Einschub: Häufige Fehler bei der HÜ 1

1) Funktionsdeklarationen mit leerer Parameterliste

   `int f();` bezeichnet eine Funktion mit beliebigen Argumenten! Wenn man eine Funktion ohne Argumente erklären will, muß man in C ... (void) schreiben, also im Bsp.: `int f(void);` beim Prototyp und `int f(void) { ... }` bei der Definition.

2) Datendeklarationen vs. Datendefinitionen in Header-Files

   Angenommen, die globale Variable `int globalVar` soll von mehreren Modulen aus zugreifbar sein. Kommt dann eine Definition `int globalVar;` in einen Header-File? Nein!! Da der Headerfile in mehrere Source-Files inkludiert wird (sonst braucht man ihn nicht!), würde der Speicherplatz für globalVar mehrfach angelegt → Binder meldet Fehler! In den Header-File gehört nur die Deklaration

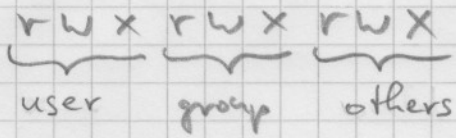
   ```
   extern int globalVar;
   ```

   diese legt keinen Speicherplatz an. Zusätzlich muß in genau einem Source-File dann natürlich auch die Definition `int globalVar;` stehen.

3) Permission-Bits bei Dateien und Directories

   Wir haben für jede Datei (und jedes Directory)

die Permission-Bits (= Rechtigungen) (E2)
r (read), w (write) und x (execute), und zwar jeweils für den Besitzer, die Gruppe und alle anderen:



Bitte richtig setzen! (Achtung Windows-Benutzer!)
Typisch ist: rwxr-xr-x für Executables/Scripts
rw-r--r-- für andere Dateien

Bem.: Das x-Bit bedeutet bei Directories: auf die darin enthaltenen Dateien/Subdirectories darf zugegriffen werden.

4) Wenn man beim Aufruf Ihres Programms andere Optionen angeben muß als bei der Referenzimplement., dann wird Ihr Programm die Tests nicht bestehen!
⇒ Behandeln Sie Optionen genauso wie die Referenzimplementierung!

5) Wenn die "write" - Instruktion mehr ausgibt, als sie soll, dann wird Ihr Programm die Tests nicht bestehen!

6) Auch bei der Modulo-Operation ist Division im Spiel: 15 % 0 ist undefiniert und führt zu einem Absturz des Programms ⇒ überprüfen!!

7) Die vorzeichenrichtige Aufweitung von 24-Bit-Werten auf 32-Bit-Werte ist nur bei Immediate-Werten in Instruktionen angebracht, jedoch niemals bei Werten, die schon auf dem Stack stehen!

⇒ Geben Sie nichts zusätzliches aus, weder bei "write" noch sonstwo!

8) Ganz generell gilt: Sie müssen mehr und systematischer testen! Nicht nur die Testfälle prüfen, von denen Sie glauben, daß Sie funktionieren – ganz im Gegenteil! Loten Sie die Grenzen Ihrer Implementierung aus; "quälen" Sie Ihren Code! Wenn Sie dabei auf Fälle stoßen die in der Aufgabenstellung nicht oder unsauber spezifiziert sind: Rücksprache mit dem "Auftraggeber" (mir)! Niemals glauben "das geht schon gut"; es geht praktisch immer schief! (E3)

9) Machen Sie Ihr Entwicklungsdirectory dicht! Wenn jemand Ihren Code klaut und der Plagiatstest schlägt an, fliegen Sie raus! Ich kann im Normalfall nicht feststellen, wer bei wem kopiert hat!

10) Der Test eines Ninja-Binärfiles auf den korrekten Format-Identifizierer "NJBF" kann nicht mit der Funktion strcmp() durchgeführt werden, denn ohne besondere Maßnahmen bilden die vier eingelesenen Zeichen keinen korrekten String (es fehlt das abschließende '\0')! Prüfen Sie die Zeichen einzeln oder verwenden Sie die Funktion strncmp()!