Lab Assignment 3.

Task 1: Preparation task.

Data types in C and their ranges.

Data type	Number of bits	Range	Description
uint8_t	8	0, 1, ..., 255	Unsigned 8-bit integer
int8_t	8	-128,-127...127	Signed 8-bit integer
uint16_t	16	0,1...65538	Unsigned 16-bit integer
int16_t	16	-32768...32767	Signed 16-bit integer
float	32	-3.4e+38 ,..., 3.4e+38	Single-precision floating-point
void	*	*	Used for pointers

*The number of bits and the range for void varies with the system type

Function phototype Example

```
#include <avr/io.h>

// Function declaration (prototype)
uint16_t calculate(uint8_t, uint8_t );

int main(void)
{
    uint8_t a = 156;
    uint8_t b = 14;
    uint16_t c;

    // Function call
    c = calculate(a, b);

    while (1)
    {
    }
    return 0;
}

// Function definition (body)
uint16_t calculate(uint8_t x, uint8_t y)
{
    uint16_t result;    // result = x^2 + 2xy + y^2
```

```
    result = x*x;
    result += 2*x*y;
    result += y*y;
    return result;
}
```

Task 2: GPIO library.

1. **Listing of library source file `gpio.c`,**

```
/*
 * gpio.c
 *
 * Created: 10/7/2020 11:36:30
 *  Author: masau
 */
/**********************************************************************
 *
 * GPIO library for AVR-GCC.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2019-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 *
 **********************************************************************/

/* Includes ----------------------------------------------------------*/
#include "gpio.h"

/* Function definitions ----------------------------------------------*/
void GPIO_config_output(volatile uint8_t *reg_name, uint8_t pin_num)
{
     *reg_name = *reg_name | (1<<pin_num);   // Data Register
}

/*-------------------------------------------------------------------*/
void GPIO_config_input_nopull(volatile uint8_t *reg_name, uint8_t pin_num)
{
       *reg_name = *reg_name & ~(1<<pin_num);  // Data Direction Register
       *reg_name++;                            // Change pointer to Data Register
```

```
        *reg_name = *reg_name & ~(1<<pin_num);    // Data Register

}

/*-------------------------------------------------------------------*/
void GPIO_config_input_pullup(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name & ~(1<<pin_num);  // Data Direction Register
    *reg_name++;                            // Change pointer to Data Register
    *reg_name = *reg_name | (1<<pin_num);   // Data Register
}

/*-------------------------------------------------------------------*/
void GPIO_write_low(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name & ~(1<<pin_num);
}

/*-------------------------------------------------------------------*/
void GPIO_write_high(volatile uint8_t *reg_name, uint8_t pin_num)
{
        *reg_name = *reg_name | (1<<pin_num);
}

/*-------------------------------------------------------------------*/
void GPIO_toggle(volatile uint8_t *reg_name, uint8_t pin_num)
{
        *reg_name = *reg_name ^ (1<<pin_num);
}

/*-------------------------------------------------------------------*/
uint8_t GPIO_read(volatile uint8_t *reg_name, uint8_t pin_num)
{
        if(bit_is_clear(*reg_name, pin_num))
        {
                return 0;
        }
        else
        {
                return 1;
        }
}
```

2. C code of the application `main.c`

```
/*
 * gpio.c
 *
 * Created: 10/7/2020 11:13:50
 * Author : masau
 */

/*****************************************************************
 *
 * Alternately toggle two LEDs when a push button is pressed. Use
 * functions from GPIO library.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2019-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 *
 *****************************************************************/

/* Defines -----------------------------------------------------------*/
#define LED_GREEN   PB5     // AVR pin where green LED is connected
#define LED_RED     PC0     // AVR pin where RED LED is connected
#define BTN         PD0     //
#define BLINK_DELAY 500
#ifndef F_CPU
#define F_CPU 16000000      // CPU frequency in Hz required for delay
#endif

/* Includes ----------------------------------------------------------*/
#include <util/delay.h>     // Functions for busy-wait delay loops
#include <avr/io.h>         // AVR device-specific IO definitions
#include "gpio.h"           // GPIO library for AVR-GCC

/* Function definitions ----------------------------------------------*/
/**
 * Main function where the program execution begins. Toggle two LEDs
 * when a push button is pressed. Functions from user-defined GPIO
 * library is used instead of low-level logic operations.
 */
int main(void)
```

```
{
    /* GREEN LED */
    GPIO_config_output(&DDRB, LED_GREEN);
    GPIO_write_low(&PORTB, LED_GREEN);

    /* RED LED */
       GPIO_config_output(&DDRC, LED_RED);
       GPIO_write_low(&PORTC, LED_RED);

    /* push button */
    GPIO_config_input_pullup(&DDRD, BTN);

    // Infinite loop
    while (1)
    {
        // Pause several milliseconds
        _delay_ms(BLINK_DELAY);
        if (!GPIO_read(&PIND, BTN))
       {
           GPIO_toggle(&PORTB, LED_GREEN);
           GPIO_toggle(&PORTC, LED_RED);
       }

    }

    // Will never reach this
    return 0;
}
```

3. **<u>In your words, describe the difference between the declaration and the definition of the function in C. Give an example</u>**

Declaration of a function provides the compiler with the basic attributes of the function i.e. the name of the function, the number and type of arguments it takes and its return type. For example:

```
int func(int, int);
```

Whereas the definition of the function is used for allocating memory for the function and provide extra details of what a particular function does. E.g.

```
int add(int x, int y)
{
    return (x + y);
}
```

Differences between function declaration and definition

Declaration	Definition
A function can be declared any number of times	A function can be defined only once
Memory will not be allocated during declaration	Memory will be allocated

`int func(int);`	`int func(int x)`
	`{`
	`  return x;`
	`}`

4. **Reprogramed Knight raider.**

```
/*
 * gpio.c :m
 *
 * Created: 10/7/2020 11:13:50
 * Author : masau
 */

/*****************************************************************************
 *
 * Alternately toggle two LEDs when a push button is pressed. Use
 * functions from GPIO library.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2019-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 *
 *****************************************************************************/

/* Defines -----------------------------------------------------------*/
//Defines
#define LED_GREEN_5   PB5   // AVR pin where green LED is connected
#define LED_RED_4     PB4   // AVR pin where RED LED is connected
#define LED_GREEN_3   PB3   // AVR pin where green LED is connected
#define LED_RED_2     PB2   // AVR pin where RED LED is connected
#define LED_GREEN_1   PB1   // AVR pin where green LED is connected
#define BTN           PD0   // AVR pin where PUSH BUTTONis connected
#define BLINK_DELAY   50    // TIME DELAY IN MILISECONDS
#ifndef F_CPU
#define F_CPU 16000000      // CPU frequency in Hz required for delay
#endif


/* Includes ----------------------------------------------------------*/
#include <util/delay.h>     // Functions for busy-wait delay loops
```

```c
#include <avr/io.h>         // AVR device-specific IO definitions
#include "gpio.h"           // GPIO library for AVR-GCC

/* Function definitions ----------------------------------------------*/
/**
 * Main function where the program execution begins. Toggle FIVE LEDs
 * when a push button is pressed.
 */


//This is the modified version of knight raider
int main(void)
{
    /* GREEN LED */
    GPIO_config_output(&DDRB, LED_GREEN_1);  //output configuration
    GPIO_write_low(&PORTB, LED_GREEN_1);     //port assignment

       GPIO_config_output(&DDRB, LED_GREEN_3);
       GPIO_write_low(&PORTB, LED_GREEN_3);

       GPIO_config_output(&DDRB, LED_GREEN_5);
       GPIO_write_low(&PORTB, LED_GREEN_5);



    /* RED LED */
       GPIO_config_output(&DDRB, LED_RED_2);
       GPIO_write_low(&PORTB, LED_RED_2);

       GPIO_config_output(&DDRB, LED_RED_4);
       GPIO_write_low(&PORTB, LED_RED_4);


    /* push button */
    GPIO_config_input_pullup(&DDRD, BTN);


 // Infinite loop
 while (1)
 {
        // Pause several milliseconds
        _delay_ms(BLINK_DELAY);

        if (!GPIO_read(&PIND, BTN))
        {

               //forward toggle
               GPIO_toggle(&PORTB, LED_GREEN_1);
               _delay_ms(BLINK_DELAY);
               GPIO_toggle(&PORTB, LED_GREEN_1);
               GPIO_toggle(&PORTB, LED_RED_2);
               _delay_ms(BLINK_DELAY);
               GPIO_toggle(&PORTB, LED_RED_2);
               GPIO_toggle(&PORTB, LED_GREEN_3);
               _delay_ms(BLINK_DELAY);
               GPIO_toggle(&PORTB, LED_GREEN_3);
               GPIO_toggle(&PORTB, LED_RED_4);
               _delay_ms(BLINK_DELAY);
```

```
                GPIO_toggle(&PORTB, LED_RED_4);
                GPIO_toggle(&PORTB, LED_GREEN_5);
                _delay_ms(BLINK_DELAY);

                //backward toggle
                GPIO_toggle(&PORTB, LED_GREEN_5);
                GPIO_toggle(&PORTB, LED_RED_4);
                _delay_ms(BLINK_DELAY);
                GPIO_toggle(&PORTB, LED_RED_4);
                GPIO_toggle(&PORTB, LED_GREEN_3);
                _delay_ms(BLINK_DELAY);
                GPIO_toggle(&PORTB, LED_GREEN_3);
                GPIO_toggle(&PORTB, LED_RED_2);
                _delay_ms(BLINK_DELAY);
                GPIO_toggle(&PORTB, LED_RED_2);
                GPIO_toggle(&PORTB, LED_GREEN_1);


        }
}
// Will never reach this
return 0;
}
```