Aufgabe 1: (12 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 virtuellen Adressräumen und der Abbildung auf den realen Hauptspeicher ist **falsch**? 2 Punkte

- ❑ Jede Seite des virtuellen Adressraums wird entweder auf ein entsprechendes Stück des realen Hauptspeichers abgebildet oder sie kann auch auf Festplatte vorübergehend ausgelagert werden.
- ❑ Bei vielen Prozessoren können zu einer Seite Zugriffsrechte (z. B. lesen, schreiben) vermerkt werden. Bei einem Zugriff, der gegen diese Rechte verstößt wird das Betriebssystem informiert, das den verursachenden Prozess abbrechen kann.
- ❑ X Die Umrechnung von virtueller Adresse auf reale Adresse erfolgt beim Laden des Programms in den Speicher.
- ❑ Die Umrechung von virtueller Adresse auf reale Adresse erfolgt zur Laufzeit des Programms durch eine spezielle Hardwareeinheit.

b) Was ist der Unterschied zwischen den wie folgt in der Datei prog.c global definierten Variablen? 2 Punkte

```
int a;
static int b;
```

- ❑ X Die Variable b ist nur für Funktionen in der Datei prog.c zugreifbar. Funktionen in anderen Modulen des Programms können auf a zugreifen wenn in dem entsprechenden Modul a mit einer extern-Deklaration bekannt gemacht wurde.
- ❑ Die Variable a ist nur für Funktionen in der Datei prog.c zugreifbar während auf b auch von anderen Modulen des Programms erreichbar ist, wenn es dort als extern deklariert wird.
- ❑ Der Speicherplatz der Variablen a wird jeweils beim Aufruf einer Funktion angelegt und beim Verlassen wieder freigegeben während der Speicherplatz der Variablen b von Programmstart bis -ende verfügbar ist.
- ❑ Der Speicherplatz der Variablen b für die gesamte Ausführungszeit des Programms reserviert ist. Deshalb darf es keine lokalen Variablen mit diesem Namen geben. Bei a wäre dies kein Problem.

c) Was ist ein Stack-Frame? 2 Punkte

- ❑ Der Speicherbereich, in dem der Programmcode einer Funktion abgelegt ist.
- ❑ Ein spezieller Registersatz des Prozessors zur Bearbeitung von Funktionen.
- ❑ Ein Fehler, der bei unberechtigten Zugriffen auf den Stack-Speicher entsteht.
- ❑ X Ein Bereich des Speichers, in dem lokale Variablen einer Funktion abgelegt sind.

d) Welche Aussage zu Zeichenketten in C (C-Strings) ist richtig? 2 Punkte

- ❑ C-Strings speichern im ersten Byte ihre Länge.
- ❑ C-Strings werden in C durch den Datentyp `String` repräsentiert.
- ❑ X C-Strings werden durch ein `char`-Feld implementiert.
- ❑ C-Strings können maximal 255 Zeichen enthalten.

e) Wozu dient das **`#ifdef`**-Konstrukt in C? 2 Punkte

- ❑ Es kann alternativ zu if-Abfragen eingesetzt werden.
- ❑ X Man kann damit Programmteile bei der Übersetzung ausblenden.
- ❑ Es kann eingesetzt werden, um sicherzustellen, dass ein Programmteil ein definiertes Ergebnis liefert.
- ❑ Es überprüft, ob die danach angegebenen Variablen definert wurden.

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

- ❑ Ein Programm kann zu einem Zeitpunkt nur einmal gleichzeitig auf einem Rechner ausgeführt werden.
- ❑ Ein Programm kann mehrfach gestartet werden. Jede Programmausführung erfolgt in einem Prozess. Tritt bei einer dieser Programmausführungen ein Fehler auf (z. B. Segmentation Fault), so werden alle Prozesse, die das Programm gerade ausführen abgebrochen.
- ❑ Die Begriffe Programm und Prozess bedeuten eigentlich das Gleiche. Der Begriff Programm stammt aus der Windows-Welt, während man in Linux-Systemen statt dessen meistens von Prozessen spricht.
- ❑ X Ein Prozess ist einProgramm, das sich in Ausführung befindet. Es ist möglich, dass verschiedene Prozesse das gleiche Programm ausführen, jedoch dabei unterschiedliche Zugriffsrechte (z. B. auf Dateien) haben.

Aufgabe 2: (30 Punkte)

a) Schreiben Sie eine Funktion **ctl_int** mit der man an einem ATmega128 einfach die verschiedenen Interrupts aktivieren bzw. deaktivieren kann:

```
void ctl_int(int what, int int_nr);
```

Für den Parameter **what** können die Werte **ENABLE** und **DISABLE** (gehen Sie davon aus, dass diese Makros vordefiniert sind) eingesetzt werden, **int_nr** gibt den betroffenen Interrupt an. Wird für **int_nr** der Wert -1 übergeben, so soll die Operation auf alle Interrupts wirken. Die Funktion soll bei ungültigen Werten für **what** oder **int_nr** nichts tun.

Externe Interrupts werden beim ATmega128 im Register **EIMSK** maskiert. Ein Interrupt wird durch Setzen von Bit **INT***x* aktiviert und durch Löschen deaktiviert. D. h. für jeden Interrupt ist ein Makro definiert, welches die Position des zuständigen Bits in **EIMSK** angibt - z. B. gibt Makro **INT0** das Bit an, das für Interrupt Nr. 0 zuständig ist, **INT7** das Bit für Interrupt Nr. 7..

```
/* Funktion ctl_int */
void ctl_int(int what, int int_nr) {
   unsigned char map[] = {INT0, INT1, INT2, INT3,
                          INT4, INT5, INT6, INT7};
   if (int_nr < -1 || int_nr > 7) return;        /* 1 */

   if(what == ENABLE) {                          /* if 1 */
      /* korrektes shiften: 3 - int_nr shiften wäre falsch! */
      if (int_nr == -1 ) { /* Fall insges. richtig beh. 3 */
         EIMSK = 0xff; /* oder sei() */          /* 1 */
      } else {
         EIMSK |= 1 << map[int_nr];            /* 1 */
      }
   } else if(what == DISABLE) {
      if (int_nr == -1 ) {
         EIMSK = 0; /* oder cli() */           /* 1 */
      } else {
         EIMSK &= ~(1 << map[int_nr]);         /* 1 */
      }
   }
}
```

12

C: 12

b) Schreiben Sie ein Programm **ueberwachung**, das an einem I/O-Port Messwerte überwacht. An dem I/O-Port ist ein Sensor angeschlossen, der Werte im Bereich 0 - 255 liefert. Der I/O-Port ist 8 Bit breit und in den Speicher auf Adresse 0x19 abgebildet.

Das Programm soll den Port pollen. Ein neuer Messwert liegt immer dann vor, wenn sich der Wert verändert. Für jeden neuen Messwert sollen die folgenden beiden Funktionen **melde** und **statistik** aufgerufen werden:

Die Funktion

```
void melde(int messwert);
```

erzeugt auf der Standardausgabe eine Meldung:

Messwert: Zeit

(z. B. `67: 2007/07/30 07:35`)

Zum Abfragen der aktuellen Zeit steht Ihnen eine Funktion

```
char *get_time();
```

zur Verfügung, die einen Zeiger auf einen String liefert, in dem die aktuelle Zeit mit Datum im richtigen Format abgelegt ist.

Für die Ausgabe benutzen Sie am besten die Funktion printf.
Schnittstelle: `int printf(char *format, arg);`
Das Format-Kürzel **%d** gibt ganze Zahlen formatiert aus, das Kürzel **%s** wird zur Ausgabe von Zeichenketten genutzt.

Programmieren Sie eine weitere Funktion, die eine Statistik über die Messwerte führt

```
void statistik(int messwert);
```

.

Diese Funktion soll in einem Feld Zähler für die Wertebereiche 0-9, 10-19, 20-29, ..., 240-249, 250-255 verwalten und für jeden Messwert jeweils den Zähler des Wertebereichs inkrementieren, in dem der Messwert liegt.

Sie dürfen dieses Blatt zur besseren Übersicht bei der Programmierung heraustrennen!

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

```
/* Funktion main */
int main(int argc, char **argv) {            /* 1 */
   volatile unsigned char *sensor =
         (volatile unsigned char *)0x19;     /* 2 */
   unsigned char lastmw;

   while (1) {           /* 1 */
    /* sicherstellen dass IMMER der erste Wert ausgegeben wird
     Problem bei einfach "if (lastmw != *sensor) {... */  1P */
      lastmw = *sensor;         /* 1 Wert merken */
      melde(lastmw);            /* 1 */
      statistik(lastmw);        /* 1 */
      while (lastmw == *sensor) ; /* 2 Pollen + Warten*/
   }
   return 0;
/* Funktion melde */
void melde(int messwert) {     /* 1 incl. Prototyp */
   printf("%d: %s\n", messwert, get_time());   /* 2 */
}

/* Funktion statistik */
void statistik(int messwert) {     /* 1 incl. Prototyp */
   static int stat[26];     /* 2 */
   stat[messwert/10]++;     /* 2 */
}
```

10

3

5

M: 8

Aufgabe 3: (18 Punkte)

In einer Gepäckabfertigungsanlage werden über ein Förderband Gepäckstücke angeliefert. Eine Lichtschranke erkennt die Gepäckstücke und meldet diese Ereignisse per Interrupt an einen Mikrocontroller (d. h. der Sensor der Lichtschranke liefert Pegel = 0 wenn kein Licht einfällt und Pegel = 1 wenn Licht einfällt).

Der Mikrocontroller soll für jedes Gepäckstück einen Aufkleber mit der laufenden Nummer des Gepäckstücks ausdrucken. Das Problem dabei ist, dass es kurze Phasen geben kann, in denen Gepäckstücke schneller ankommen können, als die Ausdrucke erfolgen. Es soll sichergestellt sein, dass immer genau die richtige Zahl von Aufklebern ausgedruckt wird - es dürfen also Gepäckstücke nicht ohne Aufkleber bleiben (von kurzen Verzögerungen abgesehen) und es dürfen auch nicht zu viele Aufkleber gedruckt werden.

a) Skizzieren Sie in programm-ähnlicher Form den Ablauf in dem Hauptablauf des Mikrocontrollerprogramms und in der Interruptbearbeitung (es kommt nicht auf korrekte Syntax an).

Hauptablauf	Interruptbearbeitung
zaehler = 0; z2=0; 1	
	zaehler++; 1
while (1) { 1	
if(zaehler > 0) 1 nix da	
print_Aufkleber(++z2); 1	
cli(); /* (d) */ ->d	
zaehler --; 1	
sei(); /* (d) */ ->d	
}	
}	
6	

6

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

b) Welche Arten von Interrupt-Steuerung kennen Sie und welche Art würden Sie bei diesem Szenario einstellen. Warum wäre die andere Art problematisch? Begründen Sie Ihre Entscheidung.

pegel, flanken 2

pegel problem weil koffer zu lang => mehrere IRQ 1

=> flanken high->low = Gepaeckanfang 1

4

4

c) Welches grundlegende Problem muss bei der Problemlösung zwischen Interruptbearbeitung und Hauptablauf bedacht werden, damit wirklich die korrekte Zahl von Aufklebern gedruckt wird (detaillierte Beschreibung - welche Stelle und welche Ursache!) ?

Nebenläufigkeit der Interruptbearbeitung

gemeinsame Variable

-- ist nicht atomar

3

3

d) Was muss in der Problemlösung getan werden, um diese grundlegende Problem zu vermeiden - tragen Sie - falls nicht schon geschehen - die entsprechenden Aktionen in Ihre Lösung ein und markieren Sie sie mit (d)?

bei -- duerfen keine interrupts auftauchen 1

=> sperren 1

cli + sei 1, Markieren, richtige Stellen 2

uebel wenn print mit im gesperrten Abschnitt!

5

12