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) Gegeben ist folgender Programmcode: 2 Punkte

```
int32_t x[] = {-1, 7, -3, 5};
int32_t *y = &x[3];
y -= 2;
```

Welchen Wert liefert die Dereferenzierung von `y` (also `*y`)?

- ❑ 3
- ❑ -5
- ❑ 7
- ❑ Zur Laufzeit tritt ein Fehler auf.

b) Welchen Wert hat die Variable c nach der Ausführung der folgenden Anweisungen? 2 Punkte

```
int a = 42;
int *p = &a;
int c = *p + *(&a);
```

- ❑ 42
- ❑ 84
- ❑ das Programm führt zu einem Laufzeitfehler
- ❑ das Programm führt zu einem Übersetzungsfehler

c) 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.
- ❑ Die Syntax von Präprozessoranweisungen ist unabhängig vom Rest der Sprache C.
- ❑ Der Präprozessor ist eine Softwarekomponente, welche Java-Klassen durch C-Funktionen ersetzt, die dann von einem C-Compiler übersetzt werden.

d) Welche Aussage zu Zeigern ist richtig? 2 Punkte

- ❑ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-value.
- ❑ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-reference.
- ❑ Ein Zeiger kann zur Manipulation von schreibgeschützten Datenbereichen verwendet werden.
- ❑ Zeiger vom Typ `void*` sind am besten für Zeigerarithmetik geeignet, da Sie kompatibel zu jedem Zeigertyp sind.

e) Was versteht man beim Zugriff auf I/O-Register unter dem Begriff "Memory-mapped"? 2 Punkte

- ❑ Der Zugriff auf die Register erfolgt mit speziellen mmap-Instruktionen des Prozessors.
- ❑ Die Register sind in den normalen Adressraum des Prozessors eingeblendet und der Zugriff erfolgt mit den normalen Speicherzugriffsinstruktionen.
- ❑ Beim Zugriff auf spezielle Speicherbereiche des Hauptspeichers werden die Inhalte der Hauptspeicherzellen automatisch in Geräteregister umkopiert.
- ❑ Die Register sind nicht real, sondern nur virtuell im Hauptspeicher vorhanden (sog. "virtual devices").

f) Welchen Wert enthält die Variable a nach Ausführung der folgenden Code-Zeilen? 2 Punkte

```
int a = 2;
a ^= a;
a = a | (1 << 2);
```

- ❑ 0
- ❑ 2
- ❑ 4
- ❑ 6

g) Was ist ein Stack-Frame? 2 Punkte

- ❏ Der Speicherbereich, in dem der Programmcode einer Funktion abgelegt ist.
- ❏ Ein spezieller Registersatz des Prozessors zur Bearbeitung von Funktionen.
- ❏ Ein Fehler, der bei unberechtigten Zugriffen auf den Stack-Speicher entsteht.
- ❏ Ein Bereich des Speichers, in dem u.a. lokale automatic-Variablen einer Funktion abgelegt sind.

h) Was versteht man unter Nebenläufigkeit? 2 Punkte

- ❏ Wenn ein Programm abwechselnd auf zwei verschiedene Speicherbereiche zugreift.
- ❏ Wenn ein Programmabschnitt in einer Schleife mehrfach durchlaufen wird.
- ❏ Die Programmabschnitte im if- und else-Teil einer bedingten Anweisung.
- ❏ Wenn für zwei Befehle aus zwei Programmabläufen nicht feststeht, welcher von beiden tatsächlich zuerst ausgeführt werden wird.

i) Welche der folgenden Aussagen zum Thema Threads ist **richtig**? 2 Punkte

- ❏ Kernel-Level-Threads können blockieren, ohne andere Threads zu behindern.
- ❏ User-Level-Threads sind die effizienteste Möglichkeit ein Multiprozessorsystem zu nutzen.
- ❏ Kernel-Level-Threads dürfen in normalen Anwendungsprogrammen aus Sicherheitsgründen nicht verwendet werden.
- ❏ Im Gegensatz zu User-Level-Threads läuft jeder Kernel-Level-Thread in einem eigenen Adressraum.

j) In Betriebssystemen wie Linux oder Windows unterscheidet man die Begriffe Programm und Prozess. Welche Aussage ist **richtig**? 2 Punkte

- ❏ Programme sind C-Quellcode-Dateien, die durch einen C-Compiler in einen lauffähigen Prozess übersetzt werden können.
- ❏ Prozesse können durch einen Aufruf der Funktion `exec(...)` terminiert werden.
- ❏ Ein Prozess kann mehrere Kindprogramme ausführen.
- ❏ Ein Prozess ist ein Programm in Ausführung

Aufgabe 2a: Sonnenschutz (30 Punkte)

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

Schreiben Sie ein Programm für einen AVR-Mikrocontroller, das eine Steuerung für einen Sonnenschutz vor einem Fenster realisiert. Über einen Helligkeitssensor wird die aktuelle Sonneneinstrahlung periodisch abgefragt und bei Bedarf wird der Sonnenschutz über einen Motor nach oben oder unten bewegt.

Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion `void init(void);`, sodass der Motor aus ist. Treffen Sie keine Annahmen über den initialen Zustand der Hardware-Register.
- Zu Beginn befindet sich das Programm im Überwachungsmodus: Der Helligkeitssensor soll periodisch abgefragt werden. Eine Abfrage wird durch einen externen Zeitgeber über einen Interrupt (kurzzeitiger High-Pegel) angefordert. Das Eintreten des Interrupts muss in dem Programm durch Setzen einer Event-Variable vermerkt werden.
- Während des Wartens auf den Interrupt des Zeitgebers soll der Mikrocontroller zum Stromsparen in den Schlafmodus gehen.
- Der aktuelle Wert des Helligkeitssensors wird von einem Analog-Digital-Umsetzer (ADC) bereitgestellt, dessen Wert mit der Bibliotheksfunktion `uint16_t adcread(void);` abgefragt werden kann. Die Funktion liefert einen 10-Bit-Wert zurück, wobei der Wert 0 die minimale Helligkeit angibt.
- Überschreitet der Sensorwert 75 % des maximalen Helligkeitswertes, soll der Sonnenschutz heruntergefahren werden. Unterschreitet der Sensorwert 25 % des maximalen Helligkeitswertes, soll der Sonnenschutz hinaufgefahren werden.
- Implementieren Sie die Überwachung der Bewegung durch Abfragen der Endschalter des Sonnenschutzes in 100ms-Intervallen. Nutzen sie hierzu eine aktive Wartefunktion `void wait(uint16_t ms);`, die `ms` Millisekunden wartet. Die Präprozessorkonstante `LOOPS_PER_MS` gibt an, wieviele Schleifendurchläufe gewartet werden muss, um eine Millisekunde verstreichen zu lassen. Die Endschalter sind jeweils geschlossen, wenn die Position "oben" bzw. "unten" erreicht ist. Wird eine Endposition erreicht, soll der Motor abgeschalten werden.

Information über die Hardware

Motor: **PORTA**, Pin 2 hoch, Pin 3 runter
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRA**-Reg. auf 1
- High-Pegel auf Pin 2 aktiviert Motor zum Hochfahren des Rollos, High-Pegel auf Pin 3 zum Herunterfahren.
- Es darf nie mehr als eine Phase aktiv sein.

Endschalter:**PORTB**, Pin 4 oben, Pin 5 unten
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRB**-Register auf 0
- Die Endschalter verbinden den jeweiligen Pin mit Masse, es muss der interne Pull-up-Widerstand verwendet werden (entsprechendes Bit in **PORTB**-Register auf 1 setzen).

Zeitgeber:**PORTD**, Interrupt-Leitung an Pin 2
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Register auf 0
- externe Interruptquelle **INT0**, ISR-Vektor-Makro: **INT0_vect**.
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT0**-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**)

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 100

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

```
/* Unterbrechungsbehandlungsfunktion */
```

A:


```
/* Funktion main */

   /* Initialisierung */

  /* Hauptschleife */

      /*  Warten auf Impuls des Zeitgebers */

      /*  Sensor abfragen */
```

```
/* Sonnenschutz ggf. hinauf/hinunter bewegen */
```

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

B:


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

```
/* Initialisierungsfunktion */
```

I:

Aufgabe 2b: archive-files (15 Punkte)

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

Schreiben Sie ein Programm **archive-files**, das Dateien archiviert, deren Pfade als Argumente an das Programm übergeben werden. Das Programm archiviert nur reguläre Dateien, wenn diese eine bestimmte Größe überschritten haben (Anwendung z. B. zur regelmäßigen Archivierung von Protokolldateien).

Das Programm soll im Detail wie folgt funktionieren:

- Das Programm **archive-files** bekommt als Argumente eine Menge von Dateipfaden übergeben, wobei jedes Argument einzeln verarbeitet wird.
- Nur wenn es sich bei einem übergebenen Argument um eine reguläre Datei handelt, die größer als das vordefinierte Makro **ARCHIVE_SIZE** ist, wird sie archiviert; sonst wird das Argument ohne weitere Ausgabe ignoriert.
- Für das eigentliche Archivieren wird ein neuer Kindprozess erzeugt. In diesem Kindprozess wird anschließend das bereits existierende Programm **/usr/bin/archive** aufgerufen mit der zu archivierenden Datei als Argument. (Ein Kommandozeilen-Aufruf dieses Programms für die Datei **my_file.txt** würde exemplarisch wie folgt aussehen: **/usr/bin/archive my_file.txt**)
- Der Vaterprozess wartet auf die Beendigung des Kindprozesses und gibt den Status mit Hilfe der bereits existierenden Funktion **void print_status(int);** aus.

Hinweise:

- Achten Sie auf eine korrekte Fehlerbehandlung der verwendeten Funktionen.
- Die Ausgabe von Fehlern soll mit Hilfe der errno-Variable auf dem stderr-Kanal erfolgen.

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


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

void print_status(int);

// Funktion main

.....

.....

   // lokale Variablen

.....

.....

.....

   // Hauptschleife: Argumente bearbeiten,
   // Dateiattribute ermitteln

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....
```

```
    // passende Datei? wenn ja, archivieren
```

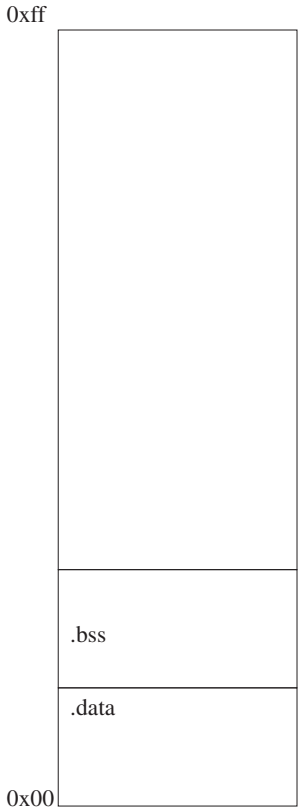
```
   // Ende der Hauptschleife
```

```
// Ende main
```

R:

Aufgabe 3: (6 Punkte)

Die folgende Abbildung soll den Speicheraufbau einer Programmausführung im RAM eines Mikrocontrollers skizzieren. Dazu ist das danebenstehende Programm gegeben.




```
int a;
int b = 42;

void main() {
    static int s = 3;
    int x;
    char *p = malloc(100);
}
```

a) Ergänzen Sie Stack und Heap sowie deren Wachstumsrichtung in der Abbildung. (2 Punkte)

b) Zeichnen Sie den mit Hilfe von malloc dynamisch allozierten Speicher ein und kennzeichnen sie ihn mit dem Buchstaben M. (1 Punkt)

c) Ordnen Sie die Variablen a, b, s, x und p den dargestellten Speicherbereichen zu und kennzeichnen Sie für Zeigervariablen den referenzierten Speicherbereich mit einem Pfeil. (3 Punkte)

Aufgabe 4: (10 Punkte)

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

a) Welche Probleme ergeben sich aus der Verwendung von Interrupts und warum ist ihr Einsatz dennoch sinnvoll? (nennen Sie ein Problem und einen Nutzen) (2 Punkte)

b) In welchen Schritten wird ein Interrupt typischerweise auf einem Mikrocontroller bearbeitet? (3 Punkte)

c) Sie kennen das Schlüsselwort `volatile`. Beschreiben Sie die Wirkung, die Notwendigkeit und den Nachteil bei unnötiger Verwendung. (3 Punkte)

d) Warum sollten Interruptbehandlungen so kurz wie möglich gehalten werden? (2 Punkte)

Aufgabe 5: (9 Punkte)

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

a) Welche Schritte sind notwendig um von einer in der Programmiersprache C geschriebenen Anwendung, die aus mehreren Modulen besteht, zu einem ausführbaren Programm zu gelangen? Wie bringt man ein solches Programm in einer Betriebssystemumgebung (wie Linux) und auf einem Mikrocontroller zur Ausführung? (3 Punkte)

b) Eine wesentliche Aufgabe von Betriebssystemen ist die Verwaltung von Betriebsmitteln. Geben Sie eine Klassifizierung von Betriebsmitteln an und nennen Sie jeweils ein typisches Beispiel. (6 Punkte)