

Aufgabe 1: (14 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) Gegeben ist folgender Programmcode:

```
uint16_t x[] = {1, 2, 3, 5, 7};
uint16_t *y = &x[4];
y--;
```

2 Punkte

Welchen Wert liefert die Dereferenzierung von **y**?

- 2
- 5
- 4
- Zur Laufzeit tritt ein Fehler auf.

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

2 Punkte

- Pegelgesteuerte Interrupts werden beim Wechsel des Pegels ausgelöst, daher der Name.
- Pegelgesteuerte Interrupts müssen durch Pollen des Pegels abgefragt werden.
- 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.

c) Was versteht man unter den formalen und tatsächlichen Parametern einer Funktion?

2 Punkte

- Der tatsächliche Parameter enthält eine Kopie des Aufrufparameters, der formale Parameter ist ein Zeiger auf den Aufrufparameter.
- Die formalen Parameter sind die Namen, unter denen auf die Aufrufparameter innerhalb der Funktion zugegriffen werden kann.
- Die tatsächlichen Parameter sind die Rückgabewerte eines Funktionsaufrufs.
- Die tatsächlichen Parameter sind die Namen, unter denen innerhalb einer Funktion auf die Aufrufparameter zugegriffen wird.

d) 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.

e) Gegeben sei folgende Enumeration:

```
enum FARBE {rot, gruen, blau};
```

Welche Aussage ist richtig?

2 Punkte

- Der Compiler meldet einen Fehler, weil den enum-Elementen kein Wert zugewiesen wurde.
- Der Wert von **blau** ist 2.
- Der Wert von **blau** ist 3.
- Der Wert von **blau** ist unbekannt; der Compiler weist zur Übersetzungszeit jedem enum-Element einen zufälligen, aber eindeutigen Wert zu.

f) Gegeben sei folgender Programmcode:

```
int array[10];
int i;
for (i = 1; i <= 10; i++) {
    array[i] = 0;
}
```

2 Punkte

- Allen Elementen des Feldes **array** wird durch Ausführung des Programm-Codes der Wert 0 zugewiesen.
- Das Programm stürzt mit einem Segmentation-Fault ab.
- Das Programm könnte in eine Endlosschleife geraten.
- Der C-Compiler meldet beim Übersetzen einen Semantikfehler.

g) Welche der folgenden Aussagen über den C-Präprozessor ist richtig?

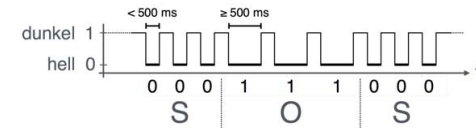
2 Punkte

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

Aufgabe 2a: SOS-Erkennung (30 Punkte)

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

Schreiben Sie ein Programm für den AVR-Mikrocontroller, das Morsecode empfängt und das Notsignal "SOS" erkennt. Das Programm bekommt Interrupts eines Lichtsensors und bestimmt mit einem Timer die Länge der Impulse. Wird die Folge erkannt, wird ein Alarm (blinkende LED) ausgelöst. Über einen Taster lässt sich das Gerät zurücksetzen.



Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion `void init(void)`. Treffen Sie keine Annahme über den initialen Zustand der Hardware-Register.
- Zu Beginn geht das Programm in den Empfangsmodus, in dem die Alarm-LED ausgeschaltet ist und der Mikrocontroller im stromsparenden Schlafmodus auf Ereignisse wartet.
- Eine fallende Flanke des Lichtsensors löst einen Interrupt aus, durch den die Zeitmessung beginnt. Beim nächsten Interrupt durch eine steigende Flanke wird durch erneute Zeitmessung die Dauer ermittelt. War die Dauer kürzer als 500 ms, wurde ein kurzes Zeichen erkannt, ansonsten ein langes Zeichen.
- Zur Zeitmessung steht Ihnen die Funktion `uint32_t getTime(void)` zur Verfügung, welche die Millisekunden seit Start des Mikrocontrollers zurückliefert. Diese Funktion darf nicht im Unterbrechungskontext aufgerufen werden.
- Die Zeichen sollen als Bits (0 = kurz, 1 = lang) mit einem Shift nach links in einer Bitmaske gespeichert werden. Wird das Notsignal durch den Vergleich der letzten 9 Bits mit dem vorgegebenen Makro `SOS` erkannt, dann wechselt das Programm in den Alarmmodus.
- Im Alarmmodus blinkt die LED, indem sie alle 300 ms ihren Zustand ändert. Implementieren Sie für das Warten eine aktive Wartefunktion `void wait(uint32_t ms)`, die mit Hilfe der Funktion `getTime()` wartet, bis `ms` Millisekunden vergangen sind.
- Bei einem Tasterdruck wird die bislang empfangene Zeichenfolge zurückgesetzt. Befindet sich das Programm im Alarmmodus so wird außerdem in den Empfangsmodus zurückgewechselt.

/* Funktion main */

/* Initialisierung und lokale Variablen */

/* Hauptschleife */

/* Warten auf Ereignisse */

K:

/* Ereignisse bearbeiten */

/* Ende main */

E:

b) Skizzieren Sie einen konkreten Ablauf, der hier zu einem Nebenläufigkeitsfehler führt, und seine Auswirkungen (Stichpunkte, kurze Sätze). (2 Punkte)

.....
.....
.....
.....
.....
.....
.....
.....

c) Benennen Sie eine Technik, mit der sich das Problem vermeiden lässt. Nehmen Sie die entsprechenden Änderungen am Programmcode vor. (2 Punkte)

.....
.....
.....
.....
.....
.....
.....
.....