Aufgabe 1: (14 Punkte)

Bei den Einfachauswahlfragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagrechten Strichen durch (☒) und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Was versteht man unter Polling? 2 Punkte

- ☐ Wenn ein Programm regelmäßig eine Peripherie-Schnittstelle abfragt, ob Daten oder Zustandsänderungen vorliegen.
- ☐ Wenn ein Programm zum Zugriff auf kritische Daten Interrupts sperrt.
- ☐ Wenn ein Gerät so lange Interrupts auslöst, bis die Daten durch den Mikrocontroller abgeholt wurden.
- ☐ Wenn ein Gerät durch Auslösen eines Interrupts Daten von einem Mikrocontroller anfordert.

b) Welche Aussage zu Zeigern ist richtig? 2 Punkte

- ☐ Zeiger können zur Manipulation von schreibgeschützten Daten verwendet werden.
- ☐ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-value.
- ☐ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-reference
- ☐ Zeiger vom Typ `void*` benötigen weniger Speicher als andere Zeiger, da bei anderen Zeigertypen zusätzlich die Größe gespeichert werden muss.

c) Welchen Wert hat die Variable `i` nach Ausführung der folgenden Zeilen Code: 2 Punkte

```
uint8_t i = 0x21;
i <<= 4;
```

- ☐ 10
- ☐ 16
- ☐ 0x25
- ☐ 0x84

d) Gegeben sei folgender Programmcode. Welche der folgenden Aussagen ist richtig? 2 Punkte

```
#define INC(x) x++;
int a = 0;
INC(INC(a));
```

- ☐ Die Variable `a` enthält nach der Ausführung den Wert `1`.
- ☐ Das Programm stürzt zur Laufzeit ab.
- ☐ Der C-Compiler meldet beim Übersetzen einen Fehler.
- ☐ Die Variable `a` enthält nach der Ausführung den Wert `2`.

e) Welche Aussage zu globalen Variablen ist richtig?

2 Punkte

- ☐ Mit Hilfe von globalen Variablen kann man Nebenläufigkeitsprobleme vermeiden.
- ☐ Man sollte globale Variablen sparsam einsetzen, da sie bei gleichen Datentypen zu Übersetzungsfehlern führen.
- ☐ Durch den Einsatz von globalen Variablen werden vor allem große Programme unübersichtlich und auf Dauer schwer wartbar, da der direkte Bezug zwischen Daten und den Funktionen verloren geht.
- ☐ Globale Variablen sind die einzige Möglichkeit, Daten über Modulgrenzen hinweg auszutauschen.

f) Welche Aussage zu folgender Funktion ist richtig?

2 Punkte

```
int *foo(void) {
    static int bar = 0;
    bar++;
    return &bar;
}
```

- ☐ Die Funktion liefert einen Zeiger auf die lokale Variable `bar` zurück. Dies ist in C nicht zulässig und führt zu einem Übersetzungsfehler.
- ☐ Die Variable `bar` enthält beim Verlassen der `foo()`-Funktion immer den Wert `1`, da `bar` bei jedem Aufruf von `foo()` mit `0` initialisiert wird.
- ☐ Die Variable `bar` ist über die Laufzeit der `foo()`-Funktion hinaus gültig. Daher kann der zurückgelieferte Zeiger sicher vom Aufrufer verwendet werden.
- ☐ Beim Verlassen der Funktion `foo()` wird die `automatic`-Variable `bar` vom Stack entfernt und der Zeiger verliert seine Gültigkeit. Ein Zugriff durch den Aufrufer führt zu zufälligen Ergebnissen.

g) In welchen Situationen wird ein Prozess in den Zustand *blockiert* versetzt?

2 Punkte

- ☐ Während der Prozesserzeugung solange die Verwaltungsstrukturen noch nicht angelegt sind.
- ☐ Ein Kindprozess des Prozesses terminiert.
- ☐ Wenn er ausgeführt werden könnte, aber ein anderer Prozess die CPU zugeteilt bekommen soll.
- ☐ Beim Lesen eines Zeichens von der Tastatur, so lange keine Taste gedrückt wird.

Aufgabe 2: Höhenverstellbarer Tisch (30 Punkte)

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Implementieren Sie die Steuerung eines höhenverstellbaren Tisches, der den Rücken von Mitarbeitenden schonen soll. Die Steuerung selbst bietet hierbei eine sehr einfach gehaltene Benutzerschnittstelle: Die Bewegungsrichtung (*hoch* bzw. *runter*) kann mittels eines Drehschalter (*mittig im Bild*) ausgewählt werden. Solange der Taster (*rechts im Bild*) anschließend gedrückt gehalten wird, bewegt sich der Tisch in die zuvor ausgewählte Richtung. Des Weiteren verfügt die Steuerung über einen Timer, der an einen regelmäßigen Wechsel der Sitz/Steh-Position erinnern soll. Falls innerhalb von 30 Minuten der Taster nicht getätigt wurde, soll eine Warn-LED (*links im Bild*) im Zeitintervallen von etwa einer Sekunde blinken. Sobald der Taster danach wieder gedrückt wurde, soll die LED wieder ausgeschaltet werden.

Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion `void init(void)`. Treffen Sie hierbei keine Annahmen über den initialen Zustand der Hardware-Register.
- Der Eingang `PD2` (Interrupt 0) ist mit dem Taster verbunden. Eine fallende Flanke tritt genau dann auf, wenn der Taster gedrückt wird und eine steigende Flanke dann, wenn dieser wieder losgelassen wird. Sie dürfen davon ausgehen, dass der Taster initial nicht gedrückt gehalten wird.
- Für die Zeittaktung soll ein 8-Bit Timer verwendet werden. Konfigurieren Sie diesen so, dass er alle Millisekunde einen Interrupt auslöst.
- Innerhalb der Unterbrechungsbehandlung soll nun fortwährend das Verstreichen von `1s` (siehe Konstante `TIMER_INTVAL`) bzw. `30min` (siehe Konstante `WAKE_INTVAL`) aufgezeichnet werden. Die entsprechenden Ereignisse sollen in Form von zwei Events an das Hauptprogramm weitergegeben werden.
- Falls der Taster gedrückt wurde, soll die Bewegungsrichtung mittels eines Aufrufs von `int16_t sb_adc_read(ADCDEV)` für das ADC-Gerät `POTI` bestimmt werden. Diese Funktion liefert eine vorzeichenlose 10-Bit-Ganzzahl als Rückgabewert. Während des Aufrufs müssen die Interrupts gesperrt sein. Bei `0` als Rückgabewert, soll die bereits vorgegebene Funktion `void move_down(void)`, für `1023` entsprechend `void move_up(void)` aufgerufen werden. Alle übrigen Werte dürfen ignoriert werden.
- Beim Lösen des Tasters soll abschließend die ebenfalls vorgegebene Hilfsfunktion `void move_stop(void)` aufgerufen werden.
- Sowohl beim Drücken als auch beim Lösen des Tasters soll der interne Zählerstand der Zeit (für `1s` bzw. `30min`) zurückgesetzt werden.
- Sind seit dem letzten Tastendruck `30min` vergangen, soll nun mit jedem weiteren Verstreichen von `1s` der Pegel der Warn-LED an dem Pin `PB0` invertiert werden.
- Benutzen Sie **keine** Gleitkommazahlen oder sonstige Mathematikbibliotheksfunktionen.
- Stellen Sie sicher, dass sich der Mikrocontroller möglichst oft im Schlafmodus befindet.

Information über die Hardware

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Taster: Interruptleitung an **PORTD**, Pin 2

- Fallende Flanke: Taster wird gedrückt
- Steigende Flanke: Taster wird losgelassen
- Pin als Eingang konfigurieren: Entsprechendes Bit im **DDRD**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren: Entsprechendes Bit im **PORTD**-Register auf 1
- Externe Interruptquelle **INT0**, ISR-Vektor-Makro: **INT0_vect**
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **INT0**-Bits im Register **EIMSK**

Konfiguration der externen Interruptquelle **INT0** (Bits im Register **EICRA**)

Interrupt 0		Beschreibung
ISC01	ISC00	
0	0	Interrupt bei low Pegel
0	1	Interrupt bei beliebiger Flanke
1	0	Interrupt bei fallender Flanke
1	1	Interrupt bei steigender Flanke

Warn-LED: Ausgang an **PORTB**, Pin 0

- Erinnert, dass die Sitz/Steh-Position gewechselt werden soll.
- Pin als Ausgang konfigurieren: Entsprechendes Bit im **DDRB**-Register auf 1
- Kontrollleuchte initial ausschalten, entsprechendes Bit im **PORTB**-Register auf 1

Zeitgeber (8-bit): **TIMER0**

- Es soll die Überlaufunterbrechung verwendet werden (ISR-Vektor-Makro: **TIMER0_OVF_vect**)
- Der ressourcenschonendste Vorteiler (*prescaler*) ist 64, wodurch es bei dem 16 MHz CPU-Takt (hinreichend genau) alle $1ms$ zum Überlauf des 8-bit-Zählers **TCNT0** kommt.
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **TOIE0**-Bits im Register **TIMSK0**

Konfiguration der Frequenz des Zeitgebers **TIMER0** (Bits im Register **TCCR0B**)

CS02	CS01	CS00	Beschreibung
0	0	0	Timer aus
0	0	1	CPU-Takt
0	1	0	CPU-Takt / 8
0	1	1	CPU-Takt / 64
1	0	0	CPU-Takt / 256
1	0	1	CPU-Takt / 1024
1	1	0	Ext. Takt (fallende Flanke)
1	1	1	Ext. Takt (steigende Flanke)

Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdint.h>
#include <adc.h>
#include "desk.h"

static const uint32_t TIMER_INTVAL = 1000lu;
static const uint32_t WAKE_INTVAL = (1000lu * 60 * 30);

// Funktionsdeklarationen, globale Variablen, etc.
// Ende Funktionsdeklarationen, globale Variablen, etc.
// Unterbrechungsbehandlungsfunktionen
// Ende Unterbrechungsbehandlungsfunktionen
```

D:

```
// Funktion main

  // Initialisierung und lokale Variablen

  // Hauptschleife
```

```
    // Button Event verarbeiten
```

```
    // Timer Event verarbeiten
```

```
// Ende main
```

M:

```
// Initialisierungsfunktion
```

```
// Ende Initialisierungsfunktion
```

I:

Aufgabe 3: shift (19 Punkte)

Die Cäsar-Chiffre ist ein einfaches, schon in der römischen Antike verwendetes Verschlüsselungsverfahren. Es beruht auf der Verschiebung aller verwendeten Buchstaben einer Nachricht um einen konstanten Wert.

Beispiel: `BSP` verschoben um 3 ergibt `EVS`

Entschlüsselt wird die Nachricht durch Addition des Restwerts (modulo 26).

Beispiel: `EVS` verschoben um $26 - 3 = 23$ ergibt `BSP`

Gerechnet wird immer modulo der Alphabetlänge: Wird das Alphabet an einem Ende verlassen, wird am anderen Ende wieder begonnen.

Beispiele: `z + 1 = a`, `Z + 1 = A`.

Schreiben Sie ein Programm `shift`, das eine Nachricht aus einer Datei ausliest und um einen gegebenen **positiven** Wert verschiebt.

Inhalt der Datei msg.txt ausgeben:

```
$> cat msg.txt
SPiC!
```

`shift` starten um Inhalt von msg.txt zu verschieben:

```
$> ./shift 13 msg.txt
CZsM!
```

Das Programm soll im Detail wie folgt funktionieren:

- Das Programm prüft zu Beginn, ob genau zwei Parameter übergeben wurden. Sollte dies nicht der Fall sein, gibt es eine entsprechende Fehlermeldung aus und beendet sich.
- Wurde das Programm korrekt aufgerufen, wird der erste Parameter als ganzzahliger (positiver) Verschiebewert eingelesen (`parse_pos_int()`).
- Anschließend wird der als zweite Parameter übergebene Dateipfad geöffnet, die Datei zeichenweise eingelesen, Buchstaben verschoben (`shift()`) und anschließend, ggf. verschoben, ausgegeben.
- Nach erfolgreicher Ausführung endet das Programm mit dem Statuscode `EXIT_SUCCESS`.
- Bei Fehlern soll eine Fehlermeldung ausgegeben werden und das Programm mit einem Fehlerstatus (`EXIT_FAILURE`) beendet werden.
- Achten Sie darauf, allokierte Ressourcen wieder freizugeben (`fclose()`, ...).
- Implementieren Sie zudem die Hilfsfunktion `int shift(int c, int count)`, welche das Zeichen `c` um den Wert `count` verschiebt, falls es ein Buchstabe ist:
  - Mit Buchstaben kann in C wie mit Ganzzahlen gerechnet werden.
  - Im verwendeten ASCII-Zeichensatz decken Großbuchstaben den Bereich von 65 (`== 'A'`) bis 90 (`== 'Z'`) ab, Kleinbuchstaben 97 (`== 'a'`) bis 122 (`== 'z'`).
  - Für korrektes Überlaufverhalten muss der tatsächliche Shiftwert relativ zur passenden Basis (`'a'` bzw. `'A'` für Klein- bzw. Großbuchstaben) bestimmt werden.
  - Beachten Sie die Arithmetik modulo 26, sodass bspw. `'Z' + 1` korrekt auf `'A'` abgebildet wird.

Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.

```
#include <stdio.h>
#include <stdlib.h>

static void die(const char *s) {
  perror(s);
  exit(EXIT_FAILURE);
}

static void usage(void) {
  fputs("usage:␣decode␣<shiftcount>␣<filename>\n", stderr);
  exit(EXIT_FAILURE);
}

// converts a string representation into the corresponding number
// expects numbers in base 10, no other base is supported
// on success, 0 is returned and *res set to the parsed number
// otherwise, -1 is returned, the errno is set and the content
// of *res is undefined
extern int parse_pos_int(const char *str, int *res);

// Funktion shift



  // 3 Fälle: Klein-/Großbuchstabe, kein Buchstabe
```

```
  // Verschiebung des Buchstabens
```

B:

```
// Funktion main
  // Argumente parsen
```

```
// Zeichen einlesen und verschieben
```

M:

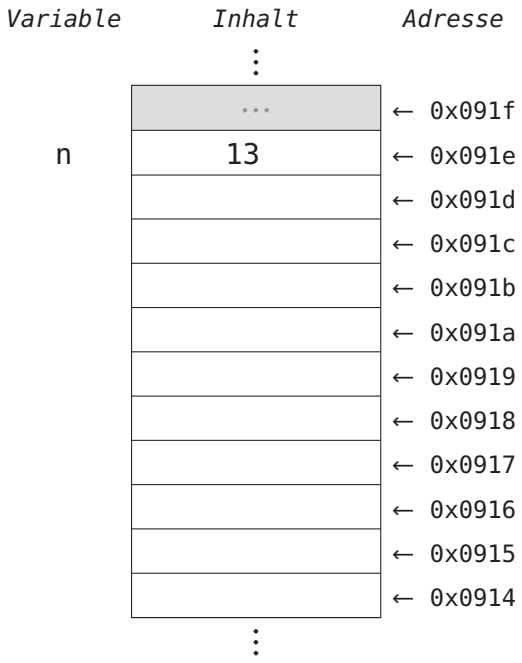
Aufgabe 4: Zeiger und Felder (8 Punkte)

Das folgende Programm wird ohne Optimierungen übersetzt und auf einem 8-Bit AVR/ATmega32 Mikrocontroller ausgeführt.

Hinweis: Lesen Sie zuerst die Aufgabenstellung – ein vollständiges Verständnis des Programms ist zur Bearbeitung der Aufgabe nicht notwendig.

```
1  #include <stdint.h>
2
3  static void str_set(char *str, uint8_t len, char c){
4      for (uint8_t i = 0; i < len; i++) {
5          str[i] = c;
6      }
7  }
8
9  void main(void) {
10   // Teilaufgabe a
11   uint8_t n = 0;
12   char c = '=';
13   char s[] = "x;y";
14   const char *p = &s[0];
15   while (*p != '\0') {
16     if (*p != ';') {
17       n++;
18     }
19     p++;
20   }
21   // Ende Teilaufgabe a
22
23   // Teilaufgabe c
24   str_set(s, 16, c);
25   // Ende Teilaufgabe c
26 }
```

Dazugehöriger Stack (Auszug):



a) Obige Grafik zeigt einen Speicherauszug des Stacks nach Erreichen der Zeile `21` an. Ändern / erweitern Sie die Grafik um einen möglichen Aufbau des Stacks (Variable und Inhalt) nach der Ausführung der Zeilen `10` bis `21`. (4 Punkte)

b) Erklären Sie den Unterschied zwischen den Typen `const char * und char * const`. (2 Punkte)

c) Die Funktion `str_set` soll entsprechend der angegebenen Länge `len` eine Zeichenkette mit dem Zeichen `c` überschreiben. Welche zwei Probleme können durch die Angabe einer ungültigen Länge auftreten? (2 Punkte)

Aufgabe 5: Nebenläufigkeit (10 Punkte)

Sie dürfen diese Seite zur besseren Übersicht heraustrennen!

Das nachfolgende Codebeispiel für einen 8-Bit-AVR-Mikrocontroller überprüft, ob der Wert der modul-globalen Variable `packets` über einem bestimmten Schwellwert liegt. Wenn dies der Fall ist, wird darauf reagiert, ansonsten wird mit der Hauptschleife fortgefahren. Der Wert von `packets` wird asynchron durch die Unterbrechungsbehandlung von `INT0` erhöht.

Die Implementierung dieser Funktionalität beinhaltet ein Nebenläufigkeitsproblem.

```
#include <avr/interrupt.h>

#define THRESHOLD 0xf00d
static volatile uint16_t packets = 0;

ISR(INT0_vect) {
  packets += 8;
}

extern void handle(uint16_t local_packets);

void main(void) {
  /* ... */
  sei();
  while (1){
    uint16_t local_packets = packets;
    if(local_packets > THRESHOLD) {
      // handle case
      handle(local_packets);
    }

    // do something else ...
  }
}
```

Die beiden folgenden Assemblerausschnitte zeigen Abschnitte der `main()`-Funktion und der Unterbrechungsbehandlungsfunktion für `INT0`. In den Assemblerausschnitten können Sie den Kommentaren entnehmen, welche C-Anweisung die folgenden Assemblerinstruktionen repräsentieren.

Hauptprogramm

```
    ;local_packets=packets;
H1: lds r20, packets   ;low  8-bit
H2: lds r21, packets+1 ;high 8-bit
    ;if(local_packets > THRESHOLD)
H3: cpi r20, 0x0d
H4: sbci r21, 0xf0
H5: brcs <target>
```

Interruptbehandlung `INT0`

```
    ;packets += 8;
I1: lds r22, packets   ;low byte
I2: lds r23, packets+1 ;high byte
I3: adiw r22, 8
I4: sts packets+1, r23 ;high byte
I5: sts packets, r22   ;low byte
```

a) Benennen Sie das Nebenläufigkeitsproblem. (1 Punkt)

b) Demonstrieren Sie einen konkreten Programmablauf, bei dem das Nebenläufigkeitsproblem auftritt. (5 Punkte)

Tragen Sie dafür die relevanten Speicher- und Registerinhalte **nach** der Ausführung jeder Assemblerinstruktion in die nachfolgende Tabelle ein. Gehen Sie davon aus, dass die Variable `packets` initial den Wert `0x00fc` hat und treffen Sie keine Annahmen über andere Variablen- oder Speicherinhalte. (Siehe erste Zeile: was wir nicht wissen wird entsprechend nicht definiert.)
Kennzeichnen Sie auch, wann gegebenfalls ein Interrupt auftritt.

Zeile	**packets**	**r20**	**r21**	**r22**	**r23**
–	`0x00fc`	–	–	–	–

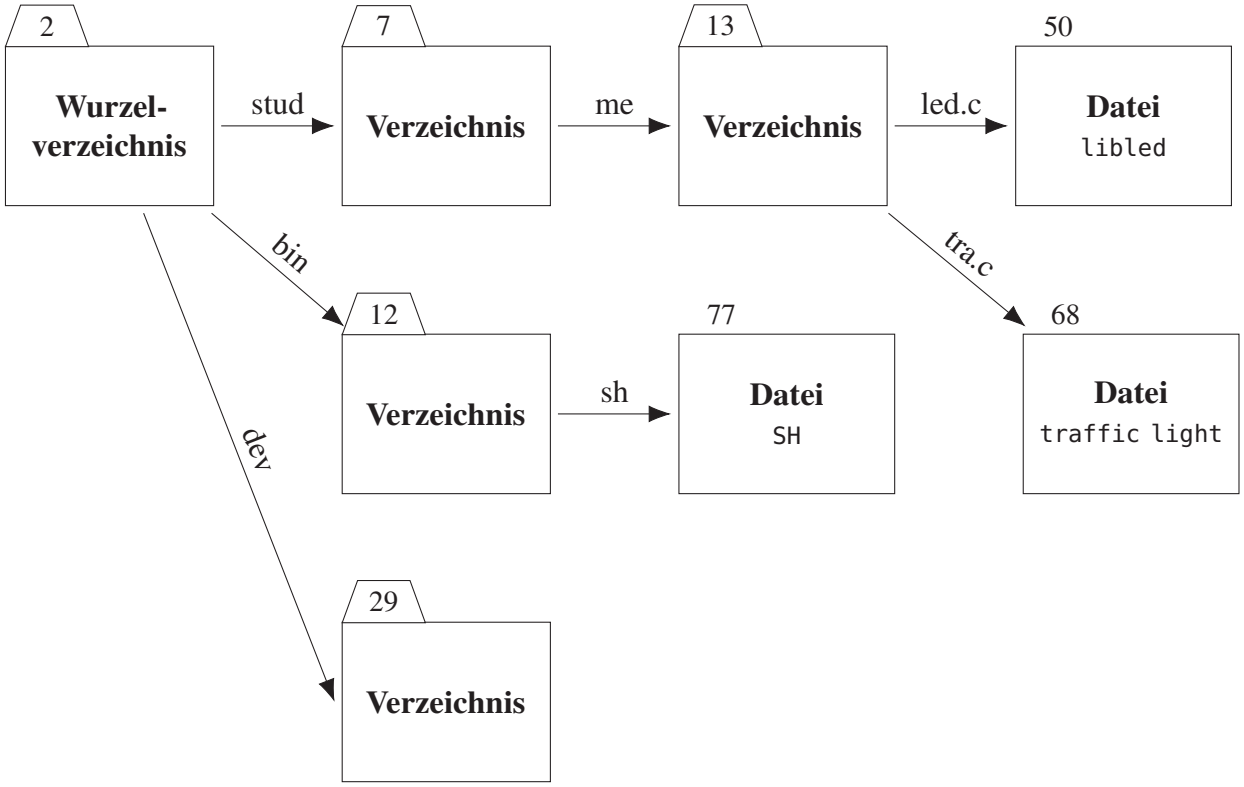
c) Welche(r) Speicher- bzw. Registerinhalt(e) enthalten nach dem von Ihnen beschriebenen Programmablauf nun falsche Werte. Nennen Sie außerdem die/den jeweils korrekten Wert(e).
(1 Punkt)

d) Mit welchem Mechanismus ließe sich das Nebenläufigkeitsproblem lösen? Benennen Sie entweder das konzeptuelle Vorgehen oder schreiben Sie den betreffenden Code hier um, sodass das Problem nicht mehr auftritt. (1 Punkt)

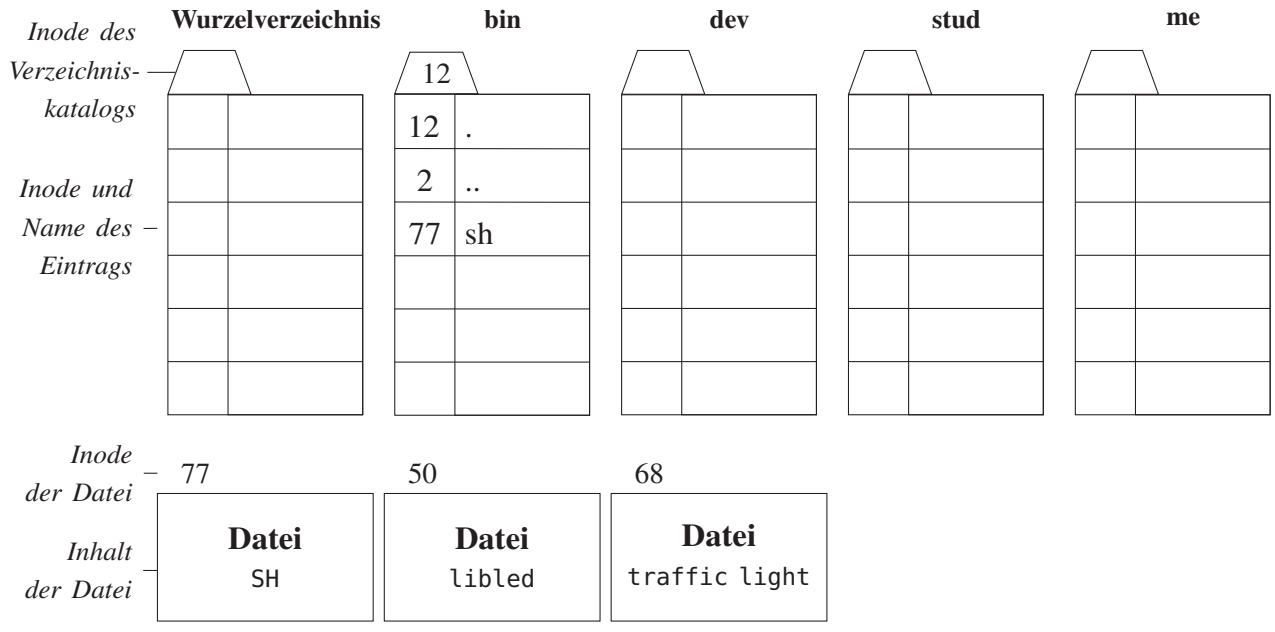
e) Ist die Verwendung des Schlüsselworts `volatile` für die Deklaration der Variablen `packets` nötig? Begründen Sie kurz. (2 Punkte)

Aufgabe 6: Dateisysteme (9 Punkte)

Ein Dateisystem ermöglicht das strukturierte Ablegen von Daten. Nachfolgend ist ein Verzeichnisbaum abgebildet. Verzeichnisse und Dateien sind mit einem beschrifteten Rechteck gekennzeichnet, ein Verweis mit einem beschrifteten Pfeil.



a) Vervollständigen Sie die dazugehörigen, vereinfachten Verwaltungsinformationen. Orientieren Sie sich dabei an dem bereits vorgebenen Schema. Tragen Sie **auch** die Einträge für `.` und `..` ein. (7 Punkte)



b) Nun wird folgender Code im Rahmen eines gültigen Programms erfolgreich ausgeführt. Das heißt, dass kein Fehlercode zurückgegeben wird und das Programm auch nicht vorzeitig beendet wird. Was wird dieses Programm im Dateibaum bewirken? Bedenken Sie, welche Datei erzeugt wird. Welchen Typ hat die Datei? Welchen Inhalt hat die Datei? Tragen Sie sämtliche Änderungen, die dieses Programm bewirken wird in den Dateibaum ein. Ignorieren sie die Verwaltungsinformationen. (2 Punkte)

Eine Datei soll dabei in folgender Struktur dargestellt sein:

<Inodenummer>

<Typ>
<Inhalt>

Code, der ausgeführt wird:

```
#include <unistd.h>

// Symlink von /dev/home/ zu /stud/me/
symlink("/stud/me", "/dev/home");
```