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, kreisen Sie bitte die falsche Antwort ein 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 ist eine Softwarekomponente, welche Java-Klassen durch C-Funktionen ersetzt, die dann von einem C-Compiler übersetzt werden.
- ❏ 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.

b) 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.

c) 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.

d) Welche Aussage zu Zeigern ist **richtig**? 2 Punkte

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

e) Was versteht man unter Polling? 2 Punkte

- ❏ Ein Konzept zur Abarbeitung von Interrupts.
- ❏ Das regelmäßige Anheben eines Pegels, um einem Gerät einen bestimmten Zustand zu signalisieren.
- ❏ Wenn ein Programm regelmäßig eine Peripherie-Schnittstelle überprüft, ob Daten oder Zustandsänderungen vorliegen.
- ❏ Wenn ein Programm zum Zugriff auf kritische Daten Interrupts sperrt.

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

- ❏ Programme sind Anwendungen des Benutzers, während Prozesse Aktivitäten des Betriebssystems sind.
- ❏ Ein Prozess hat einen eigenen virtuellen Adressraum. Daten des Prozesses sind vor direktem Zugriff durch andere Prozesse geschützt.
- ❏ Programme sind C-Quellcode-Dateien, die durch einen C-Compiler in einen lauffähigen Prozess übersetzt werden können.
- ❏ Ein Programm ist ein Prozess in Ausführung.

g) Welche Aussage zu Semaphoren ist richtig?

- ❏ 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.
- ❏ Semaphoren können genutzt werden, um gegenseitiges Warten zwischen Threads zu implementieren.
- ❏ Die V-Operation dekrementiert den Semaphor um 1 und deblockiert andere in einer V-Operation blockierte Prozesse.

h) Welche Aussage zu Threads ist richtig?

- ❏ User-Level-Threads sind ideal für Multiprozessoren, weil sie sehr einfach die Nutzung aller Prozessoren erlauben.
- ❏ Threads können nur auf Multiprozessorsystemen verwendet werden.
- ❏ Die Umschaltung zwischen Kernel-Level-Threads ist effizienter als die Umschaltung zwischen User-Level-Threads.
- ❏ Die Umschaltung zwischen User-Level-Threads ist besonders effizient.

Aufgabe 2a: Tankkontrolle (30 Punkte)

Schreiben Sie ein AVR-Mikrokontroller-Programm, das den Füllstand eines Tanks mit einem Fassungsvermögen von 50000 Litern überwacht und insbesondere einen Überlauf verhindert.

Befüllung: Der Zufluss ist mit einer Zuflusssperre und einem Durchflusssensor ausgestattet. Der Sensor signalisiert jeden durchgeflossenen Liter durch einen Interrupt. Der Zufluss kann durch Aktivierung der Sperre gestoppt werden.

Entnahme: Der Abfluss ist durch einen manuellen Absperrhahn geregelt. Öffnen und Schliessen des Hahns werden durch Interrupt an den Controller gemeldet. Ein Durchflussensor signalisiert jeden abgeflossenen Liter durch Zustandswechsel $0 \leftrightarrow 1$ (Pollen!).

Das Programm soll im Einzelnen wie folgt funktionieren:

- Zum Programmstart ist der Tank leer und der Abfluss ist gesperrt.
- Die main-Funktion ruft zunächst die Funktion `void init();` auf, welche die Initialisierung der I/O-Ports und Interruptquellen durchführt. Hierbei dürfen keine Annahmen über den initialen Zustand der Register gemacht werden. Am Ende der Initialisierung wird der Zufluss geöffnet.
- Stellt die Steuerung fest, dass der Tank voll ist, wird die Zuflusssperre aktiviert. Sinkt die Tankfüllung unter 20000 Liter, wird wieder aufgefüllt.
- Während eines Abflusses überwacht die Steuerung die abfliessende Wassermenge durch Pollen des Abflusssensors.
- Die Steuerung soll den Prozessor in den Standardstromsparmodus versetzen, wenn gerade nichts zu tun ist.

Information über die Hardware

Zuflusssensor: **PORTD**, **Pin 2**
- steigende Flanke pro einfließendem Liter Wasser
- externe Interruptquelle **0**, ISR-Vektor-Makro: **INT0_vect**
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT0**-Bits im Register **GICR**

Zuflusssperre: **PORTD**, **Pin 0**
- 1 = offen, 0 = gesperrt

Abflusshahn: **PORTD**, **Pin 3**
- 1 = offen, 0 = geschlossen (=> steigende Flanke = öffnen, fallende = schließen)
- externe Interruptquelle **1**, ISR-Vektor-Makro: **INT1_vect**
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT1**-Bits im Register **GICR**

Abflusssensor: **PORTD**, **Pin 1**
- Zustandswechsel = 1 Liter Durchfluss
- Zustand kann durch Lesen von **Bit 1** des Registers **PIND** abgefragt werden

Eingänge, Ausgänge, Externe Interruptquellen **0** bzw. **1** konfigurieren:
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Reg. auf 0
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRD**-Reg. auf 1
- angeschlossene Sensoren verbinden den Pin mit Masse, es muss der interne Pullup-Widerstand verwendet werden (entspr. Bit in **PORTD**-Reg. auf 1 setzen).
- Konfiguration der externen Interruptquellen (Bits in Register **MCUCR**):

externe Int.quelle 0		**externe Int.quelle 1**		**Beschreibung**
ISC01	**ISC00**	**ISC11**	**ISC10**	
0	0	0	0	Interrupt bei low Pegel
0	1	0	1	Interrupt bei beliebiger Flanke
1	0	1	0	Interrupt bei fallender Flanke
1	1	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>

#define FILL_MAX 50000
#define FILL_MIN 20000

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

[ ]

```
/* Unterbrechungsbehandlungsfunktionen */
```

[ ]

[ ]

I:


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

[ ]

```
  /* Initialisierung */
```

[ ]

```
  /* Warten auf Ereignisse */
```

[ ]

```
  /* Abfluss überwachen */
```

```
/* Ende der Funktion main */
```

M:


```
/* Funktion init
   (teilweise vorgegeben, um Schreibaufwand zu sparen)*/

     DDRD  &= ~(1 << PD1); /* Port1 als Eingang */
     DDRD  &= ~(1 << PD2); /* Port2 als Eingang */
     DDRD  &= ~(1 << PD3); /* Port3 als Eingang */

     PORTD |= (1 << PD1); /* Pullup aktivieren */
     PORTD |= (1 << PD2); /* Pullup aktivieren */
     PORTD |= (1 << PD3); /* Pullup aktivieren */
```

N:

Aufgabe 2b: (15 Punkte)

Schreiben Sie ein Programm, das die Anzahl der regulären und sonstigen (= "nicht regulären") Dateien in einem Verzeichnis ermittelt und die Größen (Zahl der Bytes) der regulären Dateien aufsummiert (=Gesamtspeicherbedarf).

Alle Fehlermeldungen sollen auf den Standardfehlerkanal stderr ausgegeben werden, nicht auf die Standardausgabe.

Das Programm soll wie folgt funktionieren:

- Das zu durchsuchende Verzeichnis wird als erstes Argument an das Programm übergeben. Wird das Programm ohne Argument oder mit zu vielen Argumenten aufgerufen, wird eine Fehlermeldung ausgegegeben.
- Es wird das angegebene Verzeichnis durchsucht. Verzeichniseinträge, deren Dateiname mit einem Punkt ’.’ beginnt werden ignoriert. Die Zahl der regulären Dateien und der sonstigen Dateien wird bei dem Durchlauf ermittelt. Ausserdem werden die Dateigrößen aufsummiert.
- Am Ende gibt das Programm auf der Standardausgabe folgende Meldung aus:
  `Directory AAA:`
  `a reguläre Dateien, b sonstige Dateien, Gesamtspeicherbedarf: c`
  (für AAA ist der Name des angegebenen Verzeichnisses einzusetzen, für a, b und c die ermittelten Zahlenwerte)

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

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


/* Funktion main */

----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------


/* Verzeichnis oeffnen */

----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
```

```
  /* Verzeichnis durchlaufen */
```

```
  /* Ergebnis ausgeben */
```

```
  /* Ressourcen freigeben */
```

```
/* Ende der Funktion main */
```

B:

Aufgabe 3: (18 Punkte)

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

a) Bei einem C-Programm werden die Speicherklassen *static* und *automatic* unterschieden. Welche Möglichkeiten gibt es, Variablen dieser Speicherklassen zu definieren. (4 Punkte)

b) Skizzieren Sie die Speicherorganisation eines Prozesses. In welchen Bereichen des Speichers werden welche Daten bzw. welche Arten von Variablen abgelegt? (8 Punkte)

c) Skizzieren Sie die Schritte, die durchzuführen sind, um aus einem C-Quellprogramm ein lauffähiges Programm auf einem Mikrocontroller zu erhalten.
(6 Punkte)

Aufgabe 4: (11 Punkte)

Bei der Verwaltung von Prozessen unterscheidet das Betriebssystem vor allem fünf verschiedene Zustände. Beschreiben Sie diese Zustände und die Situationen, die zu Übergängen zwischen den Zuständen führen (ggf. mit einer Skizze illustrieren).