Aufgabe 1: (14 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 über den C-Präprozessor ist richtig? 2 Punkte

- ☐ 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.
- ☐ Der Präprozessor optimiert Makros durch Zeigerarithmetik.
- ☐ Die Syntax von Präprozessoranweisungen ist unabhängig vom Rest der Sprache C.

b) Welche Aussage zur Speicherallokation ist richtig? 2 Punkte

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

c) Welchen Wert hat die Variable `a` nach Ausführung der folgenden Zeilen Code: 2 Punkte

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

- ☐ 6
- ☐ 0
- ☐ 2
- ☐ 4

d) Gegeben ist folgendes Makro: 2 Punkte

```
#define SQ(x) (x * x)
```

Wie ist das Ergebnis des folgenden Ausdrucks

```
2 * SQ(1 - 3)
```

- ☐ 2
- ☐ -4
- ☐ -10
- ☐ 8

e) Welche Aussage zu Zeigern ist richtig? 2 Punkte

- ☐ 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.
- ☐ Beim Rechnen mit Zeigern muss immer der Typ des Zeigers beachtet werden.
- ☐ Ein Zeiger kann zur Manipulation von schreibgeschützten Datenbereichen verwendet werden.

f) Was versteht man unter Nebenläufigkeit? 2 Punkte

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

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

- ☐ Programme sind Anwendungen der Benutzer, während Prozesse Aktivitäten des Betriebssystems sind.
- ☐ Jedes Programm kann nur einmal gleichzeitig ausgeführt werden.
- ☐ Ein Prozess hat einen eigenen virtuellen Adressraum. Daten des Prozesses sind vor direktem Zugriff durch andere Prozesse geschützt.
- ☐ Ein Programm ist ein Prozess in Ausführung.

Aufgabe 2: Lüft (30 Punkte)

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

Implementieren Sie ein Warnsystem zur Überwachung der Raumbelüftung und Luftqualität (*Lüft*). Zur Verringerung der Virenlast und Garantie der Luftqualität in Prüfungs- und Unterrichtsräumen wird die Raumluft über einen Schlechtluftsensor kontrolliert. Wird eine zu geringe Luftqualität festgestellt, soll eine Warnleuchte aktiviert werden, die an das Öffnen der Fenster zum Stoßlüften erinnert. Diese soll erst dann wieder erlöschen, wenn alle Fenster für mindestens drei Minuten am Stück geöffnet waren. Um sicherzustellen, dass tatsächlich gelüftet wird, sollen die aktuellen Öffnungswinkel der Fenster periodisch überwacht und das Lüftintervall und damit die Warnung bei nicht vollständig geöffneten Fenstern entsprechend verlängert werden.

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.
- Für die Zeittaktung soll ein 8-Bit Timer verwendet werden. Konfigurieren Sie diesen so, dass er alle $T = 100ms$ einen Interrupt auslöst.
- Der Eingang `PD2` (Interrupt 0) ist mit dem Schlechtluftsensor verbunden. Die externe Beschaltung stellt sicher, dass genau dann eine steigende Flanke auftritt, wenn der Sensor eine Unterschreitung der gewünschten Luftqualität nach vorheriger Lüftung registriert. Sie dürfen davon ausgehen, dass die Luftqualität zu Programmstart ausreichend ist. Sie dürfen weiterhin davon ausgehen, dass der Sensor während des Lüftens keine Interrupts auslöst.
- Bei Ausschlag des Schlechtluftsensors soll die Warnleuchte, welche am Ausgang `PB1` angeschlossen ist, so lange aktiviert werden, bis alle Fenster gleichzeitig für mindestens das Lüftintervall (`AIRTIME`) geöffnet wurden.
- Werden ein oder mehrere Fenster vor Erlöschen der Warnleuchte geschlossen, soll das Lüftintervall von neuem beginnen und die Leuchte aktiv bleiben.
- Die verfügbaren Fenster sind mit einer bei 0 beginnenden Fenster-ID durchnummeriert. Die Anzahl ist in der Konstante `WINDOWS` gespeichert.
- Implementieren Sie die Funktion `WindowPos window_state(void)`, die zurückgeben soll, ob **alle** Fenster geöffnet (`W_OPEN`) oder geschlossen (`W_CLOSED`) sind. Befinden sich nicht alle Fenster im gleichen Zustand, soll `W_UNCLEAR` zurückgegeben werden.
- Um festzustellen, ob ein Fenster vollständig geöffnet oder geschlossen ist, lässt sich sein Öffnungswinkel per 10-Bit-ADC auslesen (`int16_t sb_adc_read(ADCDEV)`). Die entsprechende Geräte-ID für das jeweilige Fenster erhalten Sie durch Aufruf der vorgegebene Funktion `ADCDEV window_to_adcdev(uint8_t wid)`. Ein Fenster gilt als geschlossen, wenn der ADC einen Wert kleiner oder gleich `W_ANGLE_CLOSED` zurückliefert. Stellen Sie sicher, dass die Interrupts während der Abfrage der ADC-Werte gesperrt sind.
- Fragen Sie die Öffnungswinkel der Fenster während des Lüftvorgangs alle 100ms ab.
- Stellen Sie sicher, dass sich der Mikrocontroller möglichst oft im Schlafmodus befindet.

Information über die Hardware

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

Schlechtluftsensor: Interruptleitung an **`PORTD`**, Pin 2

- active-high: Schlägt der Sensor an, wechselt der Pegel von LOW zu HIGH
- Pin als Eingang konfigurieren: Entsprechendes Bit im **`DDRD`**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren: Entsprechendes Bit im **`PORTD`**-Register auf 1
- Externe Interruptquelle **`INT0`**, ISR-Vektor-Makro: **`INT0_vect`**
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **`INT0`**-Bits im Register **`EIMSK`**

Konfiguration der externen Interruptquelle **`INT0`** (Bits im Register **`EICRA`**)

Interrupt 0		Beschreibung
ISC01	ISC00	
0	0	Interrupt bei low Pegel
0	1	Interrupt bei beliebiger Flanke
1	0	Interrupt bei fallender Flanke
1	1	Interrupt bei steigender Flanke

Warnleuchte: Ausgang an **`PORTB`**, Pin 1

- Bei anliegendem LOW-Pegel leuchtet die Warnleuchte
- Pin als Ausgang konfigurieren: Entsprechendes Bit im **`DDRB`**-Register auf 1
- Warnleuchte zunächst aus, entsprechendes Bit im **`PORTB`**-Register auf 1

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 2,6 MHz CPU-Takt (hinreichend genau) alle $100ms$ zum Überlauf des 8-bit-Zählers **`TCNT0`** kommt.
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **`TOIE0`**-Bits im Register **`TIMSK0`**

Konfiguration der Frequenz des Zeitgebers **`TIMER0`** (Bits im 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/interrupt.h>
#include <avr/io.h>
#include <avr/sleep.h>
#include <stdint.h>
#include <adc.h>

typedef enum {
  W_UNCLEAR,
  W_OPEN,
  W_CLOSED
} WindowPos;

static const uint8_t WINDOWS = 5;
static const uint8_t AIRTIME = 3; // in minutes

static const int16_t W_ANGLE_CLOSED = 603;

extern ADCDEV window_to_adcdev(uint8_t);

// Funktionsdeklarationen, globale Variablen, etc.

// Unterbrechungsbehandlungsfunktionen

// Ende Unterbrechungsbehandlungsfunktionen
```

D:


```
// Funktion main

  // Initialisierung und lokale Variablen

  // Hauptschleife

    // Ereignisse verarbeiten
```

```
// Ende main
```

M:


```
// Fensterstatusüberprüfungsfunktion
```

```
// Ende Fensterstatusüberprüfungsfunktion
```

F:

```
// Initialisierungsfunktion
```

```
// Ende Initialisierungsfunktion
```

I:

Aufgabe 3: cargo (19 Punkte)

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

Schreiben Sie das Programm `cargo` (**c**opy **arg**uments **o**ver), welches ein weiteres Programm mehrfach mit verschiedenen Argumenten aufruft, die zeilenweise von `stdin` gelesen werden.

```
$> # Inhalt der Datei
$> # arguments.txt ausgeben
$> cat arguments.txt
Test123
Hallo Welt
```

```
$> # cargo echo starten und arguments.txt
$> # nach stdin weiterleiten
$> ./cargo echo < arguments.txt
Test123
Hallo Welt
```

Das Programm soll im Detail wie folgt funktionieren:

– Implementieren Sie zuerst die Hilfsfunktion `int run(char *prog, char *line)`, welche wie folgt funktionieren soll:

  - Für jeden Aufruf von `run()` soll genau eine Programmausführung stattfinden.
  - Das auszuführende Programm wird in `prog` übergeben, die Argumente sind in der übergebenen Zeile `line` enthalten. Sie sind durch Leerzeichen oder Tabulatoren getrennt. Für leere Eingabezeilen soll das Programm ohne Argumente ausgeführt werden.
  - `run()` soll zunächst aus `line` und `prog` ein Array konstruieren, welches an `exec()` übergeben werden kann. Es soll **nicht** die gesamte Zeile an `exec()` übergeben werden!
  - Im Anschluss daran soll ein Kindprozess erzeugt werden, welcher den Aufruf ausführt.
  - Im Vaterprozess soll auf die Beendigung des Kindes gewartet werden. Beendet sich dieses mit Exitcode `0`, gibt `run()` ebenfalls den Wert `0` zurück, im Fehlerfall oder bei anderen Terminierungsgründen einen anderen Wert.
  - Die Hilfsfunktion soll den Vaterprozess nicht beenden.

– In der Funktion `main()` soll `cargo` zunächst prüfen, ob genau ein Kommandozeilenparameter übergeben wurden. Ist dies nicht der Fall, soll das Programm eine entsprechende Fehlermeldung ausgeben und sich beenden.

– Im Anschluss soll das Programm zeilenweise von `stdin` lesen.

– Für jede Zeile soll ein Aufruf des als Parameter übergebenen Programms erfolgen:

  - Die eigentliche Programmausführung soll in der Hilfsfunktion `run()` erfolgen, welche mit dem auszuführenden Programm und der eingelesenen Zeile aufgerufen werden soll.
  - Im Falle eines fehlgeschlagenen Aufrufs soll `cargo` sich nicht beenden, sondern mit der nächsten Zeile fortfahren und erst am Ende `EXIT_FAILURE` zurückgeben.

– Können keine Zeichen mehr gelesen werden, soll sich `cargo` mit einem passenden Exitcode beenden.

Hinweise:

– Sie dürfen davon ausgehen, dass die einzelnen Zeilen nicht länger als `1024` Zeichen sind.

Achten Sie auf eine korrekte Fehlerbehandlung der verwendeten Funktionen. Fehlermeldungen sollen generell auf `stderr` erfolgen. Zur kompakten Fehlerbehandlung können die vorgegebenen Funktionen `die()` (`errno` gesetzt) und `err()` (`errno` nicht gesetzt) genutzt werden.

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

```c
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LINE_MAX 1024

static void die(const char message[]) {
    perror(message);
    exit(EXIT_FAILURE);
}

static void err(const char message[]) {
    fprintf(stderr, "%s\n", message);
    exit(EXIT_FAILURE);
}

// Funktion run()

// Argumente vorbereiten
```



```c
// Aufruf in Kindprozess ausführen

// Ende run()
```

R:

```
// Funktion main

    // Parameter prüfen

    // Zeilenweise von stdin lesen

    // Fehlerbehandlung

// Ende main()
```

L:

Aufgabe 4: Unterbrechungen (9 Punkte)

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

a) Welche Schritte umfasst typischerweise die Abarbeitung eines Interrupts auf einem Mikrokontroller? (3 Punkte)

b) Warum sollten Interruptbehandlungen und Bereiche mit Interruptsperren möglichst kurz gehalten werden? (1 Punkt)

c) Welches Problem kann in folgendem Programmfragment auftreten, das analoge Ereignisse als digitales Signal aufzeichnet (`record`) und per Interrupt (`INT0`) über das Auftreten des Ereignisses informiert wird? Nennen Sie das Problem und markieren Sie die entsprechenden Zeilen. Skizzieren Sie außerdem einen konkreten Ablauf. *(Angabe der Reihenfolge, in der Codezeilen ausgeführt werden, genügt. Beispielantwort für einen Durchlauf der äußeren Schleife ohne Unterbrechungen und timer != 0: M8, M13-17, M6 → unproblematischer Durchlauf* (3 Punkte)

```
M1  // Records a high or low bit
M2  void record(uint8_t bit);
M3
M4  void main (void) {
M5    sleep_enable();
M6    while(1) {
M7      cli();
M8      while(timer == 0) {
M9        sei();
M10       sleep_cpu();
M11       cli();
M12     }
M13     timer = 0;
M14     sei();
M15
M16     record(high);
M17     high = 0;
M18   }
M19 }
```

```
I1  static volatile uint8_t timer = 0;
I2  static volatile uint8_t high = 0;
I3
I4  // Kommt alle 100ms
I5  ISR(TIMER0_OVF_vect) {
I6    timer = 1;
I7  }
I8
I9  ISR(INT0_vect) {
I10   high = 1
I11 }
```

d) Warum ist die Verwendung von Interrupts trotz solcher Problematiken, wie sie in Teilaufgabe c) auftreten, sinnvoll? (2 Punkte)

Aufgabe 5: C als Systemprogrammiersprache (9 Punkte)

Stellen Sie sich ein einfaches Hilfsmodul `limit` vor, das Ereignisse zählt und bei Überschreiten eines Grenzwerts eine Ausnahmebehandlung aufruft. Hierzu kann bei der Initialisierung des Moduls (`void init(uint8_t limit, callback_t cb)`) der Grenzwert und die Ausnahmebehandlungsfunktion gesetzt werden. Über eine Benachrichtigungsfunktion (`void notify(void)`) kann das Modul über das Auftreten exakt eines Ereignisses informiert werden; wird hierbei der Grenzwert überschritten, wird die Ausnahmebehandlungsfunktion aufgerufen. Die öffentliche Schnittstelle des Moduls `limit` umfasst nur die beiden genannten Funktionen.

a) Nehmen Sie an, `callback_t` wäre wie folgt definiert:

`typedef void(*callback_t)(uint8_t, uint8_t);`.

Welcher Typ verbirgt sich hinter `callback_t`? (2 Punkte)

b) Mit welchem Schlüsselwort kann in C die Sichtbarkeit von Variablen und Funktionen beschränkt werden? (1 Punkt)

c) Beschreiben Sie in knappen Stichpunkten, welche Deklarationen und Definitionen von Funktionen und Typen in den Dateien des limit-Moduls (`limit.c`, `limit.h`) mit welcher Sichtbarkeit vorhanden sein müssen, um eine korrekte Implementierung des Moduls zu gewährleisten. (3 Punkte)

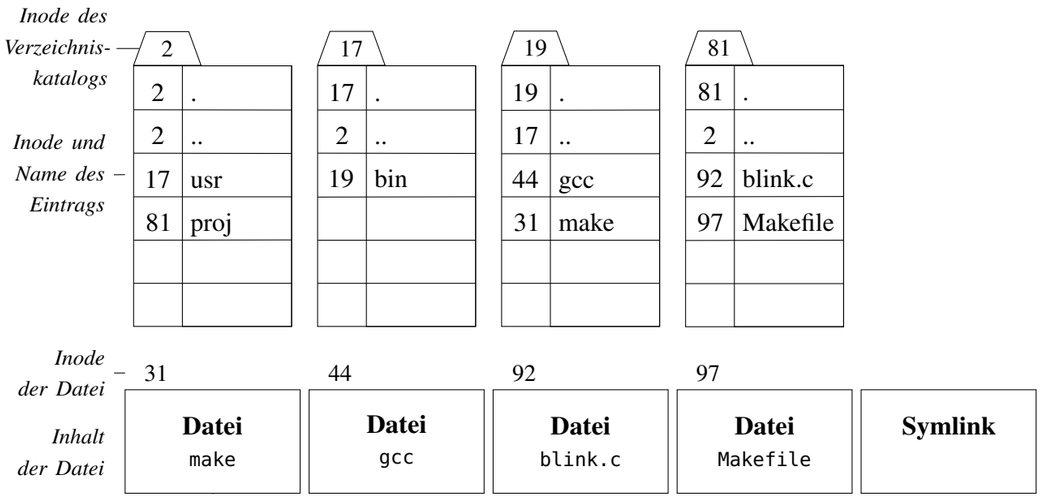
`limit.h:`

`limit.c:`

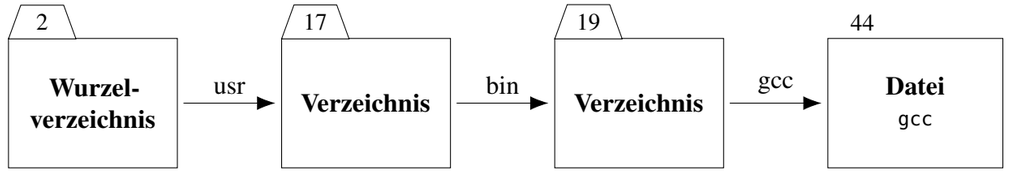
d) Nennen Sie drei Gründe warum man Software in der Programmiersprache C in Module gliedert: (3 Punkte)

Aufgabe 6: Dateisystem (9 Punkte)

Ein Dateisystem ermöglicht das strukturierte Ablegen von Daten. Nachfolgend ist ein beispielhafter und vereinfachter Ausschnitt der hierfür in Linux benötigten Verwaltungsinformationen abgebildet.



a) Vervollständigen Sie den dazugehörigen Verzeichnisbaum. Orientieren Sie sich dabei an dem bereits vorgegebenen Schema: Verzeichnisse und Dateien werden mit einem beschrifteten Rechteck gekennzeichnet, ein Verweis mit einem beschrifteten Pfeil. Das Einzeichnen der `.` und `..` Verweise ist **nicht** erforderlich. (4 Punkte)



b) Nun soll ein symbolischer Verweis (*Symlink*) `/bin` erstellt werden, welcher auf den Pfad `/usr/bin` zeigt. Vervollständigen Sie dazu die oben in der Angabe vorgegebenen Verwaltungsinformationen. Eine Anpassung des Verzeichnisbaums aus Teilaufgabe a) ist **nicht** erforderlich. (2 Punkte)

c) Erläutern Sie kurz, wie in einem Standard-UNIX-Dateisystem mehrere reguläre Verweise (*Hard Links*) auf die gleiche Datei zeigen können. (1 Punkt)

d) Nennen Sie drei verschiedene Dateiattribute, welche in Standard-UNIX-Dateisystemen zusätzlichen zum Inhalt jeder Datei gespeichert werden. Wie heißt die Datenstruktur, in der diese Metadaten gespeichert werden? (2 Punkte)