Allgemeine Hinweise

- Die Aufgaben zur Systemprogrammierung können sowohl in C++ als auch C implementiert werden.
- Es werden keine Vorgaben zur Verwendung bestimmter IDEs gemacht. Gleichwohl finden im Praktikum auch Demonstrationen statt, die diese verwenden.
- Die Lösungen der Praktikumsaufgaben müssen auf den Pool-Rechnern oder der virtualisierten Umgebung von der Praktikumsgruppe vorgeführt werden können.
- Die folgenden Regeln dienen Erzeugung von möglichst homogenem und lesbarem Code, der in der Kleingruppe von beiden Teilnehmern erstellt wird.

Regeln für die Software-Entwicklung

1. Jedes Programm sollte einen Kopf mit Programmnamen und Kurzbeschreibung (1-2 Zeilen, Autor, Datum und einer Liste der ToDo's (noch zu erledigender Änderungen, Fehlerabfragen, zu entfernender Warnungen etc.) enthalten. Beispiel:

```
/************************************************************
 * HelloWorld.c
 * Funktion: Gibt den String HelloWorld auf der Standard-
 *           Ausgabe aus.
 * Autor: M. Muster, HSOS
 * Historie:
 *   2020/09/02: main() Routine implementiert
 *   2020/09/04: Fertigstellung
 * To Do:
 * - Formatierung der Ausgabe einbauen.
 * -
 ************************************************************/
```

   Im gesamten Source-Code (auch in den Kommentaren) gibt es keine Umlaute (ä,ö.ü) und kein scharfes sz.

2. Bei der Benennung von C++-Klassen oder C-Datentypen (Structs, Unions, …) sollten Sie immer die sog. Camel-Case-Schreibweise verwenden, z.B. **`FileSystem`**, **`ProcessManager`**,… Bei Funktionen (z.B. **`openFile`**) und Methoden sollten kurze und *‚sprechende' Namen* aufweisen (in C++ in der Regel wg. des Klassenkontexts kürzer **`open`**). Globale Variablen oder Datenobjekte sollten wenn möglich vermieden, aber falls doch notwendig per Großschreibung (Bsp. **`MASTERFILE`**) herausgestellt werden.

3. Wichtige Variablen werden auf der Zeile der Deklaration kommentiert. Zentrale Variablen werden nur zu Anfang eines Blocks deklariert und können dort initialisiert werden. Laufindizes und Hilfsvariablen können auch im Block deklariert werden.

4. Zwischen sinnvollen Abschnitten des Programms werden Kommentare zur Trennung eingefügt. Beispiel: Eingabe, Berechnung, Ausgabe

5. Die Funktion **main** liefert einen Rückgabewert vom Typ **int**.

    a. Bei normaler Beendigung wird das Programm mit **return 0** und

    b. bei Fehlern mit **return -1** verlassen.

    c. Diese Codes werden auch beim Beenden des Programms mit **exit()** verwendet. Sind neben den üblichen Aufräumarbeiten von **exit** noch weitere Aktionen (Schreiben von Logs etc.) notwendig, so sollten Funktionshandler mit **atexit()** registriert werden.

6. Jedes Programm beginnt beim Aufruf mit der Ausgabe einer Zeile mit Startinformation über das Programm, z. B. „Sortierung von Zahlen“, und erfragt erst dann mögliche Eingaben. Ggf. wird auf fehlende Parameter beim Programmaufruf aufmerksam gemacht und die Befehlssyntax ausgegeben.

7. Programmabschnitte wie **main**, das Innere von Schleifen, if/case-switch- Abschnitten usw. werden um vier/acht/zwölf... Zeichen eingerückt. Diese Einstellungen können auch in einem IDE vorgenommen werden, sind in der Regel aber schon voreingestellt. Tabulatoren sollten bei Verwendung von normalen Texteditoren zur Programmierung vermieden werden, da sie zwischen verschiedenen Editoren uneinheitlich sind.
   Beispiel:
   ```
   int main() {
       /* Hier beginnt das eigentliche Programm ... */
       int i, j, umfang, flaeche;
   ```

8. Alle if-, for- und do-Abschnitte mit mehr als einer Anweisung werden durch geschweifte Klammern abgegrenzt.
   Beispiel:
   ```
   if (n == 0) {
      printf("n ist Null");
      return -1;
   }
   ```
   Folgt nur eine Anweisung, so kann diese in der gleichen Zeile bleiben.
   Beispiel:
   ```
   if (n == 0) printf("n ist Null");
   ```

9. Die öffnende Klammer kann am Ende der Zeile mit der zugehörigen Anweisung (z.B. **main**, **while**, **if**) oder unter dem ersten Buchstaben der Anweisung stehen. Die schließende Klammer steht genau unter dem ersten Buchstaben der zugehörigen Anweisung.

10. **switch**-Anweisungen enthalten neben den relevanten **case**-Abschnitten immer einen **default**-Abschnitt, der z. B. die Fehlerbehandlung enthält.

11. Der Typ einer Variablen wird nicht im Namen kodiert.

12. Nicht **void** Funktionen / Methoden sollten per zentralem **return …;** am Ende verlassen werden.

13. Überprüfen Sie bei Systemaufrufen immer, ob diese erfolgreich sind.

a. Insbesondere Fehlschläge von **`fork()`** oder **`exec…()`** können zu Problemen führen.

b. Fehler bei Dateioperationen aufgrund

   - einer Datei, die von einem anderen Programm geöffnet ist,
   - eines falsch geschriebenen Dateinamens,
   - des falschen Verzeichnisses

   sind sehr häufig. Solche Fehlerarten müssen überprüft werden.

c. Überprüfen Sie immer, ob dynamische Speicherzuweisungen erfolgreich sind. Fehler aufgrund von knappem Speicher sind später sehr schwer zu finden, da sie nach dem Neustart des Programms meist nicht reproduzierbar sind. In C++ sollten Exceptions abgefangen werden.

14. Beim Kompilieren des Programms darf es bei normalen Compiler-Einstellungen keine Warnungen geben. Idealerweise stellen Sie aber ein, dass alle Warnings (bei **`gcc`** werden per Option **`-Wall`** alle potentiellen Warnings angezeigt, normale C-Programme können mit dem strengeren **`g++`** übersetzt werden) ausgegeben werden.