Aufgabe 1: (16 Punkte)

Bei den Multiple-Choice-Fragen ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Multiple-Choice-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) Folgende Makrodefinition findet sich in der AVR-Bibliothek: 2 Punkte

```
#define PINA (*(volatile uint8_t *)0x39)
```

Welche der folgenden Aussagen bezüglich der Verwendung des `volatile`-Schlüsselworts ist in diesem Fall richtig?

- ❏ Das `volatile`-Schlüsselwort stellt hier sicher, dass der Zugriff auf `PINA` mit Interrupts synchronisiert wird.
- ❏ Das `volatile`-Schlüsselwort ermöglicht den sicheren Zugriff auf einzelne Bits des Registers.
- ❏ Auf der AVR-Plattform werden Hardware-Register (wie `PINA`) im RAM eingeblendet. Das `volatile`-Schlüsselwort zeigt an, dass es sich dabei um flüchtigen Speicher handelt.
- ❏ Wird der Port A als Eingang konfiguriert, könnte sich der Wert von `PINA` jederzeit ändern. Durch `volatile` wird der Compiler angewiesen, stets den aktuellen Wert aus `PINA` zu lesen.

b) Welche Aussage zu folgender Funktion ist richtig? 2 Punkte

```
int *foo() {
   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` ist über die Laufzeit der `foo()`-Funktion hinaus gültig. Daher kann der zurückgelieferte Zeiger sicher vom Aufrufer verwendet werden.
- ❏ Die Variable `bar` enthält beim Verlassen der `foo()`-Funktion immer den Wert 1, da `bar` bei jedem Aufruf von `foo()` mit 0 initialisiert wird.
- ❏ 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.

c) Welche der folgenden Aussagen zum Begriff der Rücksprungadresse ist richtig? 2 Punkte

- ❏ Bei Aufruf einer Funktion über einen Funktionszeiger muss der Programmierer eine Rücksprungadresse angeben, an der das Programm später fortgesetzt werden soll.
- ❏ Die Rücksprungadresse ermöglicht die Rückkehr ins Betriebssystem. Auf einer Mikrocontroller-Plattform ist sie allerdings nicht vorhanden.
- ❏ Bei Aufruf einer Funktion sichert der Prozessor selbsttätig die Adresse der folgenden Instruktion. Dies ist die Rücksprungadresse.
- ❏ Bei rekursiven Funktionsaufrufen erstellt der Compiler eine Rücksprungadresse um sicher zu stellen, dass die Rekursion terminiert.

d) Gegeben sei folgendes Programmfragment für einen AVR-Microcontroller: 2 Punkte

```
uint8_t a = 100;
uint8_t b;

b = a+a * 2-50;
```

Welche der folgenden Aussagen ist richtig?

- ❏ b hat nach Ausführung der Zuweisung den Wert 250.
- ❏ b hat nach Ausführung der Zuweisung den Wert 350.
- ❏ Während der Ausführung kommt es zu einem Bereichsüberlauf; auf der AVR-Plattform bleibt dieser jedoch unentdeckt.
- ❏ Der Compiler warnt zur Übersetzungszeit vor einem Bereichsüberlauf.

e) Gegeben sei folgender Ausschnitt eines C-Programms, welches eine Variable foo vom Typ `uint8_t` verwendet: 2 Punkte

```
foo &= ~0xaa;
foo ^=  0xaa;
```

Welche Aussage über foo ist nach der Ausführung der Anweisungen richtig?

- ❏ Das höchstwertige Bit in foo ist 1.
- ❏ Die ersten beiden Zeichen in der Zeichenkette foo sind ’aa’.
- ❏ Das niederwertigste Bit in foo ist 1.
- ❏ Über den Zustand des höchstwertigen Bits von foo kann keine Aussage getroffen werden.

f) Gegeben ist folgender Programmcode: 2 Punkte

```
uint16_t x[] = {1, 2, 4, 8};
uint16_t *y = &x[1];
y += 2;
```

Welchen Wert liefert die Dereferenzierung von `y`?

- ❑ 2
- ❑ 4
- ❑ 8
- ❑ Zur Laufzeit tritt ein Fehler auf.

g) Welche Aussage zu dem Systemaufruf `fork()` ist richtig? 2 Punkte

- ❑ Die im Vaterprozess geöffneten Dateien bleiben auch im Kindprozess geöffnet.
- ❑ Vater- und Kindprozesse besitzen den gleichen Speicher und können über diesen kommunizieren.
- ❑ `exec()` kann nur im Kindprozess aufgerufen werden.
- ❑ Da `fork()` im Kindprozess 0 zurückliefert, ist es für diesen unmöglich, die Prozess-Id des Vaterprozesses zu bestimmen.

h) Welche der folgenden Aussagen bzgl. der Interruptsteuerung ist richtig? 2 Punkte

- ❑ Pegelgesteuerte Interrupts werden beim Wechsel des Pegels ausgelöst, daher der Name.
- ❑ Pegelgesteuerte Interrupts müssen durch Pollen des Pegels abgefragt werden.
- ❑ Interrupts sind eine Besonderheit von AVR-Mikroprozessoren. Auf anderen Architekturen kommen POSIX-Signale zum Einsatz.
- ❑ Wurde gerade ein Flanken-gesteuerter Interrupt ausgelöst, so muss erst ein Pegelwechsel der Interruptleitung stattfinden, damit erneut ein Interrupt ausgelöst werden kann.

Aufgabe 2a: Bewässerungssystem (30 Punkte)

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

Schreiben Sie eine Steuerung für einen AVR-Mikrocontroller, die ein Bewässerungssystem für Pflanzen realisiert. Über einen Feuchtigkeitssensor wird der aktuelle Wasserstand periodisch abgefragt und bei Bedarf über eine Pumpe Wasser nachgefüllt. Eine LED-Leiste visualisiert zu jeder Zeit den aktuellen Füllstand.

Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion `void init(void)`, so dass die LEDs und die Pumpe aus sind. Treffen Sie keine Annahmen über den initialen Zustand der Hardware-Register.
- Zu Beginn befindet sich das Programm im Überwachungsmodus: Die LED-Leiste zeigt den aktuellen Wert des Feuchtigkeitssensors an, der alle 5 Minuten periodisch abgefragt wird. Dazu steht ein externer Zeitgeber zur Verfügung, der jede Sekunde einen Interrupt durch kurzzeitigen High-Pegel anfordert.
- Während des Wartens auf den Interrupt des Zeitgebers soll der Mikrocontroller jeweils zum Stromsparen in den Schlafmodus gehen.
- Der aktuelle Wert des Feuchtigkeitssensors wird von einem Analog-Digital-Umsetzer (ADC) bereitgestellt, dessen Wert mit der Bibliotheksfunktion `uint16_t adcread(void)` abgefragt werden kann. Die Funktion liefert einen 12-Bit-Wert zurück, wobei der Wert 0 den minimalen Füllstand angibt.
- Die LED-Leiste muss nach jeder Abfrage des Sensors aktualisiert werden. Implementieren Sie hierfür die Funktion `void showlevel(uint16_t value)`, in der der übergebene Wert gleichmäßig auf die acht LEDs abgebildet wird (eine leuchtende LED entspricht dem minimalen, acht LEDs dem maximalen Füllstand).
- Unterschreitet der Sensorwert die Hälfte des maximalen Wertes, muss die Pumpe eingeschaltet werden, bis der maximale Füllstand erreicht ist.
- Implementieren Sie die Überwachung des Pumpvorgangs durch periodisches Abfragen des Sensors alle 100 ms unter Verwendung einer aktiven Wartefunktion `void wait(uint16_t ms)` die `ms` Millisekunden wartet. Ihnen steht eine Präprozessorkonstante `LOOPS_PER_MS` zur Verfügung, die angibt, wieviele Schleifendurchläufe gewartet werden muss, um eine Millisekunde verstreichen zu lassen.

Information über die Hardware

Pumpe: **PORTC**, Pin 3, aktiviert bei High-Pegel
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRC**-Register auf 1

LEDs: **PORTB**, Pins 0-7, LED 1 an Pin 0, leuchtet bei Low-Pegel
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRB**-Register auf 1

Zeitgeber: **PORTD**, Interrupt-Leitung an Pin 3
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Register auf 0
- externe Interruptquelle **INT1**, ISR-Vektor-Makro: **INT1_vect**.
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT1**-Bits im Register **GICR**.
- IRQ wird durch kurzen externen High-Pegel signalisiert, der Pullup-Widerstand muss daher abgeschaltet sein (entsprechendes Bit in **PORTD**-Register auf 0 setzen).

Konfiguration der externen Interruptquelle 0 (Bits in Register **MCUCR**)

ISC11	ISC10	Beschreibung
0	0	Interrupt bei low Pegel
0	1	Interrupt bei beliebiger Flanke
1	0	Interrupt bei fallender Flanke
1	1	Interrupt bei steigender Flanke

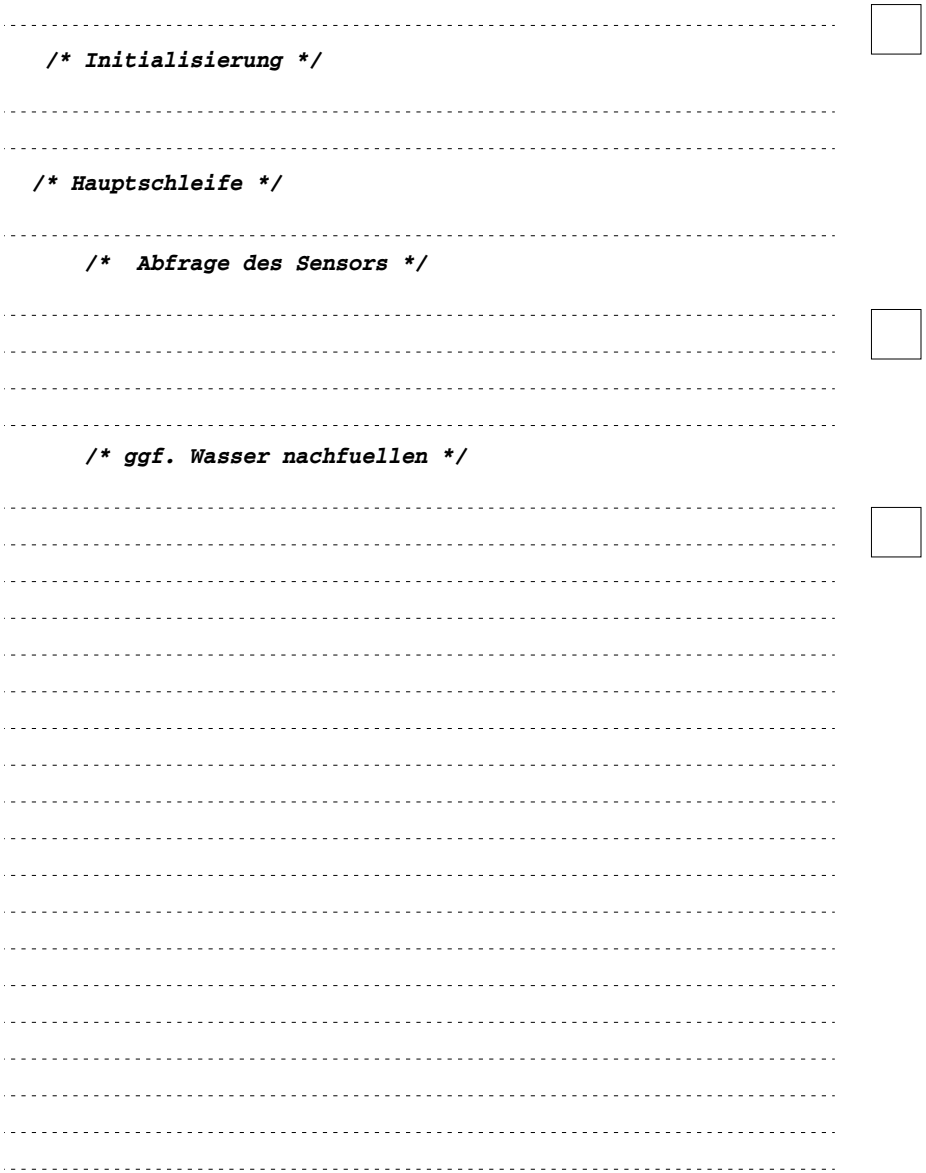
Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.

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

#define LOOPS_PER_MS 100

uint16_t adcread(void);
/* Funktionsdeklarationen, globale Variablen, etc. */
```

```
/* Unterbrechungsbehandlungsfunktion */
```

A:


```
/* Funktion main */
```



```
    /* Warten auf Impuls des Zeitgebers */
```

```
/* Ende main */

/* Funktion wait */
```

```
/* Anzeigefunktion fuer die LED-Leiste */
```

B:


```
/* Initialisierungsfunktion */
```

I:

Aufgabe 2b: simple shell (17 Punkte)

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

Schreiben Sie ein Programm, das Befehle, die auf der Standardeingabe übergeben werden, jeweils in einem neuen Kindprozess zur Ausführung bringt und danach den Exit-Status des ausgeführten Programms ausgibt.

Das Programm soll im Detail wie folgt funktionieren:

- Das Programm liest zeilenweise Befehle von der Standardeingabe. Diese Befehle bestehen immer aus dem Namen eines Programms, einem Leerzeichen und genau einem Argument.
- Sie können davon ausgehen, dass die eingelesenen Zeilen nicht länger als 1023 Zeichen sind, jedoch dürfen trotzdem keine Pufferüberläufe möglich sein.
- Das im eingelesenen Befehl bezeichnete Programm wird in einem neuen Kindprozess ausgeführt. Das Hauptprogramm wartet jeweils auf die Beendigung des Kindprozesses.
- Nach Beenden des Kindprozesses soll zwischen normalem **`exit()`** und dem Beenden durch ein Signal unterschieden werden und eine entsprechende Meldung auf dem Standardausgabekanal erfolgen:

  Normales Beenden:
  `X` `returned` `Y`
  Signal:
  `X` `killed by signal` `Y`

  Wobei ***`X`*** immer den Namen des jeweiligen Programs entsprechen soll und ***`Y`*** entweder dem zurückgegebenem Exit-Status oder dem Signal entspricht.
- Alle Fehlermeldungen sollen auf den Standardfehlerkanal stderr ausgegeben werden, nicht auf die Standardausgabe.

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


```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>

// Funktion main

   // lokale Variablen

   // Hauptschleife:
   // jeweils Zeile einlesen und in Kommando und Argument teilen
```

```
    // Kommando ausführen und auf Beendigung warten
```

```
  // Ende der Hauptschleife
```

```
// Ende main
```

S:

Aufgabe 3: (18 Punkte)

Die folgenden Beschreibungen sollen kurz und prägnant erfolgen (Stichworte, kurze Sätze).

a) Was versteht man in der Informatik unter einem Modul und wozu dient es?
(4 Punkte)

b) Wie werden Module in C realisiert?
(4 Punkte)

c) Ein Mikrocontroller-Programm "main.c" verwende die Funktionen

```
void send(char);
char receive(void);
```

eines Netzwerk-Moduls. Das Modul implementiert diese Funktionen und verwaltet zudem einen Puffer für eingehende Zeichen und eine Unterbrechungsbehandlungsfunktion, die diesen befüllt:

```
char buf[32];
void receive_irq(void){
  ...
}
```

Skizzieren Sie, welche Dateien (netzwerk.*) erforderlich sind, um das Netzwerk-Modul bereit zu stellen und im Hauptprogramm verwenden zu können. Was ist zu beachten (d.h. "was muss in welcher Datei wie stehen"), um eine korrekte Kapselung der Modulinterna sowie die korrekte Sichtbarkeit und Verwendung der Modulschnittstellen zu gewährleisten? (7 Punkte)

d) In Java gibt es zum Zwecke der Informationsverbergung (information hiding) die Sichtbarkeitsattribute "public", "protected" und "private". Nennen Sie - sofern vorhanden - die Entsprechungen in C! (3 Punkte)

Aufgabe 4: (9 Punkte)

Die folgenden Beschreibungen sollen kurz und prägnant erfolgen (Stichworte, kurze Sätze).

a) Beschreiben Sie das Konzept des Dateisystems und seine grundlegenden Abstraktionen. (4 Punkte)

b) In einem leeren UNIX Dateisystem werden folgende Kommandos ausgeführt:

```
cd /
mkdir foo
mkdir bar
```

Skizzieren Sie den Aufbau der Verzeichnisstruktur.
Wieviele Inodes und wieviele Links sind anschließend auf dem Dateisystem in Verwendung?