Aufgabe 1: (18 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) Welche der folgenden Aussagen bzgl. der Interruptsteuerung ist richtig?

2 Punkte

- ☐ Flanken-gesteuerten Interrupts können nicht blockiert werden, da sie völlig unvorhersehbar auftreten.
- ☐ Während der Bearbeitung eines Interrupts nimmt der Prozessor keine weiteren Interrupts an.
- ☐ Wurde gerade ein Pegel-gesteuerter Interrupt ausgelöst, so muss erst ein Pegelwechsel der Interruptleitung stattfinden, bevor erneut ein Interrupt ausgelöst wird.
- ☐ Pegel-gesteuerte Interrupts werden beim Wechsel des Pegels ausgelöst.

b) Welche der folgenden Aussagen zum Begriff *Rücksprungadresse* ist richtig?

2 Punkte

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

c) Welche Aussage zu Zeigern ist richtig?

2 Punkte

- ☐ Ein Zeiger kann zur Manipulation von schreibgeschützten Datenbereichen verwendet werden.
- ☐ Beim Rechnen mit Zeigern muss immer der Typ des Zeigers beachtet werden.
- ☐ Zeiger vom Typ `void*` benötigen weniger Speicher als andere Zeiger, da bei anderen Zeigertypen zusätzlich die Größe gespeichert werden muss.
- ☐ Die Speicherstelle, auf die ein Zeiger verweist, kann niemals selbst einen Zeiger enthalten.

d) Was bewirkt die folgende Funktion?

2 Punkte

```
void func(int s1, int s2) {
    int tmp = s1;
    s1 = s2;
    s2 = tmp;
}
```

- ☐ Der Compiler meldet bei der Übersetzung einen Fehler, weil die Funktionsparameter `s1` und `s2` nicht verändert werden dürfen.
- ☐ Bei einem Aufruf vertauscht die Funktion die Inhalte der an die formalen Parameter `s1` und `s2` übergebenen Variablen.
- ☐ Da in C Funktionen mit *call by value* aufgerufen werden, erhält die Funktion Kopien der Aufrufparameter, die sie vertauscht. Beim Aufrufer hat dies allerdings keine Auswirkungen.
- ☐ Die von `s2` adressierte Speicherzelle enthält nach dem Aufruf die in der Variablen `s1` abgelegte Speicheradresse.

e) 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 ermöglicht den sicheren Zugriff auf einzelne Bits des Registers.
- ☐ 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.
- ☐ Das `volatile`-Schlüsselwort stellt hier sicher, dass der Zugriff auf `PINA` mit Interrupts synchronisiert wird.
- ☐ 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.

f) Welche Aussage zu Variablen in C ist richtig?

2 Punkte

- ☐ Variablen vom Typ `int` haben auf allen Hardware-Plattformen dieselbe Größe.
- ☐ Wenn man bei einer `int8_t`-Variable zum Wert 127 eins dazu addiert, hat sie den Wert -128.
- ☐ Alle Variablen werden bei ihrer Deklaration mit dem Wert 0 initialisiert.
- ☐ Eine `int8_t`-Variable braucht mehr Speicher als eine `uint8_t`-Variable, weil zusätzlicher Platz für das Vorzeichen-Bit benötigt wird.

g) Gegeben ist folgender Programmcode: 2 Punkte

```
#define ABS(a) ((a) < 0 ? (-1) * (a) : (a))
int8_t a = 23;
int8_t b = ABS(++a);
```

Welchen Wert hat die Variable `b` nach Ausführung des Codes?

- ☐ -24
- ☐ -22
- ☐ 24
- ☐ 25

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

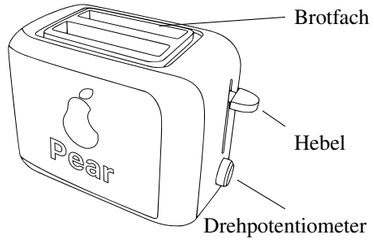
- ☐ Programme sind C-Quellcode-Dateien, die durch einen C-Compiler in einen lauffähigen Prozess übersetzt werden können.
- ☐ Prozesse können durch einen Aufruf der Funktion `exec(...)` terminiert werden.
- ☐ Ein Prozess kann mehrere Kindprogramme ausführen.
- ☐ Ein Prozess ist ein Programm in Ausführung.

i) Welche Aussage zu Semaphoren ist richtig? 2 Punkte

- ☐ Semaphore werden benutzt um in kritischen Abschnitten Interrupts zu sperren und so den gleichzeitigen Zugriff auf gemeinsame Datenstrukturen zu verhindern.
- ☐ Eine P-Operation überprüft, ob der Semaphor den Wert 0 hat und erhöht ihn anschließend.
- ☐ Semaphore können genutzt werden, um den Ablauf eines Threads mit einem Ereignis in einem anderen Thread zu synchronisieren.
- ☐ Die V-Operation dekrementiert den Semaphor um 1 und deblockiert andere in einer V-Operation blockierte Prozesse.

Aufgabe 2: Toaster (31 Punkte)

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



Eine bekannte Firma mit Obstlogo will in die zukunftsträchtige Haushaltsgeräteherstellung einsteigen, welche getreu populärer Designgrundlagen mit minimalistischen Interaktionsmöglichkeiten ausgestattet sein sollen. Das erste Produkt in diesem Segment soll ein Brotröstgerät (sog. „Toaster“) sein, welches neben einem Hebel für das Brotfach nur ein Drehpotentiometer („Touch Wheel“) zur Einstellung der Röstdauer (und damit der Bräune) als Benutzereingabe besitzt.

Durch Herabdrücken des Hebels wird das mit einem Federmechanismus ausgestattete Brotfach nach unten geschoben und intern ein Schalter gedrückt, wodurch ein Haltemagnet diese Stellung des Brotfachs beibehält. Das Heizelement wird solange aktiviert, bis die am Drehpotentiometer eingestellte Zeitdauer (20 Sekunden bis 10 Minuten) abgelaufen ist. Ein anschließendes Lösen des Haltemagnets lässt das Brotfach nach oben springen.

Auch während des Röstens kann die Zeitdauer über das Drehpotentiometer angepasst werden. Außerdem kann durch kräftiges Ziehen am Hebel während des Röstvorgangs der Haltemagnet überwunden und das Brotfach gehoben werden. Der Toaster registriert dies mittels des Schalters und beendet in diesem Fall den Röstvorgang.

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.
- Mit dem durch Drücken des Hebels aktivierten Schalter am Brotfach beginnt der Röstvorgang:
  - Haltemagnet aktivieren, damit Hebel und Brotfach an der aktuellen Position verweilen.
  - Der Röstvorgang beginnt mit dem Anschalten des Heizelements.
  - Kapselung des Startvorgangs in der Funktion `void beginRoast(void)`.
- Für die Zeittaktung soll ein 8-Bit Timer so konfiguriert werden, dass er in der Lage ist Sekunden zu zählen (siehe *Information über die Hardware*).
- Jede Sekunde soll die bereits vergangene Röstzeit mit der Röstdauereinstellung am Drehpotentiometer verglichen werden und gegebenenfalls auf Änderungen reagiert werden.
- Die aktuelle Röstdauereinstellung am Drehpotentiometer kann über die Bibliotheksfunktion `getRoastTimeSeconds()` abgerufen werden, wobei der Rückgabewert maximal `600` (Sekunden) beträgt. Das Auslesen des Potentiometers und die Umrechnung beinhalten komplexe Operationen, so dass `getRoastTimeSeconds()` nicht mit gesperrten Interrupts aufgerufen werden darf.
- Nach Ablauf der Röstzeit oder durch Ziehen des Hebels wird der Röstvorgang beendet und das Brotfach gehoben:
  - Heizelement abschalten.
  - Haltemagnet deaktivieren, damit der Federmechanismus das Brotfach heben kann.
  - Kapselung in der Funktion `void endRoast(void)`.
- Der Energiebedarf soll minimal gehalten werden:
  - Auf Polling soll so weit wie möglich verzichtet werden – der Mikrocontroller soll so viel Zeit wie möglich im Schlafmodus verbringen.
  - Der Timer-Interrupt soll nur während des Röstvorgangs aktiviert sein. Der Timer selbst muss nicht deaktiviert werden, da dies alleine kein Aufwecken der CPU auslöst.
  - Für den Timer soll ein möglichst ressourcenschonender Vorteiler gewählt werden.

Information über die Hardware

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

Haltemagnet: **PORTB**, Pin 0

- Bei anliegendem HIGH-Pegel hält der Magnet das Brotfach in der unteren Position
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRB**-Register auf 1
- Haltemagnet zunächst inaktiv, entsprechendes Bit in **PORTB**-Register auf 0

Heizelement: **PORTC**, Pin 3

- Bei anliegendem HIGH-Pegel beginnt das Heizelement Hitze zu emittieren.
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRC**-Register auf 1
- Heizelement zunächst inaktiv, entsprechendes Bit in **PORTC**-Register auf 0

Schalter am Brotfach : Interruptleitung an **PORTD**, Pin 2

- active-low: Ist der Hebel gedrückt (Brotfach heruntergefahren), liegt ein LOW-Pegel an
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren; entsprechendes Bit in **PORTD**-Register auf 1
- Externe Interruptquelle **INT0**, ISR-Vektor-Makro: **INT0_vect**
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT0**-Bits im Register **EIMSK**
- Auslesen des Zustands über entsprechendes Bit im **PIND**-Register

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

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

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

- Es soll die Überlaufunterbrechung verwendet werden (ISR-Vektor-Makro: **TIMER0_OVF_vect**)
- Der ressourcenschonendste Vorteiler (*prescaler*) ist 1024, wodurch es bei dem $16\,\mathrm{MHz}$ CPU-Takt alle $16.384\,\mathrm{ms}$ zum Überlauf des Zählers kommt (8 Bit)
- Somit entsprechen 61 Überlaufunterbrechungen etwa einer Sekunde
- Aktivierung des Überlaufinterrupts erfolgt durch Setzen des **TOIE0**-Bits im Register **TIMSK0**

Konfiguration der Frequenz des Zeitgebers **TIMER0** (Bits in 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 "roastcontrol.h"

uint16_t getRoastTimeSeconds();


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

```
// Unterbrechungsbehandlungsfunktionen
```

U:

```
// Funktion main
```



```
// Ende Funktion main
```

M:

```
// Hilfsfunktion beginRoast
```

```
// Hilfsfunktion endRoast
```

R:

```
// Initialisierungsfunktion
```

```
// Ende Initialisierungsfunktion
```

I:

Aufgabe 3: lstree (16 Punkte)

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

Schreiben Sie ein Programm `lstree`, dass alle regulären Dateien und Unterverzeichnisse eines oder mehrerer Verzeichnisse ausgibt. Die Hierarchie wird durch eine Einrückung des Namens ersichtlich, Dateinamen werden zusätzlich mit der Angabe der Dateigröße (in Bytes) ergänzt.

Ein Aufruf aus einem SPiC Übungsverzeichnis könnte zum Beispiel so aussehen:

```
$> lstree aufgabe1 aufgabe2 material
aufgabe1:
    zaehler.c (2112 Bytes)
    zaehler.elf (13052 Bytes)
aufgabe2:
    izaehler.c (4148 Bytes)
material:
    notizen.txt (2907 Bytes)
    uebungsfolien:
        u01.pdf (2413454 Bytes)
        u02.pdf (650652 Bytes)
    vorlesungsfolien:
        V_SPIC.pdf (9266730 Bytes)
```

Das Programm soll im Detail wie folgt funktionieren:

- Das Programm bekommt beliebig viele Verzeichnisse als Aufrufparameter übergeben. Für jeden Aufrufparameter soll der Verzeichnisname mit einem abschließenden Doppelpunkt auf der Standardausgabe ausgegeben werden und anschließend rekursiv die Inhalte des Verzeichnisses entsprechend eingerückt ausgegeben werden.
- Die Ausgabe der Inhalte eines Verzeichnisses soll in der Funktion

  ```
  void print_dir(unsigned int depth, const char *folder);
  ```

  implementiert werden. Datei- und Verzeichnisnamen, welche mit einem Punkt beginnen, werden ignoriert. Jede Zeile wird um `depth` Anzahl Tabulatorzeichen (`'\t'`) eingerückt.
- Für die Einrückung soll die Funktion

  ```
  void print_tabs(unsigned int num);
  ```

  implementiert werden, die genau `num` Tabs (`'\t'`) auf die Standardausgabe schreibt
- Bei regulären Dateien wird die Dateigröße (in Bytes) mit ausgegeben.
- Bei Verzeichnissen wird der Name mit einem abschließenden Doppelpunkt ausgegeben und die Funktion `print_dir` *rekursiv* aufgerufen. Dabei wird der Parameter `depth` um `1` erhöht und als `folder` der Pfad zum entsprechenden Unterverzeichnis übergeben.
- Symbolische Verknüpfungen sollen ignoriert werden.
- Tritt beim Öffnen bzw. beim Lesen eines Verzeichnisses ein Fehler auf, gibt die Funktion `print_dir` eine aussagekräftige Fehlermeldung aus und springt zum Aufrufer zurück. Tritt während der Verarbeitung eines Verzeichniseintrags ein Fehler auf, gibt die Funktion ebenfalls eine Fehlermeldung aus und fährt mit dem nächsten Verzeichniseintrag fort. Fehler, die nicht zum Abbruch des Programms führen, sollen auch keinen Einfluss auf den Exitcode des Programms haben.
- Um eine sortierte Ausgabe (wie im Beispiel) muss sich **nicht** gekümmert werden.

Hinweis: Achten Sie auf eine korrekte Fehlerbehandlung der verwendeten Funktionen. Fehlermeldungen sollen generell auf `stderr` erfolgen.

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

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

static void print_tabs(unsigned int num);
static void print_dir (unsigned int depth, const char *folder);

// Funktion main
```

```
// Funktion print_tabs
```

P:


```
// Funktion print_dir

  // Öffnen des Verzeichnisses

  // Iteration über die Einträge

  // Pfadzeichenkette zusammenbauen
```

```
    // Lesen der Dateiattribute

    // Namen ausgeben und ggf. print_dir aufrufen (Rekursion)

  // Aufräumen

// Ende der Funktion print_dir
```

T:

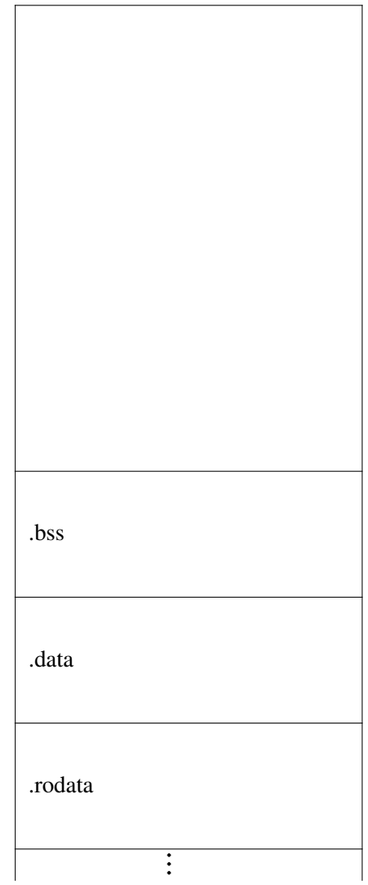
Aufgabe 4: Speicherorganisation (15 Punkte)

Das folgende Programm wird ohne Optimierungen übersetzt und auf einer 64-Bit-Architektur (mit 64-bit Speicheradressen) ausgeführt.

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

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  #define x 5
5
6  const int n = x;
7  char a[x] = {'S', 'P', 'i', 'C', '!' };
8
9  static int i = 0;
10
11 static void swap(void) {
12   char *v = a + i--;
13   char t = *v;
14   *v = a[i];
15   a[i] = t;
16 }
17
18 int sort(void){
19   static int c = 0;
20   for(i = 0 ; i < n ; i++)
21     while (i > 0 && a[i] < a[i-1] && ++c)
22       swap();
23   return c;
24 }
25
26 int main(void){
27   printf("%.*s␣(%d)\n", n, a, sort());
28   return 0;
29 }
```

Speicheraufbau (vereinfacht):



a) Vervollständigen Sie den (vereinfachten) Speicheraufbau:

5 Punkte

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

2) Ordnen Sie alle im Quelltext vorkommenden Variablen den dargestellten Speichersektionen zu. (3 Punkte)

b) Welche Speichergröße (in Bytes) wird für folgende Variablen auf der oben genannten 64-Bit-Architektur (mindestens) benötigt? (2 Punkte)

Variable	**Speichergröße (in Bytes)**
`char *v` (Zeile 12)	
`char t` (Zeile 13)	

c) Wann geschieht die statische und wann die dynamische Allokation? Ordnen Sie die genannten Speichersektionen entsprechend zu. Welcher Zusammenhang besteht dabei bezüglich der Lebensdauer der Variablen? (6 Punkte)

d) Das Schlüsselwort `static` kann unterschiedliche Bedeutungen haben. Erläutern Sie, welche Auswirkung das Schlüsselwort jeweils auf die Variablen in Zeile 9 und Zeile 19 hat. (2 Punkte)

Aufgabe 5: Betriebssystemkonzepte (10 Punkte)

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

a) Erläutern Sie kurz den Ablauf eines Systemaufrufs (*System Call*). Worin besteht der Unterschied zu einem *normalen* Funktionsaufruf? (4 Punkte)

b) Erläutern Sie kurz den Unterschied zwischen einem Verweis (*Hard Link*) und einem symbolischen Verweis (*Symbolic Link*) auf eine Datei. Können die unterschiedlichen Verweistypen auf nicht (mehr) existierende Dateien zeigen? (3 Punkte)

c) Beschreiben Sie den Unterschied zwischen einem **Spin Lock** und einem **Sleeping Lock**. (3 Punkte)