Aufgabe 1: (20 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) Welche der folgenden Aussagen über den C-Präprozessor ist richtig? 2 Punkte

- ❏ Der Präprozessor optimiert Makros durch Zeigerarithmetik.
- ❏ Nach dem Übersetzen und dem Binden müssen C-Programme durch den Präprozessor nachbearbeitet werden, um Makros aufzulösen.
- ❏ Der Präprozessor ist eine Softwarekomponente, welche Java-Klassen durch C-Funktionen ersetzt, die dann von einem C-Compiler übersetzt werden.
- ❏ Die Syntax von Präprozessoranweisungen ist unabhängig vom Rest der Sprache C.

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

- ❏ Wurde gerade ein Pegel-gesteuerter Interrupt ausgelöst, so muss erst ein Pegelwechsel der Interruptleitung stattfinden, bevor erneut ein Interrupt ausgelöst wird.
- ❏ Beim Auftreten eines Interrupts sichert der Prozessor automatisch Register in den Speicher.
- ❏ Pegel-gesteuerte Interrupts werden bei jedem Wechsel des Pegels ausgelöst.
- ❏ Flanken-gesteuerten Interrupts können nicht blockiert werden, da sie völlig unvorhersehbar auftreten.

c) Gegeben ist folgender Programmcode: 2 Punkte

```
#define SUB(a,b) a-b
#define ADD(a,b) a+b
```

Was ist das Ergebnis des folgenden Ausdrucks?

```
2 * SUB(2, ADD(3, 4))
```

- ❏ -10
- ❏ -3
- ❏ 5
- ❏ 6

d) Wie viele Bytes belegt die folgende Struktur im Speicher eines AVR-Mikrocontrollers: 2 Punkte

```
union {
    struct {
        uint16_t lo, hi;
    };
    uint32_t r32;
} reg;
```

- ❏ 2 Bytes
- ❏ 4 Bytes
- ❏ 16 Bytes
- ❏ 32 Bytes

e) 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`?

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

f) Welche der folgenden Aussagen zur Sichtbarkeit von Variablen ist richtig? 2 Punkte

- ❏ Globale static-Variablen sind in allen Programmteilen immer direkt zugreifbar.
- ❏ Eine lokale static-Variable ist nur innerhalb der Funktion, in der sie definiert wurde, sichtbar.
- ❏ Lokale static-Variablen sind aus anderen Modulen nur dann zugreifbar, wenn sie außerdem als "extern" deklariert wurden.
- ❏ Wenn eine globale und eine lokale Variable gleichen Namens existieren, dann hat die globale Variable Vorrang.

g) Welche Aussage zu *volatile* ist richtig? 2 Punkte

- ❏ Das Schlüsselwort *volatile* unterbindet Optimierungen an einer damit definierten Variable.
- ❏ Das Schlüsselwort *volatile* unterbindet alle Nebenläufigkeitsprobleme.
- ❏ Das Schlüsselwort *volatile* hat nur noch eine historische Bedeutung und ist in heutigen Programmen nicht mehr nötig.
- ❏ Das Schlüsselwort *volatile* erlaubt dem Compiler bessere Optimierungen durchzuführen.

h) Was ist ein Stack-Frame? 2 Punkte

- ❏ Ein bei Interrupts auftretendes spezielles Nebenläufigkeitsproblem.
- ❏ Ein Interrupt, der einen Überlauf des Stack-Speichers signalisiert.
- ❏ Ein Bereich des Speichers, in dem unter anderem Übergabeparameter für Funktionen abgelegt werden.
- ❏ Ein Bereich des Speichers, in dem lokale static-Variablen von Funktionen abgelegt werden.

i) Welche Aussage zur Speicherallokation ist richtig? 2 Punkte

- ❏ Die dynamische Allokation von Speicher ist auf einem Mikrocontroller zu bevorzugen, da erst zur Laufzeit geprüft wird, ob der Speicher wirklich zur Verfügung steht.
- ❏ Die Speicheradresse von statisch allokierten Variablen kann sich zur Laufzeit ändern.
- ❏ automatic-Variablen werden im Heap allokiert.
- ❏ Die Verwendung von statisch allokierten Variablen erlaubt den Speicherbedarf bereits nach dem Binden abzuschätzen.

j) In welchen Situationen wird ein Prozess in den Zustand "blockiert" versetzt? 2 Punkte

- ❏ während der Prozesserzeugung solange die Verwaltungsstrukturen noch nicht angelegt sind
- ❏ bei jedem Aufruf der V-Operation eines Semaphors
- ❏ 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 2a: Kurzzeitwecker (31 Punkte)

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

Schreiben Sie ein Programm für einen AVR-Mikrocontroller, das einen Kurzzeitwecker implementiert. Durch Drücken und Halten des Tasters wird der Wecker in die Konfigurationsphase versetzt. In dieser Phase wird mit einem Potentiometer die gewünschte Dauer eingestellt, die auf einer Flüssigkristallanzeige angezeigt wird. Durch Loslassen des Tasters wird die eingestellte Zeit heruntergezählt und nach deren Ablauf ein Weckton erzeugt.

Ihr Programm soll wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion `void init(void);`. Es sollen keine Annahmen über den initialen Zustand der Hardware-Register getroffen werden. Die Anzeige zeigt 0 Minuten und 0 Sekunden.
- Das Programm wartet nach dem Starten im Schlafmodus auf einen Tastendruck.
- Während der Taster gedrückt ist, soll in der Konfigurationsphase kontinuierlich das Potentiometer mit der Bibliotheksfunktion `uint16_t poti_read(void);` ausgelesen werden. Diese liefert einen 10-Bit-Wert zurück, welcher der gewählten Dauer in Sekunden entspricht. Der aktuelle Wert wird auf der Anzeige dargestellt, welche sich mit der Bibliotheksfunktion `void display_show(uint8_t minutes, uint8_t seconds);` ansteuern lässt.
- Durch Loslassen des Tasters wird der aktuell eingestellte Wert übernommen und die Countdown-Phase beginnt. Die noch verbleibende Zeit wird auf der Anzeige jede Sekunde heruntergezählt.
- Der Weckton wird durch das Einschalten eines Tongebers für 500 ms erzeugt. Danach wartet das Programm auf eine erneute Konfiguration.

Beachten Sie, dass nach einem Abbruch der Taster zuerst wieder losgelassen werden muss, bevor die Konfigurationsphase erneut betreten werden kann.

Implementieren Sie die Wartezeit unter Verwendung einer aktiven Wartefunktion `uint8_t wait(uint16_t ms);`, die `ms` Millisekunden wartet. Ihnen steht eine Präprozessorkonstante `LOOPS_PER_MS` zur Verfügung, die angibt, wieviele Schleifendurchläufe einer Millisekunde entsprechen.

Wird während des Wartens erneut der Taster betätigt, wird die Wartezeit sofort abgebrochen und die Funktion mit dem Rückgabewert ungleich 0 verlassen. Der Rückgabewert 0 zeigt an, dass kein Tastendruck stattgefunden hat.

Information über die Hardware

Tongeber: **PORTB**, Pin 5, aktiviert bei High-Pegel
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRB**-Register auf 1

Taster: **PORTD**, Pin 2
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Register auf 0
- Zustand abfragen: entsprechendes Bit im **PIND**-Register lesen.
- externe Interruptquelle **INT0**, ISR-Vektor-Makro: **INT0_vect**.
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT0**-Bits im Register **GICR**.
- der Taster verbindet den Pin mit Masse, es muss der interne Pullup-Widerstand verwendet werden (entsprechendes Bit in **PORTD**-Register auf 1 setzen).

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

ISC01	ISC00	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 50

uint16_t poti_read(void);
void display_show(uint8_t minutes, uint8_t seconds);

/* Funktionsdeklarationen, globale Variablen, etc. */
```

```
/* Unterbrechungsbehandlungsfunktion */
```

A:


```
/* Funktion main */

   /* Initialisierung und lokale Variablen */

  /* Hauptschleife */

      /* Schlafphase */

      /* Konfigurationsphase */
```

```
    /* Countdown-Phase */
```

```
    /* Tongeber auslösen */
```

```
/* Ende main */
```

M:


```
/* Funktion wait */
```

```
/* Initialisierungsfunktion */
```

I:

Aufgabe 2b: rundir (16 Punkte)

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

Schreiben Sie ein Programm **`rundir`**, welches ein anderes Programm mit jeder Datei des aktuellen Verzeichnisses als Parameter ausführt.

Das auszuführende Programm wird als Kommandozeilenparameter übergeben.

Dateinamen, die mit einem ’.’ beginnen, werden ignoriert.

Das Programm soll im Einzelnen wie folgt funktionieren:

- Nach dem Start sollen zunächst die Kommandozeilenparameter überprüft werden. Sie müssen nicht sicherstellen, dass das auszuführende Programm existiert.
- Das aktuelle Verzeichnis wird geöffnet und alle Dateien, die nicht mit ’.’ beginnen, werden abgearbeitet. Eine besondere Reihenfolge muss hierbei nicht beachtet werden.
- Für jede Datei wird ein Kindprozess erzeugt, in welchem das Kommando ausgeführt wird.
- Vor Start des nächsten Kommandos wird auf das Ende der Bearbeitung gewartet. Der Exit-Status muss nicht ausgegeben werden.
- Beim Auftreten eines Fehlers soll die Abarbeitung nach Möglichkeit fortgesetzt werden. Achten Sie auf sinnvolle Fehlermeldungen auf dem Standardfehlerausgabekanal.

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 <unistd.h>

// Funktion main

.....

.....

   // lokale Variablen

.....

.....

.....

.....

.....

.....

   // Aufrufargumente pruefen

.....

.....

.....

.....

.....

.....

.....
```

```
// Verzeichnis oeffnen und durchsuchen
```

```
// Ende main
```

D:

Aufgabe 3: (10 Punkte)

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

a) Wie wird die Parameterübergabe über den Stack realisiert? Beschreiben Sie, wie die tatsächlichen Parameter abgelegt werden und wie der Zugriff über die formalen Parameter erfolgt (ggf. durch Skizze veranschaulichen!) (6 Punkte)

b) In Bezug auf die Semantik bei der Übergabe von Parametern an Funktionen unterscheidet man generell die Konzepte *call by value* und *call by reference*. Beschreiben Sie die beiden Konzepte! (4 Punkte)

Aufgabe 4: (13 Punkte)

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

a) Was versteht man unter Nebenläufigkeit? Geben Sie eine kurze Definition des Begriffs. (2 Punkte)

b) In welchen Situationen tritt Nebenläufigkeit auf? (2 Punkte)

c) Bei Nebenläufigkeit können verschiedene Probleme auftreten. Beschreiben Sie exemplarisch zwei solche Probleme und wie man mit ihnen umgeht, um korrekt funktionierende Software zu gewährleisten. (5 Punkte)

d) Man unterscheidet bei Ausnahmesituationen zwischen Traps und Interrupts. Beschreiben Sie die beiden Arten und die wesentlichen Unterschiede. (4 Punkte)