Aufgabe 1: (12 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) Wann kann es zu Nebenläufigkeitsproblemen kommen? 2 Punkte

- ☐ Wenn die Programmabschnitte im `if`- und `else`-Teil einer bedingten Anweisung auf dieselbe Variable zugreifen.
- ☐ Wenn ein Programmabschnitt in einer Schleife mehrfach durchlaufen wird.
- ☐ Wenn ein Programm in einer Funktion mehrere lokale Variablen verwendet.
- ☐ Wenn aus dem Hauptprogramm und einer Unterbrechungsbehandlungsfunktion dieselbe globale Variable geschrieben wird.

b) Welche Aussage zu folgender Funktion ist richtig? 2 Punkte

```
uint16_t *func(void) {
    static uint16_t var = 42;
    --var;
    return &var;
}
```

- ☐ Die Variable `var` enthält beim Verlassen der `func()`-Funktion immer den Wert `41`, da `var` bei jedem Aufruf von `func()` mit `42` initialisiert wird.
- ☐ Die Variable `var` ist über die Laufzeit der `func()`-Funktion hinaus gültig. Daher kann der zurückgelieferte Zeiger sicher vom Aufrufer verwendet werden.
- ☐ Die Funktion liefert einen Zeiger auf die lokale Variable `var` zurück. Dies ist in C nicht zulässig und führt zu einem Übersetzungsfehler.
- ☐ Beim Verlassen der Funktion `func()` wird die `automatic`-Variable `var` vom Stack entfernt und der Zeiger verliert seine Gültigkeit. Ein Zugriff durch den Aufrufer führt zu zufälligen Ergebnissen.

c) Welche der folgenden Aussagen zum Linken eines Programms ist **richtig**? 2 Punkte

- ☐ Beim Linken werden Zugriffe aus einem C-Modul auf globale **`static`**-Variablen eines anderen C-Moduls optimiert.
- ☐ Der Linker fügt Objektdateien und Bibliotheken zu einer ausführbaren Datei zusammen.
- ☐ Der Linker löst **`#include`**-Anweisungen in einem C-Programm auf, indem er sie durch den Inhalt der einzubindenden Datei ersetzt.
- ☐ Auf einer Mikrocontroller-Plattform wird normalerweise dynamisch gelinkt, weil dies Speicherplatz spart.

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

- ☐ Wurde gerade ein Flanken-gesteuerter Interrupt ausgelöst, so muss erst ein Pegelwechsel der Interruptleitung stattfinden, damit erneut ein Interrupt ausgelöst werden kann.
- ☐ Interrupts sind eine Besonderheit von AVR-Mikroprozessoren. Auf anderen Architekturen müssen externe Ereignisse durch Pollen abgefragt werden.
- ☐ Pegelgesteuerte Interrupts müssen durch Pollen des Pegels abgefragt werden.
- ☐ Pegelgesteuerte Interrupts werden bei jedem Wechsel des Pegels ausgelöst.

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

```
uint8_t i = 0x21;
i <<= 4;
```

- ☐ 10
- ☐ 16
- ☐ 0x25
- ☐ 0x84

f) Gegeben ist folgendes Makro: 2 Punkte

```
#define MAX(x,y) (x > y) ? (x) : (y)
```

Wie ist das Ergebnis des folgenden Ausdrucks

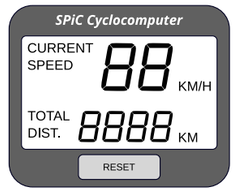
```
MAX(5, 3) * 2
```

- ☐ 3
- ☐ 5
- ☐ 6
- ☐ 10

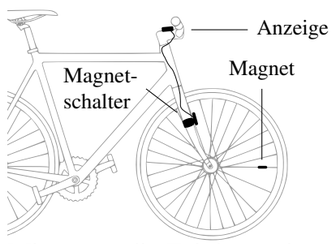
Aufgabe 2: Fahrradcomputer (30 Punkte)

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

Erlangen gilt als eine der fahrradfreundlichsten Städte Deutschlands. Um die eigene Fahrleistung zu messen, bietet sich ein Fahrradcomputer an. Schreiben Sie ein Programm für den AVR-Mikrocontroller, welches die aktuelle Geschwindigkeit und die zurückgelegte Strecke misst. Durch einen Druck auf die *RESET*-Taste soll die zurückgelegte Strecke wieder gelöscht (auf 0 zurück gesetzt) werden.



Prinzip:
Ein am Vorderrad in den Speichen befestigter Magnet löst beim Passieren der Gabel den dort befestigten Magnetschalter aus – einmal pro voller Radumdrehung.



Die zurückgelegte Distanz pro Radumdrehung ist abhängig von der Größe des Reifen. Für 27 Zoll Reifen beträgt er zum Beispiel 215 cm. Wurden nun beispielsweise $n = 500$ Umdrehungen erkannt, so beträgt die zurückgelegte Distanz

$$s = n \cdot U = 500 \cdot 215\,\mathrm{cm} = 107\,500\,\mathrm{cm} \approx 1\,\mathrm{km}$$

(*Zur Erinnerung:* 100 000 cm entsprechen 1 km)

Misst man die Zeitdauer einer Radumdrehung, kann die momentane Geschwindigkeit ermittelt werden, z.B. bei 400 ms für eine Umdrehung ($n = 1$)

$$v = \frac{n \cdot U}{t} = \frac{1 \cdot 215\,\mathrm{cm}}{400\,\mathrm{ms}} = 36 \cdot \frac{215\,\mathrm{km}}{400\,\mathrm{h}} \approx 19\,{}^{\mathrm{km}}/_{\mathrm{h}}$$

(*Hinweis:* die Geschwindigkeit in ${}^{\mathrm{cm}}/_{\mathrm{ms}}$ kann durch Multiplikation mit 36 in ${}^{\mathrm{km}}/_{\mathrm{h}}$ umgerechnet 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.
- Ein 8-Bit Timer soll so konfiguriert werden, dass im Millisekundentakt ein Interrupt ausgelöst wird. Dieser kann benutzt werden, um die Dauer einer Radumdrehung zu messen.
- Das Display soll die momentane Geschwindigkeit (in ${}^{\mathrm{km}}/_{\mathrm{h}}$) und die zurückgelegte Distanz (in km) anzeigen.
  - Das Display wird einmalig mit der Bibliotheksfunktion **`void display_enable(void)`** aktiviert (es zeigt noch keine sinnvollen Werte an).
  - Mit der Bibliotheksfunktion **`void display_show(field, uint16_t)`** können Werte auf der Anzeige aktualisiert werden. Der erste Parameter gibt dabei das entsprechende Feld an, `SPEED` für die Geschwindigkeit und `DISTANCE` für den Kilometerzähler.
  - Da die Kommunikation mit dem Display einige Zeit dauern kann, sollen die Bibliotheksfunktionen **nicht im Interruptkontext** aufgerufen werden und der Kilometerzähler nur aktualisiert werden, wenn sich der Wert geändert hat.
  - Die Anzeige für die Geschwindigkeit erfolgt als Ganzzahl (abgerundet) und soll nur zwei Stellen (00–99) berücksichtigen. Eine Fehlerbehandlung für höhere Geschwindigkeiten oder Standzeiten ist nicht erforderlich.
  - Die Distanz wird ebenfalls als abgerundete Ganzzahl angezeigt, sie soll beim Erreichen von 10 000 km wieder auf 0 km zurück gesetzt werden.
- Ein Druck der *RESET*-Taste setzt (nur) den Kilometerzähler wieder auf `0` zurück.
- Um die Laufzeit des Fahrradcomputers zu maximieren, soll auf Polling verzichtet werden und der Mikrocontroller so viel Zeit wie möglich im Schlafmodus verbringen.

Information über die Hardware

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

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

- Es soll ungefähr jede Millisekunde eine Überlaufunterbrechung des Zähler ausgelöst werden (ISR-Vektor-Makro: **`TIMER0_OVF_vect`**)
- Dazu soll ein Vorteiler (*prescaler*) von 64 verwendet werden. Bei dem 16 MHz CPU-Takt kommt es dadurch alle 1.024 ms zum Überlauf des 8-bit-Zählers **`TCNT0`** (die Ungenauigkeit von 0.024 ms darf ignoriert werden)
- Aktivierung der Interruptquelle 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)

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

- Passiert der Speichenmagnet den Schalter, so liegt kurzzeitig ein HIGH-Pegel an
- Pin als Eingang konfigurieren: entsprechendes Bit in **`DDRD`**-Register auf 0
- Internen Pull-Up-Widerstand deaktivieren; entsprechendes Bit in **`PORTD`**-Register auf 0
- Externe Interruptquelle **`INT0`**, ISR-Vektor-Makro: **`INT0_vect`**
- Aktivierung der Interruptquelle erfolgt durch Setzen des **`INT0`**-Bits im Register **`EIMSK`**

Resettaste: Interruptleitung an **`PORTD`**, Pin 3

- active-low: Wird die Taste gedrückt, so 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 **`INT1`**, ISR-Vektor-Makro: **`INT1_vect`**
- Aktivierung der Interruptquelle erfolgt durch Setzen des **`INT1`**-Bits im Register **`EIMSK`**

Konfiguration der externen Interruptquellen **`INT0`** und **`INT1`** (Bits in Register **`EICRA`**)

Interrupt 0		Beschreibung	Interrupt 1	
ISC01	ISC00		ISC11	ISC10
0	0	Interrupt bei low Pegel	0	0
0	1	Interrupt bei beliebiger Flanke	0	1
1	0	Interrupt bei fallender Flanke	1	0
1	1	Interrupt bei steigender Flanke	1	1

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

*Inhalt der vorgegebenen Bibliotheksdatei **display.h***

```
#ifndef _DISPLAY_H
#define _DISPLAY_H

#include <stdint.h>

// Enumeration zur Auswahl des Anzeigefeldes
typedef enum {
  SPEED,     // Geschwindigkeit
  DISTANCE,  // Distanz
} field;

// Display initialisieren
void display_enable(void);

// Ausgabe eines Wertes auf dem vorgegebenen Feld
void display_show(field, uint16_t);

#endif
```

---

*Inhalt der Datei **main.c***

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdint.h>

#include "display.h"

// Rollumfang in cm
// (zurueckgelegte Distanz pro volle Radumdrehung)
const uint8_t circumference = 215;


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

A:


```
// Unterbrechungsbehandlungsfunktionen
```

```
// Ende Unterbrechungsbehandlungsfunktionen
```

U:

```
// Funktion main

  // Initialisierung und lokale Variablen

  // Hauptschleife

    // Warten auf Ereignisse
```

M:


```
    // Ausgabe auf dem Display

// Ende der Main
```

O:

```
// Initialisierungsfunktion
```

```
// Ende Initialisierungsfunktion
```

I:

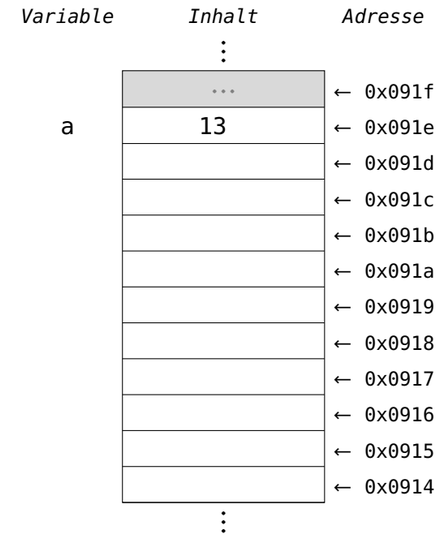
Aufgabe 3: Zeiger und Felder (10 Punkte)

Das folgende Programm wird ohne Optimierungen übersetzt und auf einem 8-Bit AVR/ATmega32 Mikrocontroller ausgeführt.

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

```
1  #include <stdint.h>
2
3  static uint8_t * min(const uint8_t array[], uint8_t size){
4    uint8_t min = *array;
5    while (--size > 0){
6      if (array[size] < min){
7        min = array[size];
8      }
9    }
10   return &min;
11 }
12
13 void main(void) {
14   uint8_t a = 13;
15
16   // Aufgabe a)
17   uint8_t b = 1;
18   uint8_t * c = &a;
19   *c *= b + 1;
20   uint8_t d[3] = {3, 7};
21   uint8_t e = *(d + b) * 3;
22   // Ende Aufgabe a)
23
24   // Aufgabe d)
25   uint8_t *m = min(d, 4);
26   uint8_t n = 42;
27   n += *m;
28
29   /* ... */
30
31 }
```

Dazugehöriger Stack (Auszug):



a) Obige Grafik zeigt einen Speicherauszug des Stacks nach der Ausführung der Zuweisung in Zeile 14 an. Ändern / erweitern Sie die Grafik um einen möglichen Aufbau des Stacks (Variable und Inhalt) nach der Ausführung der Zeilen 16 bis 22. (4 Punkte)

b) Die Funktion `min()` erhält als ersten Parameter ein `array`. Wie wird in C ein Feld als Funktionsparameter übergeben? Was bewirkt im Beispiel das Schlüsselwort **`const`**? (2 Punkte)

c) Erklären Sie den Unterschied zwischen den Typen **`const uint8_t *`** und **`uint8_t * const`**. (2 Punkte)

d) Die Funktion `min` soll das kleinste Element eines Feldes (Arrays) finden. Welches Problem kann bei der Verarbeitung des Rückgabewertes von `min` (Zeile 24ff) auftreten? Wieso? (2 Punkte)

Aufgabe 4: Nebenläufigkeit (8 Punkte)

Das nachfolgende Codebeispiel für einen 8-Bit-AVR-Mikrocontroller soll ein generisches Grundgerüst für eine stromsparende Ereignisbehandlung darstellen: Der Mikrocontroller befindet sich solange in einem Schlafzustand, bis die Unterbrechungsbehandlung von `INT0` asynchron aufgerufen und damit mittels der modul-globalen Variable `event` die Ereignisbehandlung der `main()` Funktion ausgeführt wird.

Jedoch beinhaltet diese Implementierung ein Nebenläufigkeitsproblem.

```
1  #include <avr/interrupt.h>
2  #include <avr/sleep.h>
3
4  static volatile int16_t event = 0;
5
6  ISR(INT0_vect){
7      event |= 1;
8  }
9
10 void main(void) {
11
12     /* ... */
13
14     for(;;) {
15         cli();
16         while (!event) {
17             sei();
18             sleep_enable();
19             sleep_cpu();
20             sleep_disable();
21             cli();
22         }
23         event = 0;
24         sei();
25
26         /* ... Ereignisbehandlung ... */
27
28     }
29 }
```

a) Benennen Sie das Nebenläufigkeitsproblem und kennzeichnen Sie den kritischen Abschnitt im Quellcode. (2 Punkte)

b) Skizzieren Sie einen konkreten Ablauf, der hier zu einem Nebenläufigkeitsfehler führt, und seine Auswirkungen (Stichpunkte, kurze Sätze). (4 Punkte)

c) Beschreiben und begründen Sie die notwendigen Änderungen am Programmcode, um dieses Problem zu vermeiden. (2 Punkte)