

**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) Welche Aussage zu Zeigern ist richtig?

2 Punkte

- Ein Zeiger darf nie auf seine eigene Speicheradresse verweisen.
- Der Speicherbedarf eines Zeigers ist unabhängig von der Größe des Objekts, auf das er zeigt.
- 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.

b) Gegeben ist folgender Programmcode:

```
#define SUB(a,b) a-b  
#define MUL(a,b) a*b
```

Was ist das Ergebnis von folgendem Ausdruck:

```
4 * SUB( MUL(2,3) , 4)
```

2 Punkte

- 8
- 8
- 20
- 96

c) Welche der folgenden Aussagen bzgl. der Interruptsteuerung ist richtig?

2 Punkte

- Wurde gerade ein Pegel-gesteuerter Interrupt ausgelöst, so muss erst ein Pegelwechsel der Interruptleitung stattfinden, bevor erneut ein Interrupt ausgelöst wird.
- Beim Auftreten eines Interrupts sichert der Prozessor automatisch Register in den Speicher.
- Pegel-gesteuerte Interrupts werden beim Wechsel des Pegels ausgelöst.
- Flanken-gesteuerten Interrupts können nicht blockiert werden, da sie völlig unvorhersehbar auftreten.

d) Welche Aussage zu *volatile* ist richtig?

2 Punkte

- Das Schlüsselwort *volatile* unterbindet Optimierungen an einer damit definierten Variable.
- Das Schlüsselwort *volatile* unterbindet alle Nebenläufigkeitsprobleme.
- Das Schlüsselwort *volatile* hat nur noch eine historische Bedeutung und ist in heutigen Programmen nicht mehr nötig.
- Das Schlüsselwort *volatile* erlaubt dem Compiler bessere Optimierungen durchzuführen.

e) Was ist ein Stack-Frame?

2 Punkte

- Ein bei Interrupts auftretendes spezielles Nebenläufigkeitsproblem.
- Ein Interrupt, der einen Überlauf des Stack-Speichers signalisiert.
- Ein Bereich des Speichers, in dem unter anderem Übergabeparameter für Funktionen abgelegt werden.
- Ein Bereich des Speichers, in dem lokale static-Variablen von Funktionen abgelegt werden.

f) In welcher Situation kann ein lost-update-Problem auftreten?

2 Punkte

- Bei der Modifikation von lokalen Variablen in zwei Interrupt-Handlern.
- Wenn eine Interrupt-Sperre zu lange gehalten wird.
- Wenn während einer Interrupt-Bearbeitung weitere Interrupts gesperrt sind.
- Wenn sowohl in der main-Funktion als auch in einem Interrupt-Handler modifizierend auf die gleiche `uint16_t`-Variable zugegriffen wird.

g) Welche Aufgabe erfüllt der C-Präprozessor?

2 Punkte

- Er entfernt nicht sichtbare globale Variablen aus dem Programm.
- Er bindet mehrere `.o`-Dateien zusammen.
- Er erzeugt einen internen Zwischencode (ähnlich dem Java-Bytecode), der dann durch den C-Compiler in ausführbaren Code umgewandelt wird.
- Er führt textuelle Ersetzungen im C-Code durch, die sich durch Makrodefinitionen steuern lassen.

- h) Welche Aussage zu Traps und Interrupts auf modernen Computern ist richtig? 2 Punkte
- Interrupts werden aus Effizienzgründen direkt an die entsprechende Anwendung geschickt.
  - Wird ein Interrupt ausgelöst, wechselt der Prozessor in den privilegierten Modus und springt an eine zuvor konfigurierte Stelle des Betriebssystems.
  - Traps werden vom Programm ausgelöst und müssen daher auch von diesem bearbeitet werden.
  - Interrupts werden durch eine spezielle Hardwareeinheit "I2S" in Signale umgewandelt, welche dann an den entsprechenden Treiber weitergeleitet werden.
- i) Welche Aussage zu Prozessen ist richtig? 2 Punkte
- In einem Prozess können mehrere Programme gleichzeitig ausgeführt werden.
  - Laufen mehrere Prozesse auf einer CPU, können diese direkt Daten austauschen.
  - Auf Multi-Core-Systemen können mehrere Prozesse gleichzeitig ausgeführt werden. Jedoch nur so viele, wie Prozessorkerne zur Verfügung stehen.
  - Auf Single-Core-Systemen gibt es keine Nebenläufigkeit.
- j) Welche Aussage zu dem Systemaufruf `fork()` ist richtig? 2 Punkte
- Die im Vaterprozess geöffneten Dateien bleiben auch im Kindprozess geöffnet.
  - Vater- und Kindprozesse besitzen den gleichen Speicher und können über diesen kommunizieren.
  - `exec()` kann nur im Kindprozess aufgerufen werden.
  - Da `fork()` im Kindprozess 0 zurückliefert, ist es für diesen unmöglich, die Prozess-Id des Vaterprozesses zu bestimmen.

**Aufgabe 2a: safe (30 Punkte)**

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

Schreiben Sie eine Steuerung für einen AVR-Mikrocontroller in einem Safe, der mit einem 8-stelligen Binärschlüssel gesichert ist. Der Code wird während der Eingabe durch LEDs visualisiert.

Die Eingabe des Schlüssels erfolgt mit Hilfe von zwei Tastern, die entsprechend mit "0" (Taster 0) und "1" (Taster 1) beschriftet sind. Jede der 8 Stellen des Schlüssels wird durch den Benutzer nacheinander durch Drücken eines der beiden Taster eingegeben. Die 8 LEDs visualisieren den eingegebenen Schlüssel, so dass die LEDs nach der Eingabe jeder Stelle den bisher eingegebenen Binärcode anzeigen.

Nach der achten eingegebenen Stelle wird der Schlüssel mit einer gegebenen statischen Konstante SECRET verglichen. War die Eingabe erfolgreich, öffnet sich das Schloss für eine Gesamtdauer von 5 Sekunden während alle LEDs mit einer Periodendauer von einer Sekunde (1000 ms) blinken, dann schließt das Schloss wieder. Nach dieser Sequenz oder im Fall eines falschen Schlüssels startet die Eingabe des Schlüssels wieder von vorne.

Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion `void init(void)`.
- Das Programm startet mit allen LEDs ausgeschaltet, das Schloss ist geschlossen und die Eingabe des Benutzers wird erwartet. Während des Wartens auf eine Eingabe soll der Mikrocontroller jeweils in den Schlafmodus gehen.
- Implementieren Sie das Blinken der LEDs während der Öffnung des Safes unter Verwendung einer aktiven Wartefunktion `void wait(uint16_t ms)`, die *ms* Millisekunden wartet. Ihnen steht eine Präprozessorkonstante `LOOPS_PER_MS` zur Verfügung, die angibt, wieviele Schleifendurchläufe gewartet werden muss, um eine Millisekunde verstreichen zu lassen.

**Information über die Hardware**

LEDs: **PORTA**, Pins 0-7, Start bei LED 1 an Pin 0, eingeschaltet bei low-Pegel

- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRA**-Register auf 1

Schloss: **PORTB**, Pin 5, offen bei high-Pegel, geschlossen bei low-Pegel

- Pin als Ausgang konfigurieren: entspr. Bit in **DDRB**-Reg. auf 1

Taster: **PORTD**, "Null"-Taster (Taster 0) = Pin 2, "Eins"-Taster (Taster 1) = Pin 3

- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Register auf 0

- externe Interruptquellen **INT0** und **INT1**, ISR-Vektor-Makros: **INT0\_vect** und **INT1\_vect**

- Aktivierung der Interruptquellen erfolgt durch Setzen des **INT0**- bzw. **INT1**-Bits im Register **GICR**.

- die Taster verbinden den Pin mit Masse, es müssen die internen Pullup-Widerstände verwendet werden (entsprechende Bits in **PORTD**-Register auf 1 setzen).

- Konfiguration der externen Interruptquellen 0 und 1 (Bits in Register **MCUCR**)

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



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

```
.....  
/* Initialisierung und lokale Variablen */
```

```
.....  
/* Hauptschleife */
```

```
.....  
/* Code-Eingabe */
```

```
.....  
/* Schloss oeffnen, naechster Durchlauf */
```

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

**M:**



**Aufgabe 2b: stringstack (18 Punkte)**

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

Implementieren Sie ein Modul `stringstack`, in dem man Zeichenketten speichern kann und das die zuletzt hinzugefügte Zeichenkette als erstes wieder zurück gibt.

Im Detail soll das Modul `stringstack` folgendermaßen funktionieren:

- Zur internen Verwaltung der Zeichenketten wird eine verkettete Liste verwendet. Der Aufbau der Listenelemente ist vorgegeben (`struct elem`).
- Die Funktion `int pushstr(char* st)` bekommt eine Zeichenkette übergeben, speichert diese möglichst platzsparend und fügt sie mit Hilfe eines neuen Listenelements am Anfang der verketteten Liste ein.  
Im Fehlerfall wird `-1` zurückgegeben (an dieser Stelle keine Meldung ausgeben).
- Die Funktion `int popstr(char *buf, int bufsize)` überprüft zunächst, ob der übergebene Puffer für die zu entnehmende Zeichenkette groß genug ist. Ist er zu klein, wird die tatsächlich benötigte Puffergröße zurückgegeben.  
Andernfalls wird das erste Element aus der verketteten Liste entnommen und die zugehörige Zeichenkette in den übergebenen Puffer kopiert.  
Im Erfolgsfall wird `0` zurückgeliefert, befinden sich keine Elemente in der Liste wird `-1` zurückgeliefert.

Achten Sie auf die Behandlung möglicher Fehler sowie darauf, dass allozierter Speicher jeweils so bald wie möglich wieder freigegeben wird.

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



**Modul stringstack.c**

```
#include <stdlib.h>
#include <string.h>
#include "stringstack.h"
```

```
struct elem {
    struct elem * next;
    char * str;
};
```

```
// Modul-globale Variablen
```

.....

.....

```
// Funktion pushstr
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**U:**







