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) The following program code is given: 2 Points

```
int32_t x[] = {-1, 7, -3, 5};
int32_t *y = &x[3];
y -= 2;
```

What value does the dereferencing of `y` (i.e. `*y`) return after the program code has been executed?

- ☐ An error occurs at runtime.
- ☐ 3
- ☐ -5
- ☐ 7

b) 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 `volatile` keyword ensures that access to `PINA` is synchronized with interrupts.
- ☐ The `volatile` keyword enables safe access to individual bits of the register.
- ☐ 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`.

c) The following program code is given: 2 Points

```
#define SUB(a,b) a-b
#define ADD(a,b) a+b
```

What is the result of the following expression: `2 * SUB(2, ADD(3, 4))`

- ☐ -10
- ☐ 5
- ☐ -3
- ☐ 6

d) 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 compiler reports an error because no value has been assigned to the `enum` elements.
- ☐ The value of `C` is `3`.
- ☐ The value of `C` is `2`.

e) What is the purpose of the **`#ifdef`** construct in C? 2 Points

- ☐ It can be used to ensure that a program section returns a defined result.
- ☐ It can be used to hide parts of the program during translation
- ☐ It checks whether the variables specified afterwards have been defined.
- ☐ It can be used as an alternative to `if` statements.

f) Given the following C program snippet that uses a variable `foo` of type `uint8_t`: 2 Points

```
foo &= ~0xaa;
foo ^= 0xaa;
```

Which statement about `foo` is correct after executing the statements?

- ☐ The least significant bit in `foo` is `1`.
- ☐ The most significant bit in `foo` is `1`.
- ☐ No statement can be made about the state of the most significant bit of `foo`.
- ☐ The first two characters in the string `foo` are `"aa"`.

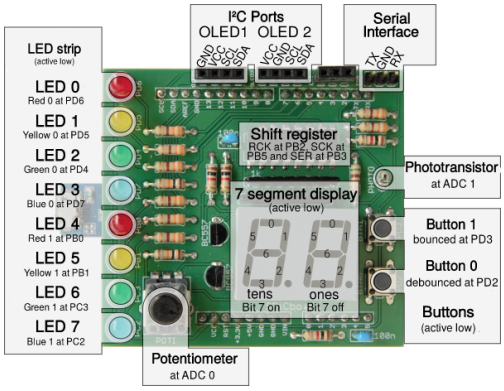
g) In which of the following situations is a running process transferred to the blocked state? 2 Points

- ☐ The scheduler dispatches the process to another CPU.
- ☐ A child process of the process terminates.
- ☐ The process is reading a file and the corresponding data block is not yet available in main memory.
- ☐ The operating system cannot transfer a running process to the blocked state because the process would otherwise trigger a trap.

Problem 2: Boardtest (30 Points)

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

Implement the test program for the SPiCBoard. To test the individual functionalities, the program should be able to switch between three different test modes (see `enum Mode`) using `BUTTON0`: In `TEST_POTI` mode, the number of switched-on LEDs is determined by the potentiometer. In `TEST_PHOTO` mode, the number of switched-on LEDs shows the ambient brightness. In `TEST_SEG` mode, the seven-segment display should count down repeatedly from 99 to 0.



The program works in detail as follows:

- Initialize the hardware in the function **`void`** `init(`**`void`**`)`. Do not make any assumptions about the initial state of the hardware registers.
- The input `PD2` (interrupt 0) is connected to the button. A falling edge occurs exactly when the button is pressed and a rising edge occurs when it is released again. You can assume that the button is not pressed.
- When the button is pressed, the registered `ISR(INT0_vect)` signals the corresponding event to the `main` function using a module-local variable. The next test mode is selected according to the sequence in `enum`. After the last test mode, the first mode is started again. All test-related state is reset with each mode transition: Deactivating the seven-segment display using `sb_7seg_disable(void)`, switching off the LEDs using `sb_led_setMask(uint8_t)`, ...
- An 8-bit timer should be used for timing. Configure the most resource-efficient prescaler and trigger an event once per millisecond. You will find details on the next page.
- If the configured count value of the timer is reached, only the event should be signaled in `ISR(TIMER0_OVF_vect)` using a module-local variable. The internal counter of the elapsed milliseconds should then be incremented in the `main` function.
- For the test modes `TEST_POTI` and `TEST_PHOTO`, the respective ADC device (`POTI` or `PHOTO`) is to be read using `sb_adc_read`. The 8 most-significant bits of the read 10-bit integer value are then to be interpreted as an unsigned 8-bit value. The provided function `sb_led_showLevel` is used to switch on the number of LEDs corresponding to this value.
- For the test mode `TEST_SEG`, the seven-segment display should count down repeatedly from 99 to 0. Wait $500ms$ between the individual calls of `sb_7seg_showNumber`. Use an internal counter value for this, which is incremented every millisecond.
- Do not use any button/timer functionality of the libspicboard (`button.h`/`timer.h`).
- Do not use floating point numbers or other math library functions.
- Make sure that the microcontroller is in sleep mode as often as possible.

Information about the hardware

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

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

- Falling edge: button is pressed
- Rising edge: button is released
- Configure pin as input: corresponding bit in the **DDRD** register to 0
- Activate internal pull-up resistor: corresponding bit in the **PORTD** register to 1
- External interrupt source **INT0**, ISR vector macro: **INT0_vect**
- Activating/deactivating the interrupt source is done by setting/clearing the **INT0** bit in the **EIMSK** register

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

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

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 16ṀHz CPU clock (sufficiently accurate).
- Activating/deactivating the interrupt source is done by setting/clearing 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 skeleton 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>
#include <led.h>
#include <7seg.h>

enum Mode { TEST_POTI = 0, TEST_PHOTO = 1, TEST_SEG = 2 };

// Allows the array of LEDs to be used as a (fill) level, progress, or
// similar indicator. The 8 LEDs are used to display the current level
// as a ratio of the maximum value (<=255) in 9 steps.
//
// level: the current level to be shown
// max: the maximum possible value for level
extern int8_t sb_led_showLevel(uint8_t level, uint8_t max);

// Function Declarations, Global Variables, etc.
```

D:

```
// End Function Declarations, Global Variables, etc.

// Interrupt Service Routines
// ISRs **must** only set an event variable
```

D:

```
// End Interrupt Service Routines
```

D:

```
// Function main

    // Initialization and Local Varibables

    // Event Loop
```

```
// Processing of Button Event
```

```
// Processing of Timer Event
```

```
        // Test required Mode
```

```
// End main
```

M:

```
// Initialization Function
```

```
// End Initialization Function
```

I:

Problem 3: SMS (19 Points)

To ensure smooth SLP-exercise operation, its submission system is continuously monitored. To be notified even in the event of an Internet failure, serious problems are reported by SMS (**S**hort **M**essage **S**ervice). Implement a program `sms` that reads in a telephone number and text message via the standard input and then sends it.

```
$ ./sms
Enter phone number:
091318527276
Enter SMS:
i4spic.cs.fau.de is broken...
```

- The program initially opens the file `/sys/sms/spool` (see `OUTPUT_FILE`) for writing.
- If the file is opened successfully, the prompt `"Enter␣Phone␣Number:\n"` appears on the standard output and the phone number is read from the standard input. The maximum length of a telephone number is `LEN_NUM` characters (excluding the final `'\n'` and `'\0'`).
- The external function `sanitize` (see declaration on following page) is used to check whether the maximum character length has been exceeded. The processed string always ends with a `'\0'` byte. The function `int check_phone_number(const char*)` implemented by you is be used to check whether only valid digits (`'0'` - `'9'`) have been transferred. If one of the two checks fails (return value: `-1`), the program exits with an error message. In the case of success, `check_phone_number` returns `0`.
- The input phone number is then written to file in the format `"{{{␣<number>␣}}}\n"`.
- The text of the short message should be processed after the prompt `"Enter␣SMS:"`. The maximum length of an SMS is `LEN_SMS` characters (excluding the final `'\0'`). Read the message text in a loop from the standard input and check repeatedly using `sanitize` whether the maximum character length has been exceeded. Write the read text in the format `"{{{␣<text>␣}}}\n"` to the opened file. If this limit has been exceeded, continue with the loop. Otherwise, use `EXIT_SUCCESS` to signal successful program execution.

Ensure correct error handling of the functions used. Error messages should generally be sent to `stderr`.

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

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LEN_NUM 12
#define LEN_SMS 160

const char OUTPUT_FILE[] = "/sys/sms/spool";

// Sanitize input string before processing.
//
// Removes trailing newlines, escapes all special meaning characters,
// and checks for valid input length. Upon successful completion,
// sanitize returns 0. If input is too short (e.g., "") or
// too long (see max_input_len), -1 is returned.
//
// The processed string will always end with a '\0' byte.
extern int sanitize(const char *input, size_t max_input_len);

static void die(const char message[]) {
  perror(message);
  exit(EXIT_FAILURE);
}

static void err(const char message[]) {
  fprintf(stderr, "%s\n", message, stderr);
  exit(EXIT_FAILURE);
}

// Function check_phone_number
```

B:

```
// Function main

  // Open OUTPUT_FILE for writing

  // Read phone number from stdin

  // Check validity of phone number

  // Write phone number to OUTPUT_FILE
```

```
  // Read sms from stdin
```

```
    // Check validity of sms text
```

M:

Problem 4: Concurrency (8 Points)

a) Consider the following code snippet. Describe the lost-update problem using the given example! (4 Points)

```
1  static volatile uint8_t counter = 0;
2  ISR(INT0_vect) {
3      counter++;
4  }
5
6  void main(void) {
7      while(1) {
8          if(counter > 0) {
9              counter--;
10             // Process key press
11             ...
12         }
13         ...
14     }
15 }
```

You will find the next subexercise on the following page!

b) Consider the following code snippet. Describe the lost-wakeup problem and its consequences using the given example! (3 Points)

```
1  ISR(TIMER1_COMPA_vect) {
2      event = 1;
3  }
4
5  void main(void) {
6      sleep_enable();
7      event = 0;
8      while(!event) {
9          sleep_cpu();
10     }
11     sleep_disable();
```

---

c) Describe hwo the lost-update/lost-wakeup problem in these examples can be solved! (1 Point)

Problem 5: Memory Organisation (12 Points)

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

a) The following program is executed on an 8-bit AVR/ATmega32 microcontroller.
Complete the properties of the named variables and expressions in the table on the next page. (6 Points)

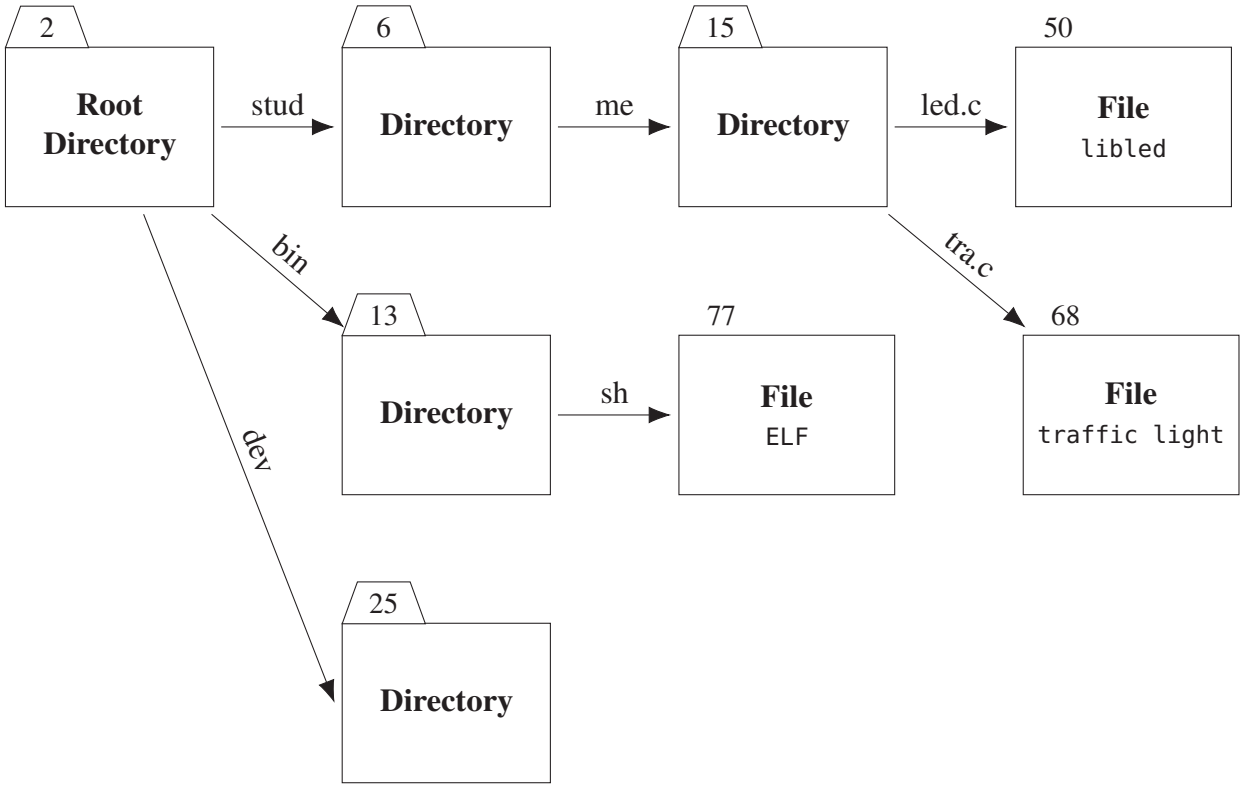
```
static const char *text = "C i5 c00l";
const uint8_t BUFFER_SIZE = 3;

static volatile uint8_t move_text;

static void move_text_timer_callback(void) {
    move_text = 1;
}

void main(void) {
    sei();
    static uint8_t time = 400;

    sb_timer_setAlarm(
        move_text_timer_callback,
        time, time
    );

    const char *text_start = text;
    move_text = 1;

    while(42) {
        if(move_text){
            move_text = 0;
            if((*text_start) == '\0') {
                text_start = text;
            }

            char buffer[BUFFER_SIZE];
            buffer[0] = text_start[0];
            buffer[1] = text_start[1];
            buffer[2] = '\0';

            text_start++;
        }
        ...
    }
}
```

Variable	Scope	Lifespan	Section	Memory consumption in bytes
text	module		.data	
BUFFER_SIZE	program	program		
move_text		program	.bss	1
time				1
text_start			stack	2
buffer	end of block	end of block		

b) When does static and when does dynamic allocation take place? Assign the memory sections mentioned above accordingly. What is the relationship with regard to the lifetime of the variables? (6 Points)

Problem 6: Filesystems (7 Points)

A file system enables the structured storage of data. A directory tree is shown below. Directories and files are marked with a labeled rectangle, a link with a labeled arrow.



Complete the associated, simplified management information. Use the scheme already provided as a reference. Enter **also** the entries for `.` and `...`