Problem 1: (14 Points)

For the single-choice questions in this problem, only **one** correct answer must be clearly marked with a cross. The correct answer is awarded the indicated number of points.

If you want to correct an answer, please cross out the incorrect answer with three horizontal lines (☒) and mark the correct answer with a cross.

Read the question carefully before you answer.

a) Which of the following statements about the C preprocessor is correct? 2 Points

- ☐ After compiling and binding, C programs must be post-processed by the preprocessor in order to resolve macros.
- ☒ The preprocessor is a software component that performs text transformations on the source code, which are then translated by a C compiler.
- ☐ The preprocessor optimizes macros using pointer arithmetic.
- ☐ The preprocessor is a software component that replaces Java classes with C functions, which are then translated by a C compiler.

b) The following enumeration is given: 2 Points

```
enum SPRACHE {Java, Python, C};
```

Which of the following statements is correct?

- ☐ The value of `C` is unknown; the compiler assigns a random but unique value to each `enum` element at compile time.
- ☐ The value of `C` is `3`.
- ☐ The compiler reports an error because no value has been assigned to the `enum` elements.
- ☒ The value of `C` is `2`.

c) The following macro definition can be found in the AVR library: 2 Points

```
#define PINA (*(volatile uint8_t *)0x39)
```

Which of the following statements regarding the use of the `volatile` keyword is correct in this case?

- ☐ The keyword `volatile` allows the compiler to perform better optimizations.
- ☒ If port A is configured as an input, the value of `PINA` could change at any time. `volatile` instructs the compiler to always read the current value from `PINA`.
- ☐ The `volatile` keyword enables safe access to individual bits of the register.
- ☐ The `volatile` keyword ensures that access to `PINA` is synchronized with interrupts.

d) Which statement on the term „polling“ is correct? 2 Points

- ☐ A concept for processing interrupts.
- ☒ When a program periodically queries a peripheral device to determine whether data or status changes are pending.
- ☐ The periodic raising of a voltage level to signal a certain status to a device.
- ☐ When a program masks interrupts to access critical data.

e) Given are the following two C source files to be compiled: 2 Points

```
// foo.c:
uint8_t a = 42;
// main.c:
extern uint32_t a;
```

Which statement on the subject of compiling is correct?

- ☐ When compiling, a total of 5 bytes (1 byte + 4 bytes) of memory are reserved for `a`.
- ☒ When compiling, a single 1 byte of memory are reserved for `a`.
- ☐ When compiling, a total of 4 bytes of memory are reserved for `a`.
- ☐ In any case, the compiler detects a type error and exits with an error.

f) What is the difference between the following global variables defined in a C file? 2 Points

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

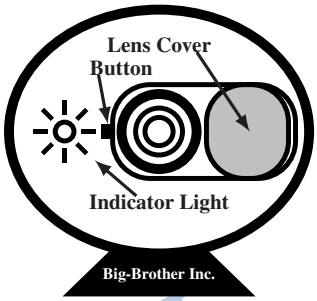
- ☐ The variable `b` can only be accessed from functions that have also been declared with the keyword `static`.
- ☐ The variable `a` can only be accessed by functions in the same file, while the variable `b` can also be accessed by functions in other modules of the program.
- ☐ Variable `b` can only be assigned once and then remains constant.
- ☒ The variable `b` can only be accessed by functions in the same file, while the variable `a` can also be accessed by functions in other modules of the program.

g) Which statement about *signals* is correct? 2 Points

- ☐ No concurrency problems can occur with signals.
- ☐ Different signals can never interrupt each other.
- ☒ Signals can be blocked using `sigprocmask` during the execution of critical areas.
- ☐ The default behavior when receiving a signal is to abort the process.

Problem 2: Webcam (0 Points)

You may detach this page for a better overview during programming!



Implement the control of an activity indicator within a webcam to protect the privacy of its users. To prevent unwanted spying, it has a lens cover that can be pushed in front of the lens if wanted. Inside the webcam is a stop button that registers every opening/closing of the cover. When the cover is opened, the button is released and the webcam is active. During recording, a red indicator light now also lights up at intervals of about one second. To increase the image quality, the ambient brightness should also be measured, processed, and communicated to the actual camera controller. When the cover is closed and, thus, the button is pressed, the indicator light should be switched off and the ambient brightness should no longer be queried.

In detail, your program should work as follows:

- Initialize the hardware in the `void init(void)` function. Do not make any assumptions about the initial state of the hardware registers.
- The input `PD2` (interrupt 0) is connected to the button. A rising edge occurs when the lens cover is opened and a falling edge occurs when it is closed. Do not make any assumptions about the initial position of the lens cover.
- An 8-bit timer should be used for timing. Configure the timer so that it triggers an interrupt every millisecond. The main program should be informed every `500ms` (see macro `ACTIVATION_INTERVAL`) when the required time interval has elapsed.
- An 8-bit timer should be used for the timing. Configure this in such a way so that it triggers an interrupt every millisecond. The main program should trigger an interrupt every `500ms` (see macro `ACTIVATION_INTERVAL`) when the required time interval has elapsed.
- During recording (= lens cover opened), the state of the indicator light at output `PB0` should be toggled every `500ms`.
- The ambient brightness should also be determined at the same time interval. To do this, first call the function `int16_t sb_adc_read(ADCDEV dev)` for the `PHOTO` device to read the value h of the phototransistor as an unsigned 10-bit integer. Interrupts must be disabled during the function invocation.
- Determine the (approximated) decadic logarithm of the measured value h and pass the result to the external function `void set_iso(uint16_t value)`, which encapsulates the actual control of the camera.
- Implement the auxiliary function `uint16_t log10(uint16_t value)`, which implements the following resource-aware approximation:

$$\log_{10} h = \frac{\log_2 h}{\log_2 10} \approx \left\lfloor \frac{\lfloor \log_2 h \rfloor}{3} \right\rfloor \tag{1}$$

  The rounded binary logarithm $\lfloor \log_2 h \rfloor$ corresponds to the position of the most significant set bit in h. Example: $\lfloor \log_2 0b100 \rfloor = 2$. (Special case: $\lfloor \log_2 0b0 \rfloor = 0$).
- Do not use **any** floating point numbers or other math library functions.
- Make sure that the microcontroller enters a sleep mode as often as possible.

Information about the hardware

You may detach this page for a better overview during programming!

Button: interrupt line connected to **PORTD**, pin 2

- Rising edge: lens cover is opened, recording begins
- Falling edge: lens cover is closed, recording ends
- Configure pin as input: set the corresponding bit in the **DDRD** register to 0
- Activate internal pull-up resistor: set the corresponding bit in the **PORTD** register to 1
- External interrupt source **INT0**, ISR vector macro: **INT0_vect**
- Masking/Unmasking the interrupt is done by setting/clearing the **INT0** bit in the **EIMSK** register

Configuration of the external interrupt source **INT0** (bits in register **EICRA**)

interrupt 0		description
ISC01	ISC00	
0	0	interrupt at low level
0	1	interrupt at either edge
1	0	interrupt on falling edge
1	1	interrupt on rising edge

Indicator light for recording: output at **PORTB**, pin 0

- Signal active recording: set to LOW to switch on the LED, otherwise set to HIGH
- Configure pin as output: set the corresponding bit in the **DDRB** register to 1
- Initially switch off indicator light: set corresponding bit in **PORTB** register to 1

Timer (8-bit): **TIMER0**

- The overflow interruption is to be used (ISR vector macro: **TIMER0_OVF_vect**).
- The most resource-efficient prescaler (*prescaler*) is 64, which causes the 8-bit counter **TCNT0** to overflow every $1ms$ at the 16MHz CPU clock (sufficiently accurate).
- Activating/deactivating the interrupt source is done by setting/deleting the **TOIE0** bit in the register **TIMSK0**.

Configuration of the frequency of the timer **TIMER0** (bits in register **TCCR0B**)

CS02	CS01	CS00	description
0	0	0	timer off
0	0	1	CPU clock
0	1	0	CPU clock / 8
0	1	1	CPU clock / 64
1	0	0	CPU clock / 256
1	0	1	CPU clock / 1024
1	1	0	Ext. clock (falling edge)
1	1	1	Ext. clock (rising edge)

Complete the following code structure so that a fully compilable program is created.

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

#define ACTIVATION_INTERVAL 500
#define LD_10 3

extern void set_iso(uint16_t value);

// Function Declarations, Global Variables, etc.
static void init(void); // Punkte dort
static uint16_t log10(uint16_t value);
                                          //
// Ignore first iteration of event loop. Alternatively,
// read PIN register in main (see below).
static volatile uint8_t event_shutter = 1;
static volatile uint8_t event_timer = 0;
// End Function Declarations, Global Variables, etc.

// Interrupt Service Routines
ISR(TIMER0_OVF_vect) {
  static uint16_t counter = 0;
  counter++;
  if (counter == ACTIVATION_INTERVAL) {
    counter = 0;
    event_timer = 1;
  }
}
ISR(INT0_vect) {
  event_shutter = 1;
}
// End Interrupt Service Routines
```

0

0

D: 0

```
// Function main
void main(void) {
// Initialization and Local Variables
  init();
    // It's also ok to read the PIN register here.
  uint8_t active = 0;


// Event Loop
  while (1) {
    cli(); // Enter critical section
    while (!event_timer && !event_shutter) {
      sleep_enable();
      sei(); // sleep_cpu() *directly* after sei()
      sleep_cpu();
      sleep_disable(); // Optional
      cli(); // Enter critical section
    }
    sei();
```

0

0

```
// Process Events
    if (event_shutter) { // Shutter *before* Timer
      event_shutter = 0;

      active = (PIND & (1 << PD2)) != 0;
      if (!active) {
        PORTB |= (1 << PB0);
      }
    }


    if (event_timer) {
      event_timer = 0;

      if (active) {
        PORTB ^= (1 << PB0);

        cli();
        int16_t sensor = sb_adc_read(PHOTO);
        sei();

        set_iso(log10(sensor));
      }
    }
  }
}

// End main
```

0

0

M: 0

```
// Logarithm to Base 10
static uint16_t log10(uint16_t value) {
  uint8_t log2 = 0;
  for (uint8_t i = 0; i < 16; i++) {
    if (value & (1 << i)) {
      log2 = i;
    }
  }

  return log2 / LD_10;
}
```

0

```
// End Logarithm to Base 10
```

L: 0

```
// Initialization Routine
static void init(void) {
  // Configure prescaler (64)
  TCCR0B &= ~(1 << CS02);
  TCCR0B |= (1 << CS01) | (1 << CS00);

  // Activate timer interrupts
  TIMSK0 |= (1 << TOIE0);

  // Configure PB0 (RED0) as output
  DDRB |= (1 << PB0);
  PORTB |= (1 << PB0);

  // Configure PD2 (Button0) as input
  DDRD &= ~(1 << PD2);
  PORTD |= (1 << PD2);

  // Configure interrupts for PD2 at either edge
  EICRA |= (1 << ISC00);
  EICRA &= ~(1 << ISC01);

  // Activate Button0 interrupts
  EIMSK |= (1 << INT0);
}
// End Initialization Routine
```

0

I: 0

Problem 3: alt (1 Point)

Course supervisors at FAU usually receive many emails from students asking for help. As courses progress at a fast pace, such emails are often outdated after a few days and a reply is no longer necessary. Help work-shy supervisors by writing a program `alt` that deletes all files in a given directory that are older than one week.

```
$> ./alt /home/tutor/Maildir/spic
```

The program should work in detail as follows:

- The program checks at the beginning whether exactly one parameter has been passed. If this is not the case, it issues a corresponding error message and exits.
- If the program was called correctly, the passed directory is searched (not recursively) using `opendir()` and `readdir()`.
- The modification time (`st_mtime`) is queried for each regular file found using `stat()`.
- If the file is older than 7 days, it should be removed using `unlink()`.
- After successful execution, the program ends with the status code `EXIT_SUCCESS`.
- If errors occur, an error message should be displayed and the program should end with an error status (`EXIT_FAILURE`).
- Make sure to release allocated resources again (`free()`, `closedir()`, ...).
- Also implement the auxiliary function `time_t calc_time_limit(void)`, which determines the timestamp for 7 days ago from today:
  - Time is counted in UNIX-like systems in seconds since 01.01.1970.
  - You can use `time(NULL)` to return the timestamp for the current second.
  - Assume standard days (24 hours per day, 60 minutes per hour, 60 seconds per minute).
  - Call `calc_time_limit` only once during a program run in order to have a consistent basis for comparison.

Ensure correct error handling of the functions used. Error messages should generally be sent to `stderr`. The predefined functions `die()` (`errno` set) and `err()` (`errno` not set) can be used for compact error handling.

Complete the following code structure so that a fully compilable program is created.

```
#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

static void die(const char *s) {
  perror(s);
  exit(EXIT_FAILURE);
}

static void err(const char *s) {
  fputs(s, stderr);
  exit(EXIT_FAILURE);
}

// Function calc_time_limit
static time_t calc_time_limit(void) {
  time_t now = time(NULL);
  time_t seven_days = 7*24*60*60;
  return now-seven_days;
}
```

C: 0

```
// Function main
int main(int argc, char *argv[])
{
  // Check arguments + error handling
  if (argc != 2) {
    err("usage");
  }

  char *dirpath = argv[1];

  // Open directory
  DIR *dirp = opendir(dirpath);
  if (!dirp) {
    die("opendir");
  }

  time_t limit = calc_time_limit();

  errno = 0;
  struct dirent *de = readdir(dirp);
  while (de != NULL) {
    size_t len = strlen(dirpath) + strlen(de->d_name) + 2;
    char *filename = malloc(len);
    if (filename == NULL) {
      die("malloc");
    }

    // Concatenate file path
    strcpy(filename,dirpath);
    strcat(filename,"/");
    strcat(filename,de->d_name);
```

```
    // Check timestamp
    struct stat sb;
    if (stat(filename, &sb) < 0) {
      die("stat");
    }

    // 1.0P Check
    if ((sb.st_mode & S_IFREG) && (sb.st_mtime < limit)) {
      if (unlink(filename) < 0) {
        die("unlink");
      }
    }

    free(filename);

    errno = 0;
    de = readdir(dirp);
  }

  if (errno != 0) {
    die("readdir");
  }

  closedir(dirp);

  return EXIT_SUCCESS;

}
// End main
```

1

L: 1

Problem 4: Memory Layout (11 Points)

Note: Read the task first - a complete understanding of the program is not necessary to complete the problem.

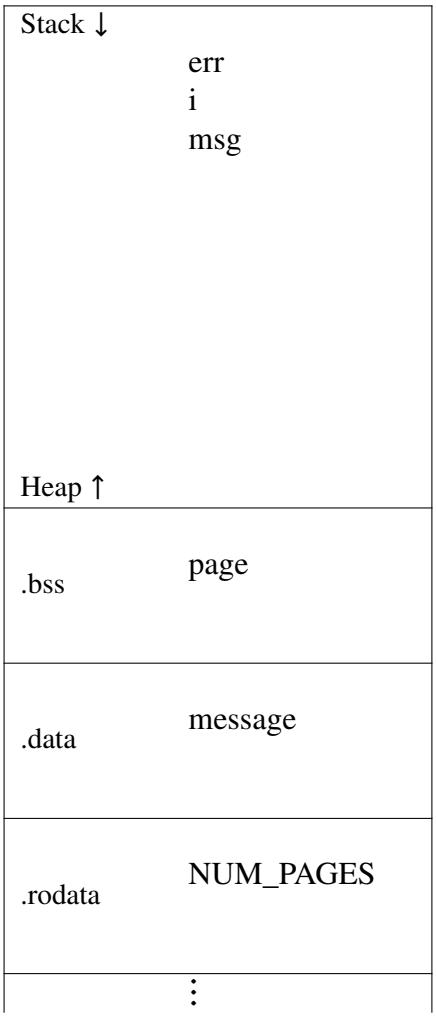
```
#include <stdint.h>
#include <display.h>
#include <timer.h>
#include <avr/interrupt.h>

static const uint8_t NUM_PAGES = 8;
static char message[] = "SPiC Exam!";

static void show_message(char msg[]) {
    static int page = 0;
    sb_display_showStringWide(page, 0, msg);
    page = (page + 1) % NUM_PAGES;
}

void main(void) {
    sei();
    int8_t err = sb_display_enable();
    if (err) {
        while(1) { /* ... */ }
    }
    for (uint8_t i = 0; i < NUM_PAGES; i++) {
        show_message(message);
        sb_timer_delay(5000);
    }
    while(1);
}
```

Memory Layout (simplified):



a) Complete the (simplified) memory layout:

5 Points

1) Add the terms *stack* and *heap* and their direction of growth in the figure. (2 Points)

2) Assign all variables occurring in the source code to the corresponding memory segments. (3 Points)

Note: Order of stack variables is not taken into consideration.

b) Upon starting the microcontroller, how must the RAM memory be prepared before the actual program execution in `main()` can begin? (2 Points)

- Copy .data from non-volatile memory (flash/ROM) to RAM
- Zero .bss segment

c) The following table contains four pointer types. For all types, specify whether the dereferenced value and the pointer are mutable (i.e., not constant) (**yes** or **no** in each case). If a data type is not possible in C, please check the column **syntax error**. (4 Points)

	Value mutable	**Pointer mutable**	**Syntax error**
`uint8_t *`	yes	yes	
`const uint8_t *`	no	yes	
`uint8_t * const`	yes	no	
`const uint8_t * const`	no	no	

Note: No points if someone checked 'Syntax error' and another column

Problem 5: Concurrency (3 Points)

The following descriptions should be short and concise (keywords, short sentences).

a) Which steps **in hardware** typically comprise the processing of an interrupt on a microcontroller? (3 Points)

1. Device signals interrupt,
   Programm will be interrupted
2. Further interrupts are blocked
3. Register values will be saved
4. ISR will be determined
5. ISR will be executed
6. ISR terminates, register values will be restored,
   program resumes

b) The use of interrupts can lead to various concurrency problems. Name and describe **two** examples of such problems. In addition, describe how to deal with them to ensure the correct program behaviour. (0 Points)

Lost-update:
Register contains value, ISR updates the value and
restores the register upon returning

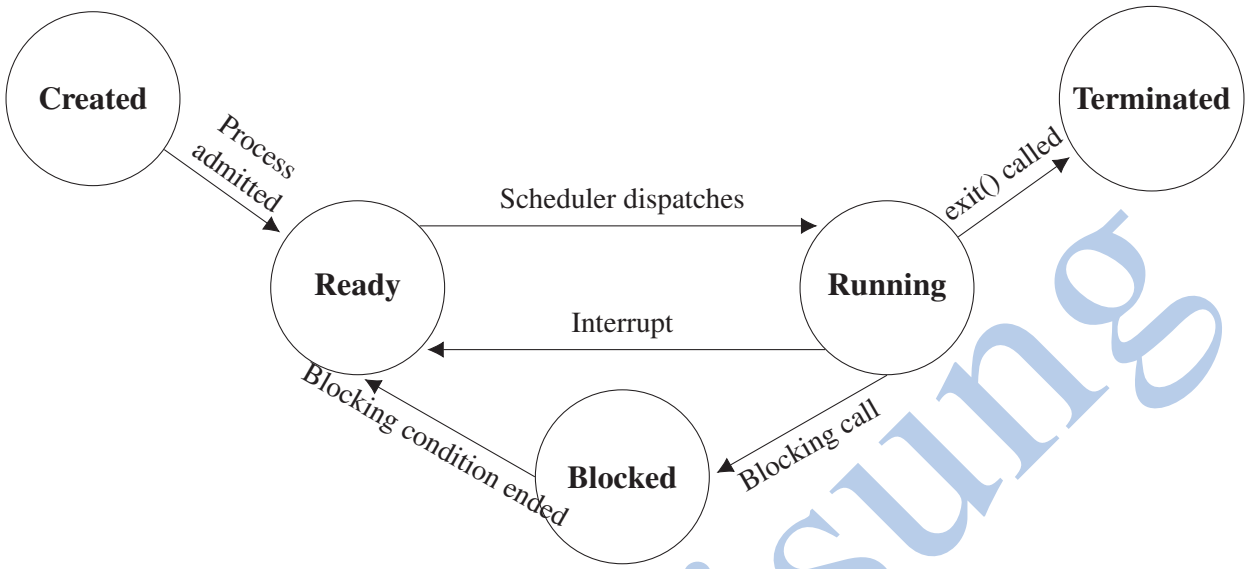
Read-Write:
Value exceeds maximum representable value
of a single register. Value will be updated.
ISR interrupts update right in between and
corrupts the result of the calculation
(e.g. overflowing operation)

or alternatively: Lost wakeup:
Interrupt right between checking the sleep condition
and going to sleep

Solution: Mask interrupts briefly

Problem 6: Processes (0 Points)

a) Complete the following graph for the different Linux process states with the corresponding state transitions. Each node corresponds to a process state and each edge corresponds to a state transition. Make sure to label **all** nodes and edges that have not yet been labeled. (0 Points)



Note:
Answers must not exactly correspond to solution.
Comparable answers are ok.

b) Name one common problem each when using a **Spin Lock** and a **Sleeping Lock**.
(0 Points)

Spin Lock: For long critical section,
CPU cycles are wasted

Sleeping Lock: For short critical section,
large overhead related to blocking/resuming