Hinweis: Die Aufgaben nehmen einige Zeit in Anspruch. Fangen Sie schon in der ersten Woche damit an (auch wenn Sie zwei Wochen Zeit haben)! Relevante Folien: „02_slides_hello_world.pdf“.

→ Alle Aufgaben (Aufgabe 1, 2 und 3) sind Pflichtaufgaben.

Aufgabe 1 „Das erste Programm“ (Pflichtaufgabe, abzugeben: Quellcode+CMakeLists.txt):

Sie müssen die Abgaben der Übung in Gruppen von 3-5 Leuten durchführen. Idealerweise sind es 4 Leute pro Gruppe.

Zweck der ersten Aufgabe ist es:

- Die Gruppenmitglieder mir mitzuteilen.
- Den Aufbau eines einfachen C++ Projektes mit mehreres *.cpp Dateien zu definieren.
- Mit CMake und dem C++ Compiler vertraut zu werden.

Anforderungen:

- Erstellen Sie ein C++ Projekt mit CMake.
- Das Projekt besteht aus
  - einer main.cpp Datei und
  - einer *.cpp-Datei pro Gruppenmitglied (z.B. pierre_bayerl.cpp).
- Das Programm soll die Gruppenmitglieder unmissverständlich ausgeben (siehe Beispiel Aufgabe_Gruppe.tgz)

Geben Sie den Quellcode zusammen mit der CMakeLists.txt-Datei ab (nicht das Kompilat abgeben, auch nicht die temporären Dateien). Die Abgabe erfolgt über Moodle.

Bei Problemen: Rückfragen über bayerl@mail.hs-ulm.de oder in der Vorlesung.

Aufgabe 2 „Klausurfragen“ (Pflichtaufgabe; abzugeben: (1) kurze Antworten und (2) die Foliennummer aus den Vorlesungen für die Folie, die das Thema am besten beschreibt):

Aufgabe 2.1: Das Projekt in 02_01.zip erzeugt einen Fehler beim Erstellen des Executables („**[…] multiple definition of `log(char const*)' […]**“). Stellen Sie dies nach und erklären Sie, was hier falsch gemacht wurde (bzw. welche Regel verletzt wurde).

_____ Folie: _____

Aufgabe 2.2: Welche Zeilen erzeugen eine Warnung bzgl. „Narrowing“? Folie: _____

```
int main() {
  float a=1.1;                  // Warnung: [ ]-ja   [ ]-nein
  double b=3.2;                 // Warnung: [ ]-ja   [ ]-nein
  a=b;                          // Warnung: [ ]-ja   [ ]-nein
  b=a;                          // Warnung: [ ]-ja   [ ]-nein
  a={b};                        // Warnung: [ ]-ja   [ ]-nein
  b={a};                        // Warnung: [ ]-ja   [ ]-nein
}
```

Aufgabe 2.3: Was ist ein „dangling pointer“? Folie: _____

_____

_____

Aufgabe 2.4: Was gibt folgendes Programm aus? Antwort: _____ zwei Folien: _____

```
#include <iostream>
void f(int *p) { (*p)++; }
void f(int &p) { p-=10; }
int main() {
  int x=0; f(&x); f(x); f(&x);
  std::cout << x << "\n";
}
```

Aufgabe 2.5: Welche Zeilen erzeugen einen Compilerfehler? Folie: _____

```
#include <iostream>
int main() {
  int a;
  std::cin >> a; // read an int
  const int b=a;                  // Fehler: [ ]-ja   [ ]-nein
  a++;                            // Fehler: [ ]-ja   [ ]-nein
  b++;                            // Fehler: [ ]-ja   [ ]-nein
  constexpr int c = a;            // Fehler: [ ]-ja   [ ]-nein
}
```

Aufgabe 2.6: Welche Zeilen erzeugen einen Compilerfehler? Folie: _____

```
#include <iostream>
int main() {
  int x=0;
  const int *p1 = &x;
  int* const p2 = &x;
  const int* const p3 = &x;

  *p1 = 1;                        // Fehler [ ]-ja [ ]-nein
  *p2 = 2;                        // Fehler [ ]-ja [ ]-nein
  *p3 = 3;                        // Fehler [ ]-ja [ ]-nein

  p1 = nullptr;                   // Fehler [ ]-ja [ ]-nein
  p2 = nullptr;                   // Fehler [ ]-ja [ ]-nein
  p3 = nullptr;                   // Fehler [ ]-ja [ ]-nein
}
```

Aufgabe 3: „Klausur-Programmieraufgabe“ (Pflichtaufgabe; abzugeben: Quellcode)

Zweck der Aufgabe ist es:

- Verwenden von „function overloading“
- Programmierpraxis (Hilfe z.B. über https://en.cppreference.com/w/cpp)

Anforderungen:

Ermöglichen Sie mit einem Funktionsaufruf „analyze“ sowohl Textsequenzen (**std::string** und/oder **const char***), als auch ganze Zahlen (**int**) zu analysieren (vgl. Code). Dabei soll die Anzahl an Buchstaben/Zahlen zurückgegeben werden, die man benötigt um

a) den Text darzustellen (z.B. **strlen** oder **std::string::size**), bzw.
b) die Zahl dezimal anzuzeigen (z.B. wie oft kann man die Zahl durch 10 teilen; Vorzeichen nicht vergessen!).

Beispiele sind als „asserts“ gegeben.

```
#include <iostream>
#include <string>
#include <cassert>

// insert code here or create analyze.h/cpp
// (possible with <20 LOC)
#include "analyze.h"

int main(int, char**) {
  // count number of digits (incl. Sign) or chars
  assert( analyze(0) == 1); // one digit
  assert( analyze(9) == 1); // one digit
  assert( analyze(-1) == 2); // one digit+1 char
  assert( analyze(1234) == 4); // 4 digits
  assert( analyze(-90) == 3); // 2 digits+1 char
  assert( analyze("Hello World") == 11); // 11 chars
  assert( analyze("ABC") == 3); // 3 chars
}
```