Table for Data type

Data type	Number of bits	Range	Description
unit8_t	8	0, 1, …, 255	Unsigned 8-bit integer
unit8_t	8	-128, …,127	Signed 8-bit integer
uint16_t	16	0, …, 65535	Unsigned 16-bit integer
int16_t	16	-32768, …, 32767	Signed 16-bit integer
float	32	-3.4E38, …,3.4E38	Single-precision floating-point
void	x	x	x

What is the difference between the declaration and the definition of the function in C? Give an example.

A function definition means the specification of the function name, the return type, the parameters and the complete function body - the actual function. So it is the complete description of the function

Example:

```
             //the definition of function
int sum(a,b) //return_type function name (parameter list)
{
       int c;
       c=a+b;           // Body of function

       return c;

}
```

A function declaration gives information to the compiler about a function name and how to call the function. A compiler reads and translates the source code from top to bottom. If he comes across a word – for example a function name - that he is not yet familiar with at one point in the source text, an error message will be given. It is therefore necessary to make functions known before they are used.

Example:

```
int sum(a,b); //the declaration of function

int main()
{
 int c=0;
 c=sum(5,8);
 printf("Sum=%d",c);
```

```
return 0;
}

int sum(a,b) //return_type function name (parameter list)
{
       int c;
       c=a+b;          // Body of function
return c;
}




                     /*********************************************//**
                     *         Nadir Osman Al-Wattar
                                      main.c
                      ************************************************/
/* Defines -----------------------------------------------------------*/
#define LED_GREEN   PB5     // AVR pin where green LED is connected
#define LED_RED   PC0       // AVR pin where red LED is connected
#define BIN 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
uint8_t perform=0;

/* 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_high(&PORTB, LED_GREEN); // Turn on Led, because active-high Led


    /* second LED */
        GPIO_config_output(&DDRC, LED_RED);
        GPIO_write_high(&PORTC, LED_RED); // Turn off Led, because active-low Led


    /* push button */

       GPIO_config_input_pullup(&DDRD,BIN);



    // Infinite loop

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

```
                            perform=GPIO_read(&PORTD,BIN); // assign the function to
the "perform"

                    if (perform==1)
                    {
                           GPIO_toggle(&PORTB,LED_GREEN);
                           GPIO_toggle(&PORTC,LED_RED);

                    }

      }
    // Will never reach this
    return 0;
}





                    /*******************************************//**
                    *         Nadir Osman Al-Wattar
                               gpio.c
                     ************************************************/

/* 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);
}

/*-------------------------------------------------------------------*/
/* GPIO_config_input_nopull */

/*-------------------------------------------------------------------*/
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(if we increment
the pointer then the pointer point the PORT 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);// Clear bit (and not)
}

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

/*-------------------------------------------------------------------*/
/* GPIO_toggle */

void GPIO_toggle (volatile uint8_t *reg_name, uint8_t pin_num)
```

```
{
 *reg_name = *reg_name ^ (1<<pin_num);   //Toggle bit(XOR)
}
/*--------------------------------------------------------------------*/
/* GPIO_read */

uint8_t GPIO_read(volatile uint8_t *reg_name, uint8_t pin_num)
{


        if (bit_is_clear(PIND,pin_num))
       {

          return 1;// if pressed it returns the value 1
       }
       else
       {
             return 0; // if the button is not pressed it returns the value 0
       }
}
```