Data type	**Number of bits**	**Range**	**Description**
`uint8_t`	8	0, 1, ..., 255	Unsigned 8-bit integer
`int8_t`	8	-128, ..., 127	Signed 8-bit integer
`uint16_t`	16	0, 1, ..., 65535	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`	0	0	zero type

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

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

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+2*x*y+y*y;

    return result;
}
```

```c
#ifndef GPIO_H
#define GPIO_H

/*****************************************************************************
 *
 * 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.
 *
 *****************************************************************************/

/**
 * @file  gpio.h
 * @brief GPIO library for AVR-GCC.
 *
 * @details
 * The library contains functions for controlling AVRs' gpio pin(s).
 *
 * @note
 * Based on AVR Libc Reference Manual. Tested on 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 <avr/io.h>

/* Function prototypes -----------------------------------------------*/
/**
```

```
 * @brief Configure one output pin in Data Direction Register.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
 * @param pin_num - Pin designation in the interval 0 to 7
 */
void GPIO_config_output(volatile uint8_t *reg_name, uint8_t pin_num);

/**
 * @brief Configure one input pin in DDR without pull-up resistor.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
 * @param pin_num - Pin designation in the interval 0 to 7
 */
void GPIO_config_input_nopull(volatile uint8_t *reg_name, uint8_t pin_num);

/**
 * @brief Configure one input pin in DDR and enable pull-up resistor.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
 * @param pin_num - Pin designation in the interval 0 to 7
 */
void GPIO_config_input_pullup(volatile uint8_t *reg_name, uint8_t pin_num);

/**
 * @brief Set one output pin in PORT register to low.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
 * @param pin_num - Pin designation in the interval 0 to 7
 */
void GPIO_write_low(volatile uint8_t *reg_name, uint8_t pin_num);

/**
 * @brief Set one output pin in PORT register to high.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
```

```
 * @param pin_num - Pin designation in the interval 0 to 7
 */
void GPIO_write_high(volatile uint8_t *reg_name, uint8_t pin_num);

/**
 * @brief Toggle one output pin value in PORT register.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
 * @param pin_num - Pin designation in the interval 0 to 7
 */
void GPIO_toggle(volatile uint8_t *reg_name, uint8_t pin_num);

/**
 * @brief Get input pin value from PIN register.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
 * @param pin_num - Pin designation in the interval 0 to 7
 * @return value of GPIO pin - 0 or 1
 */
uint8_t GPIO_read(volatile uint8_t *reg_name, uint8_t pin_num);

/**
 * @brief Shifts forward bits of GPIO register by one.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
 */
void GPIO_shift(volatile uint8_t *reg_name);

/**
 * @brief Shifts back bits of GPIO register by one.
 * @param reg_name - Address of Data Direction Register, such as &DDRA,
 *                   &DDRB, ...
 */
void GPIO_unshift(volatile uint8_t *reg_name);
```

```
#endif
```

```c
/***********************************************************************
 *
 * 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);
}

/*-------------------------------------------------------------------*/
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
}
```

```c
/*-------------------------------------------------------------------*/
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;
    }
}

/*-------------------------------------------------------------------*/
void GPIO_shift(volatile uint8_t *reg_name)
{
```

```
    *reg_name=*reg_name<<1;
}

/*-------------------------------------------------------------------------*/
void GPIO_unshift(volatile uint8_t *reg_name)
{
    *reg_name=*reg_name>>1;
}
```

```c
/*****************************************************************************
 *
 * 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     // AVR pin where button is connected
#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_high(&PORTC, LED_RED);

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

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

        if(bit_is_clear(PIND, BTN))
        {
            GPIO_toggle(&PORTB, LED_GREEN);
            GPIO_toggle(&PORTC, LED_RED);
        }
    }

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

Deklarace funkce:

Při deklaraci funkce se uvádí pouze datový typ návratové hodnoty, jméno funkce a typy argumentů. Mohou se uvést i názvy argumentů, ale to není nutné. Než je funkce v programu volána, musí být deklarována, ale definována může být až později.

```
float vypocet(int_8t, float, float *);     /* deklarace funkce */
```

Definice funkce:

V definici funkce jsou navíc názvy argumentů a obsahuje také tělo, ve kterém je napsaný algoritmus, který funkce výkoná. Na konci může být návratová hodnota, kterou funkce po jejím skončení vrátí příkazu, který tuto funkci volal.

```
float vypocet(int_8t a, float b, float *c)
{
    float f;
    *c = (float) a *b;
    f = (float) a + b;
    b = 55.5;


    return f;
}
```

```c
/*****************************************************************************
 *
 * 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_RED_0     PC0     // AVR pin where red LED0 is connected
#define LED_RED_1     PC1     // AVR pin where red LED1 is connected
#define LED_RED_2     PC2     // AVR pin where red LED2 is connected
#define LED_RED_3     PC3     // AVR pin where red LED3 is connected
#define LED_RED_4     PC4     // AVR pin where red LED4 is connected
#define BTN         PD0     // AVR pin where button is connected
#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.
 */
```

```c
int main(void)
{
    int8_t i;
    /* RED LED0 */
    GPIO_config_output(&DDRC, LED_RED_0);
    GPIO_write_high(&PORTC, LED_RED_0);

    /* RED LED1 */
    GPIO_config_output(&DDRC, LED_RED_1);
    GPIO_write_low(&PORTC, LED_RED_1);

    /* RED LED2 */
    GPIO_config_output(&DDRC, LED_RED_2);
    GPIO_write_low(&PORTC, LED_RED_2);

    /* RED LED3 */
    GPIO_config_output(&DDRC, LED_RED_3);
    GPIO_write_low(&PORTC, LED_RED_3);

    /* RED LED4 */
    GPIO_config_output(&DDRC, LED_RED_4);
    GPIO_write_low(&PORTC, LED_RED_4);

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

    // Infinite loop
    while (1)
    {
        for(i=0;i<4;i++)
        {
            // Pause several milliseconds
            _delay_ms(BLINK_DELAY);

            loop_until_bit_is_clear(PIND, BTN);
```

```
            GPIO_shift(&PORTC);
        }

        for(i=0;i<4;i++)
        {
            // Pause several milliseconds
            _delay_ms(BLINK_DELAY);

            loop_until_bit_is_clear(PIND, BTN);

            GPIO_unshift(&PORTC);
       }
    }

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