Ihren eigenen Rechner verwenden wollen und Windows als Betriebssystem haben, brauchen Sie cygwin oder MinGW.
**Abgabe:** zip-Datei mit **Quellcode**-Dateien zu Aufgabe **3, 4 und 6**.

## 1. Hello, World!

Schreiben Sie ein Programm, das „Hello, World!“ ausgibt.
Kompilieren Sie es auf die „harte Tour“ mit gcc:

- Erzeugen Sie zuerst mit „gcc -c …“ nur die Objektdatei und
- linken Sie sie erst im zweiten Schritt mit „gcc –o …“, siehe Foliensatz „TEI2_CPR_01_Einführung.pdf“, Folie 25 (Seite 14 im pdf!)
- Zum Ausführen eines Programms im aktuellen Verzeichnis muss bei den meisten Linux-Konfigurationen der Pfad mit angegeben werden, also z.B.: `./Hello` (der Punkt steht für das aktuelle Verzeichnis). Warum? (Dies könnte eine Klausurfrage sein.)

## 2. Beispielprogramm zu printf und Variablen-Initialisierung

Auf der Moodle-Seite gibt es die Quellcode-Datei `bsp.c`.

a) Kompilieren Sie und starten Sie das Programm.
b) Vollziehen Sie die ersten vier printf-Ausgaben des Hauptprogramms nach.
c) Beachten Sie die Ausgaben der Funktionen `globalfunction` und `globalfunction2`: Bei exakt denselben Funktionsaufrufen kommt es zu verschiedenen Ausgaben! Warum? Können Sie auch die ausgegebenen Werte erklären? (Dies könnte eine Klausurfrage sein.)
d) Probieren Sie statt „-c“ die Option „-S“. Es entsteht eine Datei mit Endung „.s“. Was enthält sie? Versuchen Sie die Zeilen mit der Initialisierung der Variablen `i,j,k` in dieser Datei wiederzufinden.

## 3. Nutzung von Funktionen aus anderen Dateien/Bibliotheken

Um eine Funktion aus einer anderen Datei oder einer Bibliothek benutzen zu können muss:

- die Schnittstelle (Signatur) der Funktion bekannt sein (normalerweise geschieht das durch Einbinden einer Header-Datei mit `#include`)
- das Object-File oder die Bibliothek mit der Implementierung „gelinkt“ werden.

Die Datei `func.c` enthält eine Funktion zur Berechnung der dritten Wurzel aus einer Zahl.
Erweitern Sie die Datei `bsp.c` um die Berechnung der dritten Wurzel aus 3.375 unter Nutzung der Funktion aus `func.c` und lassen Sie das Ergebnis mit `printf` ausgeben.
Schreiben Sie dazu eine Header-Datei `func.h` und benutzen Sie diese – *nicht* `func.c` selbst – in `bsp.c`. Linken Sie `bsp.o` und `func.o` sowie (falls nötig) die Mathematik-Bibliothek `libm` (siehe Folie 39 in „TEI2_CPR_01_Einführung.pdf“).
Für die Abgabe: Kommandozeilen zum Kompilieren/Linken als Kommentar oben in `bsp.c` einfügen.

## 4. Präprozessor: Macros

Erweitern Sie die Datei `bsp.c` wie folgt:

a) Schreiben Sie ein Macro PYTHAGORAS(a,b), das die Länge der Hypotenuse eines rechtwinkligen Dreiecks gemäß der folgenden Formel berechnet: $\sqrt{a^2 + b^2}$
b) Lassen Sie PYTHAGORAS(3,4) und PYTHAGORAS(1+3,2+1) berechnen und ausgeben.
c) Sind die Ergebnisse wie erwartet, nämlich jeweils 5? Wenn nicht, was ist zu korrigieren?
d) Probieren Sie statt „-c“ die Kommandozeilenoption „-E“ von „gcc“. Was passiert?

## 5. Optional: Präprozessor: Bedingte Kompilierung

Benutzen Sie einen `#ifdef` / `#endif` Block um (zur Kompilierzeit) mit Hilfe der Definition einer Makro-Konstanten `ZEIGEVARS` auswählen zu können, ob die ersten vier Ausgaben des Hauptprogramms angezeigt werden oder nicht. Mit Hilfe der zusätzlichen Option „`-D`“ von gcc kann von der Kommando-Zeile die Makro-Konstante definiert werden (ohne Leerzeichen zwischen `-D` und dem Namen der Konstanten, also `-DZEIGEVARS`). Probieren Sie auch nochmal „gcc -E".

## 6. Endianness

Schreiben Sie ein C-Programm, das Folgendes durchführt:

- Setzen Sie eine int-Variable auf den hexadezimalen Wert 0xbaadf00d.
- Definieren Sie eine `union`, um auf die einzelnen Bytes der Integerzahl zugreifen zu können.
- Lassen Sie sich die einzelnen Bytes als hexadezimale Zahlen ausgeben.

Ist Ihr System ein Big- oder ein Little-Endian System?

## 7. Optional: Rundungsfehler

a) Setzen Sie eine `float`-Variable `v` auf `8.9f` (ohne das `f` wird die Zahl als `double` interpretiert und Sie erhalten evtl. eine Warnung).
Lassen Sie `v, v*100` und `v*100-890` ausgeben.
Sind die Ergebnisse so wie erwartet?

b) Definieren Sie eine `union` über ein `float f` und ein `int i`.

c) Legen Sie eine Variablen `u1` der `union` an und setzen Sie das niedrigste Bit auf `1` (alle anderen Bits sollen 0 sein).

d) Lassen Sie sich `u1.f` mit `%e` und mit `%.48f` (und zur Kontrolle `u1.i` als `int`) ausgeben.

e) Setzen Sie das `float` einer zweiten Variable `u2` der `union` auf `u2.f=100.f`.

f) Lassen Sie sich `u2.i` als hexadezimalen Wert ausgeben.

g) Addieren Sie zu `u2.f` den Wert von `u1.f` hinzu: `u2.f+=u1.f;` (also die beiden `float`-Werte addieren).

h) Lassen Sie sich `u2.i` wieder als hexadezimalen Wert ausgeben.

i) Hat sich der Wert gegenüber der ersten Ausgabe geändert? Haben Sie eine Idee, warum nicht? (Tipp: Erinnern Sie sich an das, was Sie im ersten Semester über Gleitkommazahlen gelernt haben.)

j) Um wieviel ändert sich `u2.f`, wenn Sie zu `u2.i` den Wert `1` hinzuaddieren? Lassen Sie sich von `u2.f` mindestens 10 Nachkommastellen anzeigen.

k) Berechnen Sie `float diff=u2.f-100.f;` und lassen Sie sich `diff` ausgeben. Vergleichen Sie den Wert mit dem von `u1.f`.