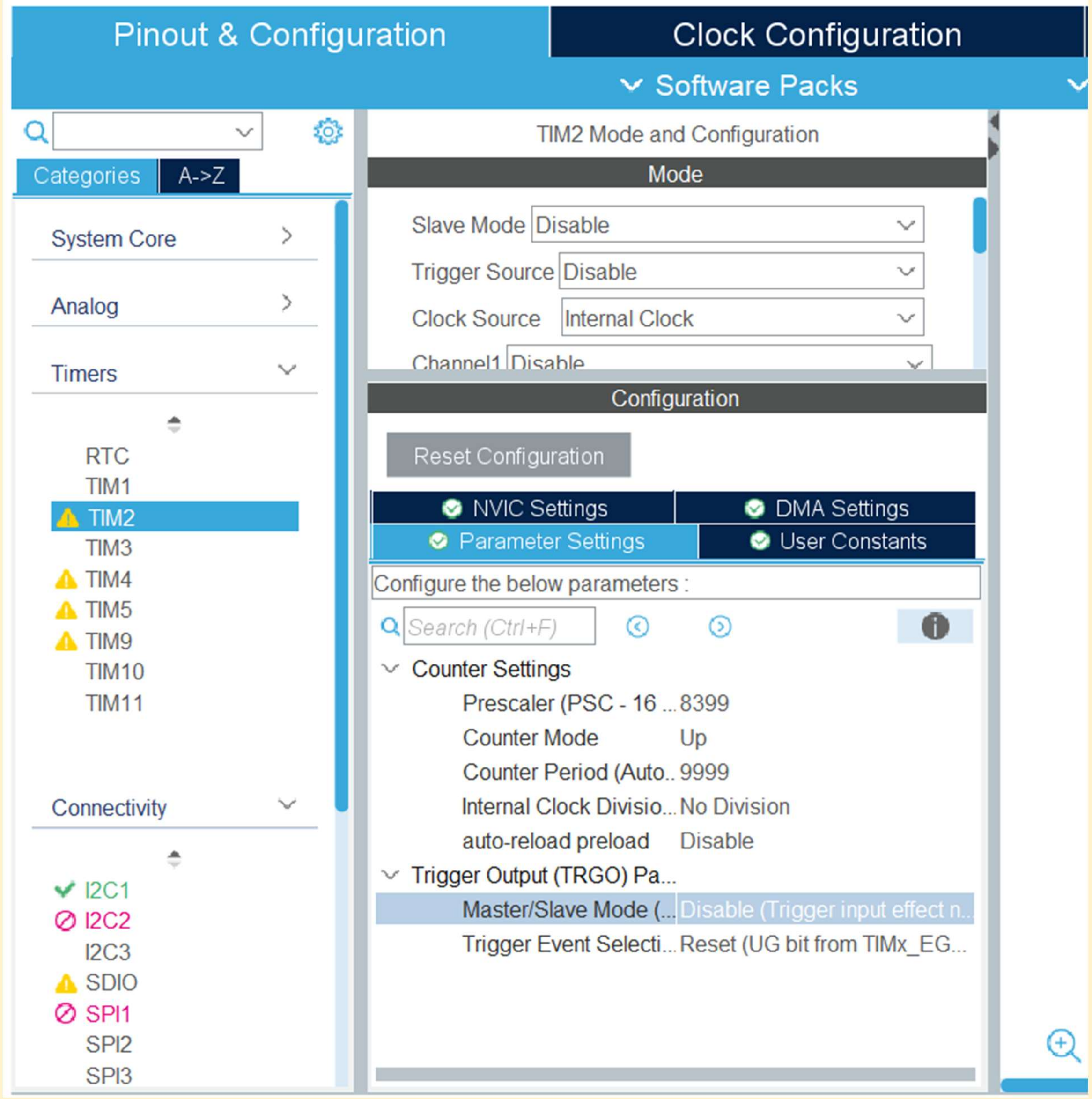
Mark	/11

Team name:	*A14*		
Homework number:	*HOMEWORK 07*		
Due date:	03/11/25		
Contribution	NO	Partial	Full
Mattia Di Mauro			*x*
Francesca Biondi			*x*
Lorenzo Castelli			*x*
Carmen Maria Niro			*x*
Pietro Albrigi			*x*
Notes: none			

Project name	HOMEWORK 7		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Firstly we configured Timer 2 with a prescaler value of 8399 and a period value of 9999 to generate an interrupt with a frequency of 1 Hz.



TIM2 Mode and Configuration

Mode

Slave Mode: Disable
Trigger Source: Disable
Clock Source: Internal Clock
Channel1: Disable

Configuration

Configure the below parameters :

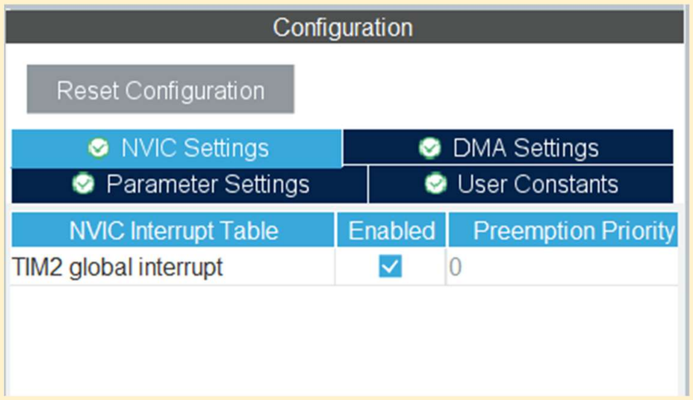
Counter Settings

Prescaler (PSC - 16 ...	8399
Counter Mode	Up
Counter Period (Auto...	9999
Internal Clock Divisio...	No Division
auto-reload preload	Disable

Trigger Output (TRGO) Pa...

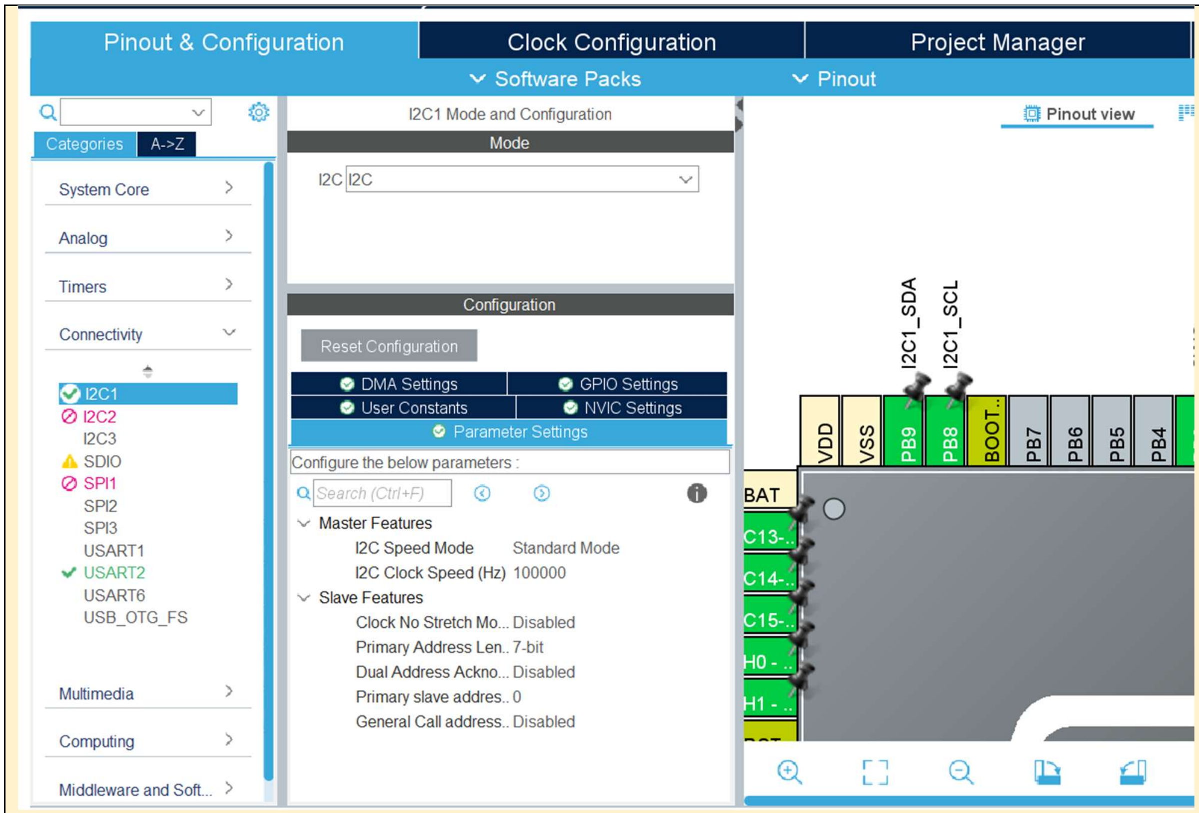
Master/Slave Mode (...	Disable (Trigger input effect n...
Trigger Event Selecti...	Reset (UG bit from TIMx_EG...

The TIM2 global interrupt was then enabled to allow operation in interrupt mode.



NVIC Interrupt Table	Enabled	Preemption Priority
TIM2 global interrupt	✓	0

In the Connectivity section of the .ioc file, we enabled I2C communication, configuring pins PB8 and PB9 as I2C1_SCL and I2C1_SDA, respectively. In addition, USART2 communication was activated to allow data transmission to a remote terminal.



I2C1 Mode and Configuration

Mode

I2C: I2C

Configuration

Reset Configuration

DMA Settings | GPIO Settings | User Constants | NVIC Settings | Parameter Settings

Configure the below parameters :

Master Features

I2C Speed Mode	Standard Mode
I2C Clock Speed (Hz)	100000

Slave Features

Clock No Stretch Mo...	Disabled
Primary Address Len..	7-bit
Dual Address Ackno...	Disabled
Primary slave addres..	0
General Call address..	Disabled

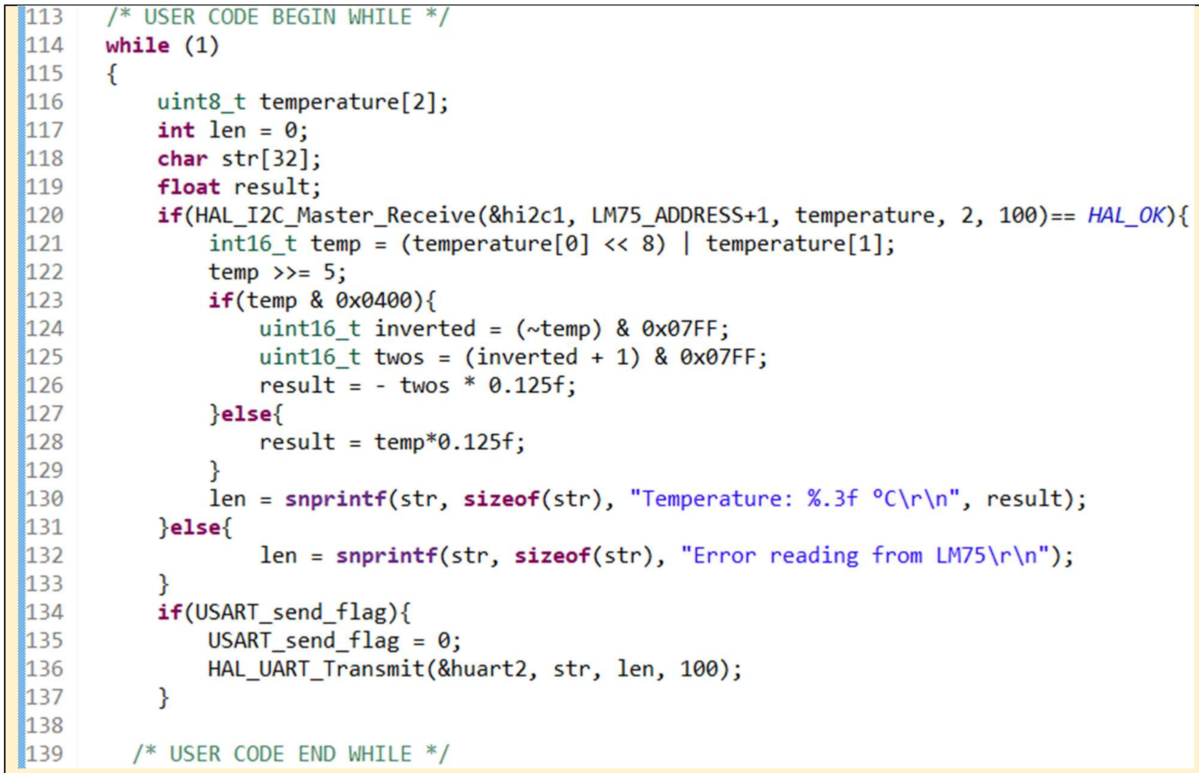
In the main.c file, we declared a flag used to handle the timer callback, as well as the address of the LM75B temperature sensor and its temperature register. The timer was then started in interrupt (IT) mode.

```c
/* USER CODE BEGIN PV */
uint8_t LM75_ADDRESS = 0b10010000;
uint8_t LM75_TEMP_ADDRESS = 0x00;
int USART_send_flag = 0;
/* USER CODE END PV */
```

```c
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim2);
HAL_I2C_Master_Transmit(&hi2c1, LM75_ADDRESS, &LM75_TEMP_ADDRESS, 1, 10);
/* USER CODE END 2 */
```

Every second, the timer interrupt triggers the routine for temperature acquisition.

```c
/* USER CODE BEGIN 0 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim == &htim2){
        USART_send_flag = 1;
    }
}
/* USER CODE END 0 */
```

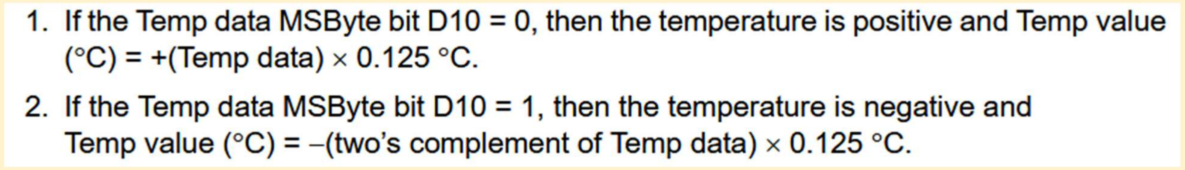


```c
/* USER CODE BEGIN WHILE */
while (1)
{
    uint8_t temperature[2];
    int len = 0;
    char str[32];
    float result;
    if(HAL_I2C_Master_Receive(&hi2c1, LM75_ADDRESS+1, temperature, 2, 100)== HAL_OK){
        int16_t temp = (temperature[0] << 8) | temperature[1];
        temp >>= 5;
        if(temp & 0x0400){
            uint16_t inverted = (~temp) & 0x07FF;
            uint16_t twos = (inverted + 1) & 0x07FF;
            result = - twos * 0.125f;
        }else{
            result = temp*0.125f;
        }
        len = snprintf(str, sizeof(str), "Temperature: %.3f °C\r\n", result);
    }else{
            len = snprintf(str, sizeof(str), "Error reading from LM75\r\n");
    }
    if(USART_send_flag){
        USART_send_flag = 0;
        HAL_UART_Transmit(&huart2, str, len, 100);
    }

  /* USER CODE END WHILE */
```

Within this routine, if the reception through HAL_I2C_Master_Receive() is successful, the two bytes received are combined: the MSByte (most significant byte) is stored in the upper 8 bits of the variable temp, while the LSByte (least significant byte) is stored in the lower 8 bits. The last 5 bits are ignored, as the LM75 temperature register is only 11 bits long (8 bits from the MSByte and 3 bits from the LSByte).

The resulting value is then converted into degrees Celsius. When the MSB is 0, the temperature is positive and the value is multiplied by 0.125. When the MSB is 1, the temperature is negative and the value is first converted to its two's complement, then multiplied by –0.125, as specified in the sensor's datasheet:

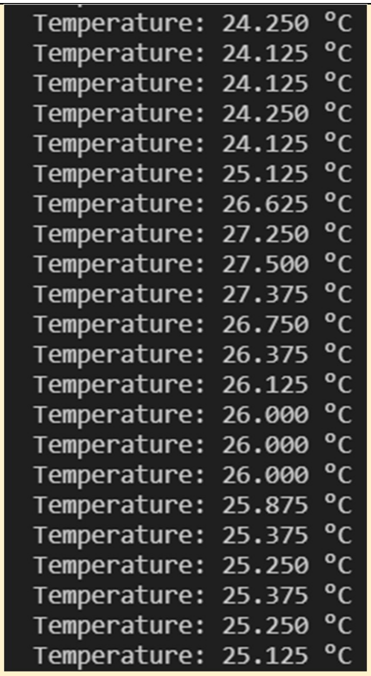


1. If the Temp data MSByte bit D10 = 0, then the temperature is positive and Temp value (°C) = +(Temp data) × 0.125 °C.
2. If the Temp data MSByte bit D10 = 1, then the temperature is negative and Temp value (°C) = −(two's complement of Temp data) × 0.125 °C.

The converted temperature value is then formatted into a string (str) to be sent to the remote terminal. If the I2C reception fails, the string is set to "Error reading from LM75" to indicate a communication error. Finally, the string str is transmitted to the remote terminal using the command HAL_UART_Transmit(). After the transmission, the flag is reset to zero, allowing the timer callback to be executed again at the next interrupt.

" If you only read once the temperature bytes, you .... it passes from 26 °C to 26.875 °C instead of 25.875 °C. "

Although we did not observe this specific issue during our experiments, it is a known behavior of the LM75B sensor related to how temperature data is read through the I^{2}C interface.



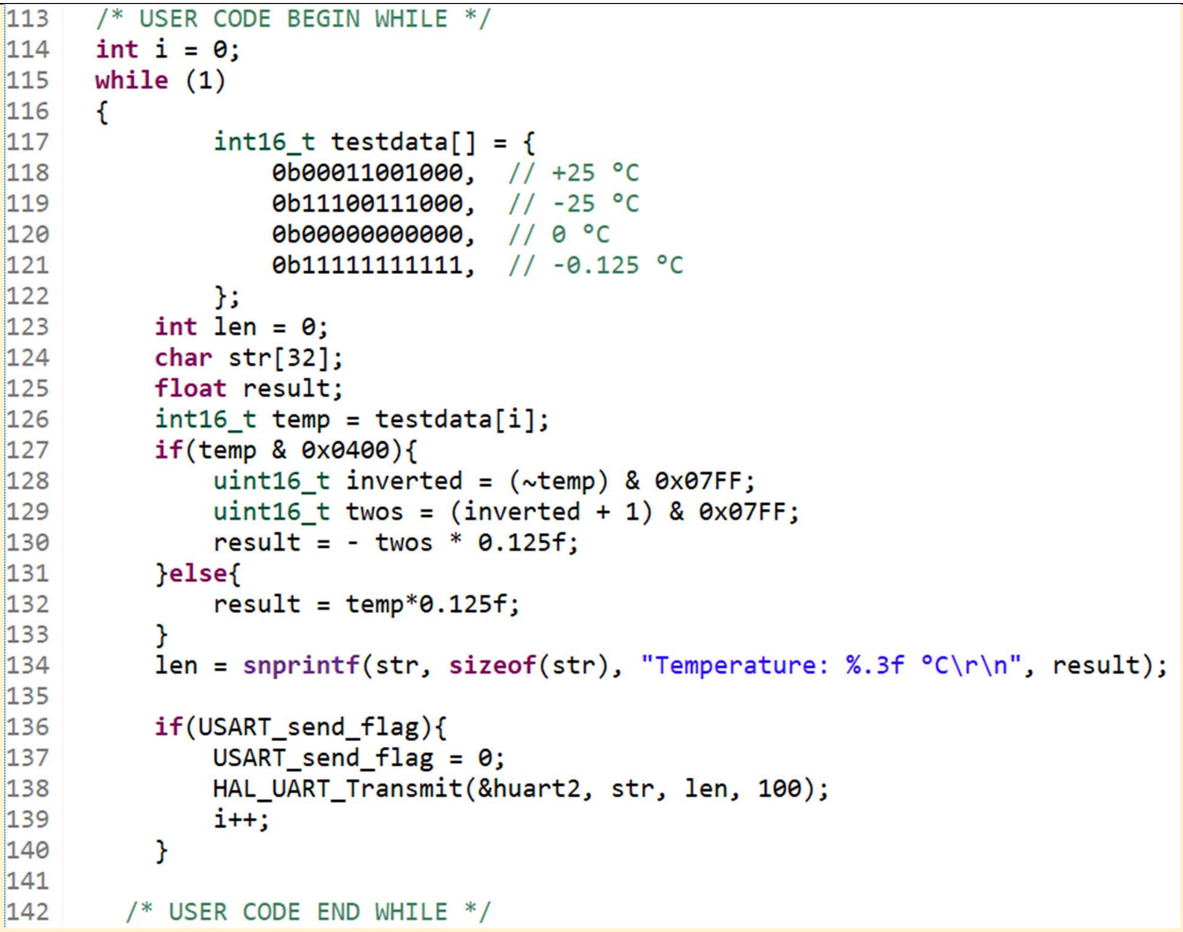
```
Temperature: 24.250 °C
Temperature: 24.125 °C
Temperature: 24.125 °C
Temperature: 24.250 °C
Temperature: 24.125 °C
Temperature: 25.125 °C
Temperature: 26.625 °C
Temperature: 27.250 °C
Temperature: 27.500 °C
Temperature: 27.375 °C
Temperature: 26.750 °C
Temperature: 26.375 °C
Temperature: 26.125 °C
Temperature: 26.000 °C
Temperature: 26.000 °C
Temperature: 26.000 °C
Temperature: 25.875 °C
Temperature: 25.375 °C
Temperature: 25.250 °C
Temperature: 25.375 °C
Temperature: 25.250 °C
Temperature: 25.125 °C
```

The LM75B stores the temperature value in a two-byte register (MSB and LSB). When the temperature changes, both bytes are updated internally, but this update is not atomic.
If the microcontroller reads the two bytes exactly while the LM75B is updating its registers, it may read the MSB from the previous temperature and the LSB from the new one, resulting in an inconsistent value (e.g., a jump from 26 °C to 26.875 °C instead of 25.875 °C). This behavior occurs because the I^{2}C read operation takes several microseconds, and the sensor updates asynchronously.

Possible solutions include:

1. Performing a double read and validating that two consecutive results are identical.
2. Reading at a lower rate than the internal update frequency (e.g., once per second).
3. Using the one-shot mode to synchronize conversion and read operations.

To verify the correctness of the temperature conversion, including for negative values, we introduced a test routine in the main function:

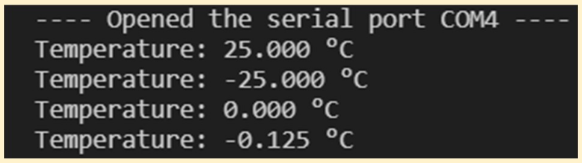


```c
  /* USER CODE BEGIN WHILE */
  int i = 0;
  while (1)
  {
          int16_t testdata[] = {
              0b00011001000,  // +25 °C
              0b11100111000,  // -25 °C
              0b00000000000,  // 0 °C
              0b11111111111,  // -0.125 °C
          };
      int len = 0;
      char str[32];
      float result;
      int16_t temp = testdata[i];
      if(temp & 0x0400){
          uint16_t inverted = (~temp) & 0x07FF;
          uint16_t twos = (inverted + 1) & 0x07FF;
          result = - twos * 0.125f;
      }else{
          result = temp*0.125f;
      }
      len = snprintf(str, sizeof(str), "Temperature: %.3f °C\r\n", result);

      if(USART_send_flag){
          USART_send_flag = 0;
          HAL_UART_Transmit(&huart2, str, len, 100);
          i++;
      }

    /* USER CODE END WHILE */
```

This test confirms that the conversion from raw data to temperature values is correct, both for positive and negative temperatures. The printed results verified that the bit shifting and scaling procedure works as expected.



```
 ---- Opened the serial port COM4 ----
 Temperature: 25.000 °C
 Temperature: -25.000 °C
 Temperature: 0.000 °C
 Temperature: -0.125 °C
```

Professor comments: