Friedrich-Alexander-Universität Erlangen-Nürnberg

Schriftliche Prüfung zu der Lehrveranstaltung „Systemnahe Programmierung in C (SPiC)“ im den Bachelor-Studiengängen Mechatronik, Mathematik, Technomathematik

	erreichbare Punkte	erhaltene Punkte						
Aufgabe 1	**15**							
Aufgabe 2	**50**	**L**	**I**	**D**	**C**			
Aufgabe 3	**25**	**3a**	**3b**	**3c**	**3d**			
Summe	**90**							
Note								

(Name) (Vorname) (Matrikel-Nr.)

(Studiengang) (Semester)

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausurunterlagen (15 Seiten inkl. Deckblatt sowie ein Blatt mit den Manualseiten zu opendir/readdir/closedir, exec, fork, strcmp),
- die Kenntnisnahme der Hinweise auf Seite 2.

Erlangen, 25.07.2008

(Unterschrift)

Hinweise

Die folgenden Informationen bitte aufmerksam lesen und die Erklärung auf Seite 1 unterschreiben.

- Die Bearbeitungszeit beträgt 60 Minuten.
- Es sind **keine** eigenen Hilfsmittel zugelassen.
- Die Lösung einer Aufgabe soll auf das Aufgabenblatt in den dafür vorgesehenen Raum geschrieben werden. Beachten Sie bitte, dass der freigelassene Platz großzügig bemessen ist und nicht unbedingt der erwarteten Antwortlänge entspricht. Sollte der Platz nicht ausreichen, können Sie die Rückseiten der Aufgabenblätter mitverwenden. Kennzeichnen Sie dabei die Zugehörigkeit Ihrer Lösung zu einer Aufgabe. Bei Bedarf können zusätzliche Lösungsblätter (weiß) ausgeteilt werden. Vermerken Sie vor deren Verwendung unbedingt Ihren Namen und Ihre Matrikelnummer darauf!
- Die Lösungen müssen dokumentenecht in blau oder schwarz geschrieben werden. Als falsch Erkanntes muss deutlich durchgestrichen werden. Tintenkiller darf nicht verwendet werden. Keinen Bleistift verwenden!
- Schmierpapier (farbige Blätter) darf **nicht** abgegeben werden. Bei Bedarf ist von der Aufsicht weiteres Schmierpapier erhältlich.
- Fragen zu den Prüfungsaufgaben können grundsätzlich nicht beantwortet werden.
- Tragen Sie Ihren Namen und Vornamen, Ihre Matrikelnummer, Studiengang und Fachsemesterzahl auf dem Deckblatt der Klausur ein.
- Bitte legen Sie Ihren Studenten- und einen Lichtbildausweis zur Kontrolle bereit.
- Sie dürfen den Raum nicht verlassen bevor Ihre Personalien überprüft wurden und Sie die Klausurunterlagen der Aufsicht zurückgegeben haben.
- In den letzten 15 Minuten der Bearbeitungszeit können Sie den Raum nicht mehr verlassen. Bleiben Sie an Ihrem Platz sitzen, bis am Ende alle Klausurunterlagen eingesammelt und nachgezählt sind und die Aufsicht das Zeichen zum Gehen gibt.

Die **Klausurergebnisse** werden in ca. zwei Wochen im WWW unter der Seite der Vorlesung im SS 2008, Unterpunkt "Ergebnisse" veröffentlicht.

Aufgabe 1: (15 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 Aussage zu Zeigern ist **richtig**? 2 Punkte

- ❏ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-value.
- ❏ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-reference.
- ❏ 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) 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.

c) 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.
- ❏ Programme sind C-Quellcode-Dateien, die durch einen C-Compiler in einen lauffähigen Prozess übersetzt werden können.
- ❏ 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.

d) Welche der folgenden Aussagen bzgl. der Interruptsteuerung **trifft zu**? 2 Punkte

- ❏ Pegel-gesteuerte Interrupts werden beim Wechsel des Pegels ausgelöst, daher der Name.
- ❏ Interrupts sind eine Besonderheit von AVR-Mikroprozessoren. Auf anderen Architekturen kommen POSIX-Signale zum Einsatz.
- ❏ Pegelgesteuerte Interrupts müssen durch Polling 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.

e) Gegeben ist folgendes Makro: 2 Punkte

```
#define ADD(a,b) a+b
#define MUL(a,b) a*b
```

Was ist das Ergebnis von folgendem Ausdruck:

```
4 * MUL( ADD(1,2), 3)
```

- ❏ 10
- ❏ 18
- ❏ 28
- ❏ 36

f) Welche Aussage zu folgender Funktion ist **richtig**? 2 Punkte

```
int *foo() {
   static int bar = 0;
   bar++;
   return &bar;
}
```

- ❏ Die Funktion liefert einen Zeiger auf die lokale Variable `bar` zurück. Dies ist in C nicht zulässig und führt zu einem Übersetzungsfehler.
- ❏ Beim Verlassen der Funktion `foo()` wird die automatic-Variable `bar` vom Stack entfernt und der Zeiger verliert seine Gültigkeit. Ein Zugriff durch den Aufrufer führt zu zufälligen Ergebnissen.
- ❏ Die Variable `bar` ist über die Laufzeit der `foo()`-Funktion hinaus gültig. Daher kann der zurückgelieferte Zeiger sicher vom Aufrufer verwendet werden.
- ❏ Die Variable `bar` enthält beim Verlassen der `foo()`-Funktion immer den Wert 1, da `bar` bei jedem Aufruf von `foo()` mit 0 initialisiert wird.

g) Was ist **keinesfalls** eine Eigenschaft eines Hardware-Timers?

3 Punkte

- ❏ Ein Hardware-Timer ist ein Register welches mit der steigenden oder fallenden Flanke eines angelegten Taktes inkrementiert wird.
- ❏ Ein Hardware-Timer kann beim Überlauf einen Interrupt generieren.
- ❏ Durch Vorschaltung einer Prescalereinheit kann die Taktrate des Timers erhöht werden.
- ❏ Eine CTC-Einheit (Clear Timer on Compare Match) kann im Zusammenspiel mit einer Output-Compare-Einheit zur periodischen Generierung von Interrupts verwendet werden.

Aufgabe 2a: (30 Punkte)

Schreiben Sie ein Lauflicht-Programm für einen AVR-Mikrokontroller mit 8 LEDs an Port A. Es soll immer *genau eine LED aktiv* sein, im Ausgangszustand die LED an Pin 0. Nach Erreichen der LED an Pin 7 erhöht das Programm die Geschwindigkeit des Lauflichts und beginnt wieder mit der LED an Pin 0. Nach Erreichen einer Maximalgeschwindigkeit stoppt das Programm nach Einschalten der LED an Pin 7. Mit einem Taster kann die Geschwindigkeit während des Programmablaufs reduziert werden. Die Geschwindigkeitsänderung wird jeweils zu Beginn des nächsten Durchlaufs aktiv. Im Detail soll sich das Programm wie folgt verhalten:

- Die Initialisierung der Hardware soll in einer eigenen Funktion `void init()` erfolgen, die zu Beginn des Programms aufgerufen wird. Es können hierbei keine Annahmen über den initialen Zustand der Hardware gemacht werden.
- Anschließend erfolgt ein kompletter Lauflichtdurchlauf mit der aktuellen Geschwindigkeit.
- Zum Warten zwischen dem Umschalten zweier LEDs wird die Funktion `void wait(unsigned int interval)` aufgerufen. In dieser Funktion wird aktiv in einer Warteschleife mit `interval` Durchläufen gewartet.
- Die Zahl der Schleifendurchläufe der `wait`-Funktion soll sich im Bereich [1000; 10000] bewegen und sich in 1000er Schritten nach einem Lauflichtdurchlauf bzw. einem Tastendruck verändern. In der Ausgangsgeschwindigkeit ist die Zahl der Schleifendurchläufe 5000.
- Nach einem Lauflichtdurchlauf wird der Mikrokontroller in den Standard-Stromsparmodus versetzt falls die Durchlaufgeschwindigkeit das Maximum erreicht hat. Durch einen Tastendruck soll der Mikrokontroller aus dem Stromsparmodus geweckt werden können und mit der Ausgangsgeschwindigkeit beginnen.

Information über die Hardware

LEDs: **PORTA**, Pins 0-7, Start bei LED an Pin 0
- Pin als Ausgang konfigurieren: entspr. Bit in **DDRA**-Reg. auf 1

Taster: **PORTD**, Pin 2
- Pin als Eingang konfigurieren: entspr. Bit in **DDRD**-Reg. auf 0
- externe Interruptquelle **INT0**, ISR-Vektor-Makro: **INT0_vect**
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT0**-Bits im Register **GICR**.
- Taster verbindet den Pin mit Masse, es muss der interne Pullup-Widerstand verwendet werden (entspr. Bit in **PORTD**-Reg. auf 1 setzen).
- Konfiguration der externen Interruptquelle 0 (Bits in Register **MCUCR**)

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

/* Funktiondeklarationen, globale Variablen, etc. */

-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
/* Unterbrechungsbehandlungsfunktion */

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



/* Funktion main */

-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
  /* Initialisierung */

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

```
  /* Hauptschleife */
```

```
    /* ein Lauflicht-Durchlauf */
```

```
    /* Vorbereitung des naechsten Durchlaufs bzw. Schlafen */
```

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

L:

```
/* Funktion init */

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
/* Ende der Funktion init */
/* Funktion wait() */

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
/* Ende der Funktion wait() */
```

I:

Aufgabe 2b: (20 Punkte)

Schreiben Sie eine Funktion `void compiledir()`, welche alle C-Dateien (Dateiendung ".c") im aktuellen Verzeichnis übersetzt.

Die Funktion soll wie folgt funktionieren:

- Sämtliche Fehlermeldungen sollen auf den Standardfehlerkanal `stderr` ausgegeben werden, nicht auf die Standardausgabe. Im Fehlerfall soll die Funktion das gesamte Programm beenden.
- Das Programm durchsucht das aktuelle Verzeichnis. Für alle gefundenen Dateien, die mit dem String ".c" enden, wird die Funktion
  ```
  void compile(const char *datei);
  ```
  aufgerufen. Dieser wird der Name der C-Datei übergeben.
- Funktion `compile`: In dieser Funktion wird ein neuer Prozess zur Übersetzung der C-Datei erzeugt. In diesem Prozess soll zur Übersetzung das Kommando
  `cc -c` *C-Datei*
  ausgeführt werden. Das `cc`-Kommando soll hierbei im Systempfad gesucht werden.
- Die Funktion `compiledir` kehrt zurück, nachdem das Verzeichnis vollständig durchlaufen wurde.

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

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <dirent.h>
#include <errno.h>
#include <unistd.h>
/* Funktiondeklarationen, globale Variablen, etc. */

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

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

----------------------------------------------------------------------------
/* Funktion compiledir */

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

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

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

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

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

----------------------------------------------------------------------------
/* aktuelles Directory oeffnen */

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

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

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

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

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

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

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

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

```
  /* Directory durchsuchen und .c-Dateien compilieren */
```

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

```
/* Ende der Funktion compiledir */
```

D:

```
/* Funktion compile */

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
/* Ende der Funktion compile */
```

C:

Aufgabe 3: (25 Punkte)

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

a) Eine wesentliche Aufgabe von Betriebssystemen ist die Verwaltung von Betriebsmitteln. Geben Sie eine Klassifizierung von Betriebsmitteln an und nennen Sie jeweils ein typisches Beispiel.

b) Beschreiben Sie die Unterschiede bei der Ausführung eines Programms auf einem Mikrokontroller ohne Betriebssystem und auf einem Betriebssystem wie z. B. Linux.

c) Beschreiben Sie den Unterschied zwischen Pegel- und Flanken-gesteuerten Interrupts.

d) Eine Lichtschranke ist so an einem Port angeschlossen, dass sie genauso wie ein Taster Interrupts auslösen kann. Skizzieren Sie eine Lösung, wie sie die Zeit der Unterbrechung mit einer Genauigkeit von 1 ms ermitteln können.
Ihnen steht eine mit 1 MHz getaktete 8-Bit-Timer-Einheit mit einer Output-Compare-Einheit zur Verfügung. Der Takt kann durch eine Prescaler-Einheit mit den Teilern 1, 8, 64, 256, 1024 skaliert werden.

Skizzieren Sie,

- wie Sie den Timer konfigurieren würden.
- wie Sie mit den Interrupts der Lichtschranke und des Timers umgehen (kein vollständiges Programm, nur die wesentlichen Schritte angeben!).