PROJECT 1 (PART B)

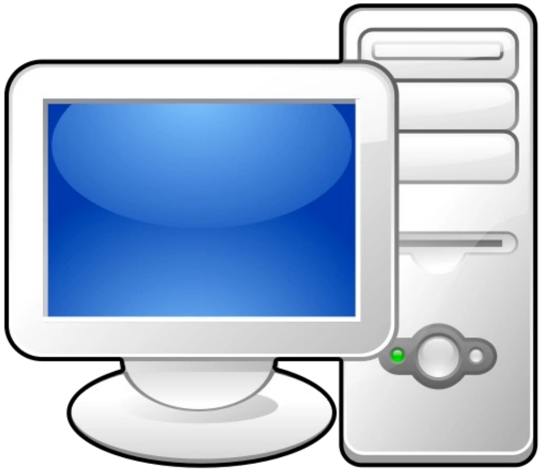
Onboard Spacecraft Software

Ojective

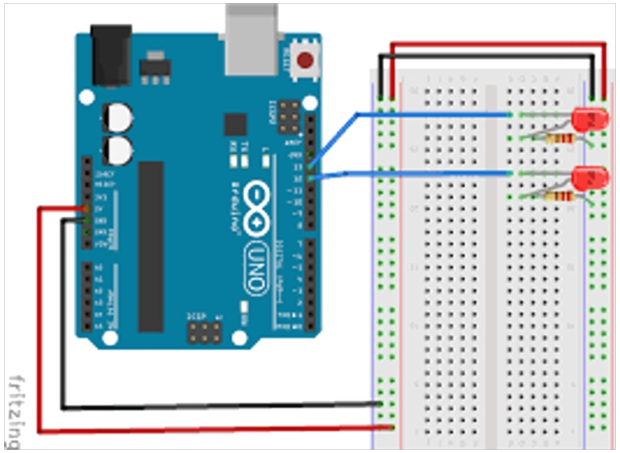
- To build the microcontroller subsystem prototype.
  - Using Arduino
- The subsistem one of a two-part project
  1. A main computing component to controls the whole system.
  2. A microcontroller subsystem that has a direct access to the sensors and actuators.
- Both subsystems communicate using a master/slave message protocol defined for this project.
  - To test the microcontroller subsystem a test unit simulating the main computing subsytem is made using arduino

Development modules

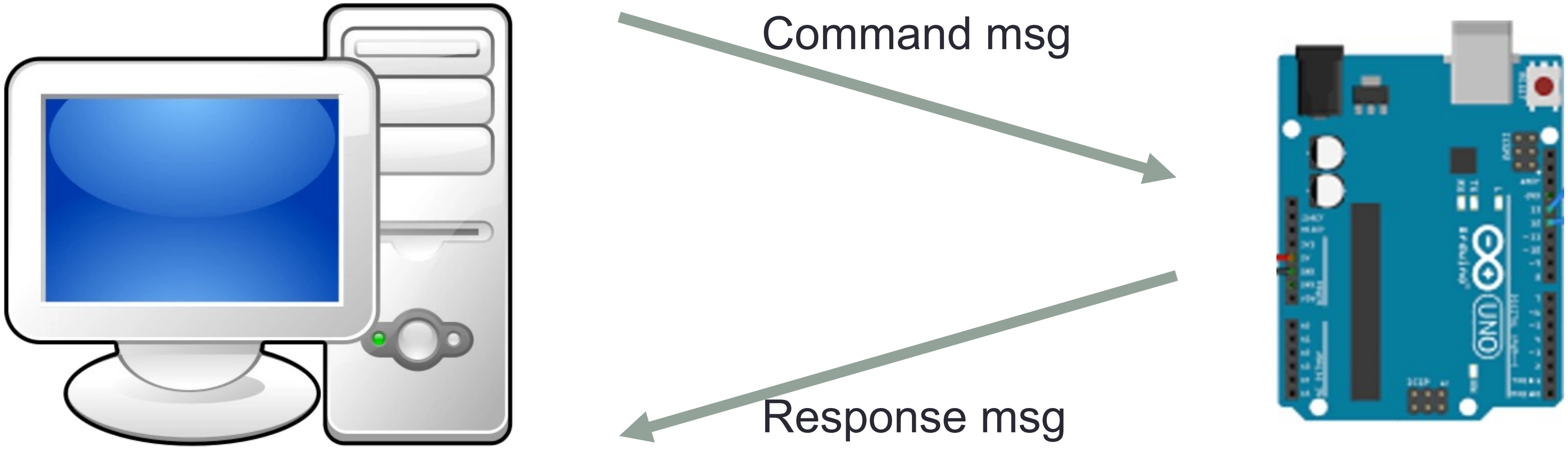
- Software module
  - Desktop PC
  - RTMS O.S.
  - Controls/simulates the logic of the project.



- Hardware module
  - Electornic circuit.
  - Based on Arduino
  - Controls/simulates the hardware of the project.



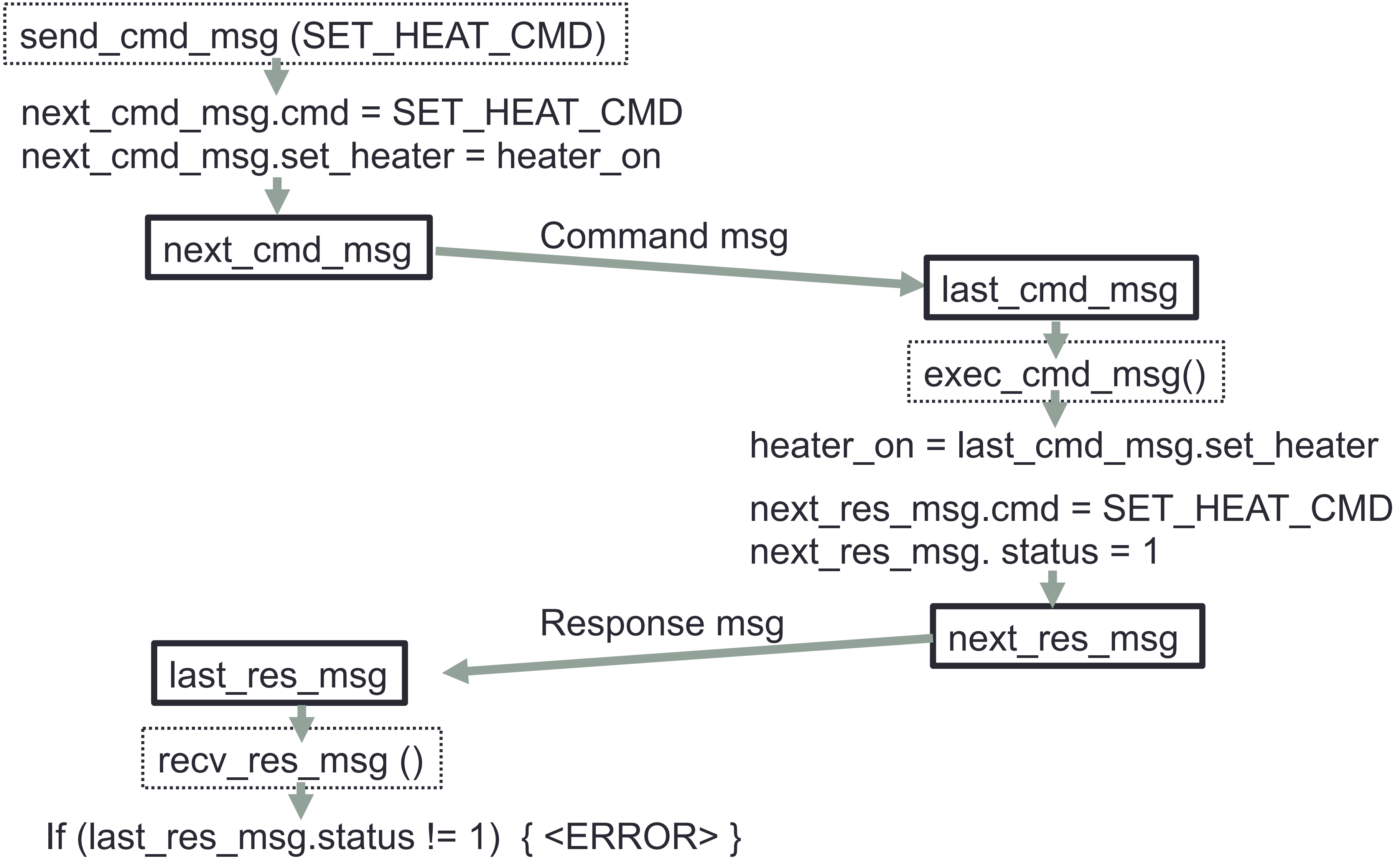
Comunication protocol: Master / Slave



Command msg

Response msg

Protocol example: Actuator



send_cmd_msg (SET_HEAT_CMD)

next_cmd_msg.cmd = SET_HEAT_CMD
next_cmd_msg.set_heater = heater_on

next_cmd_msg

Command msg

last_cmd_msg

exec_cmd_msg()

heater_on = last_cmd_msg.set_heater

next_res_msg.cmd = SET_HEAT_CMD
next_res_msg. status = 1

next_res_msg

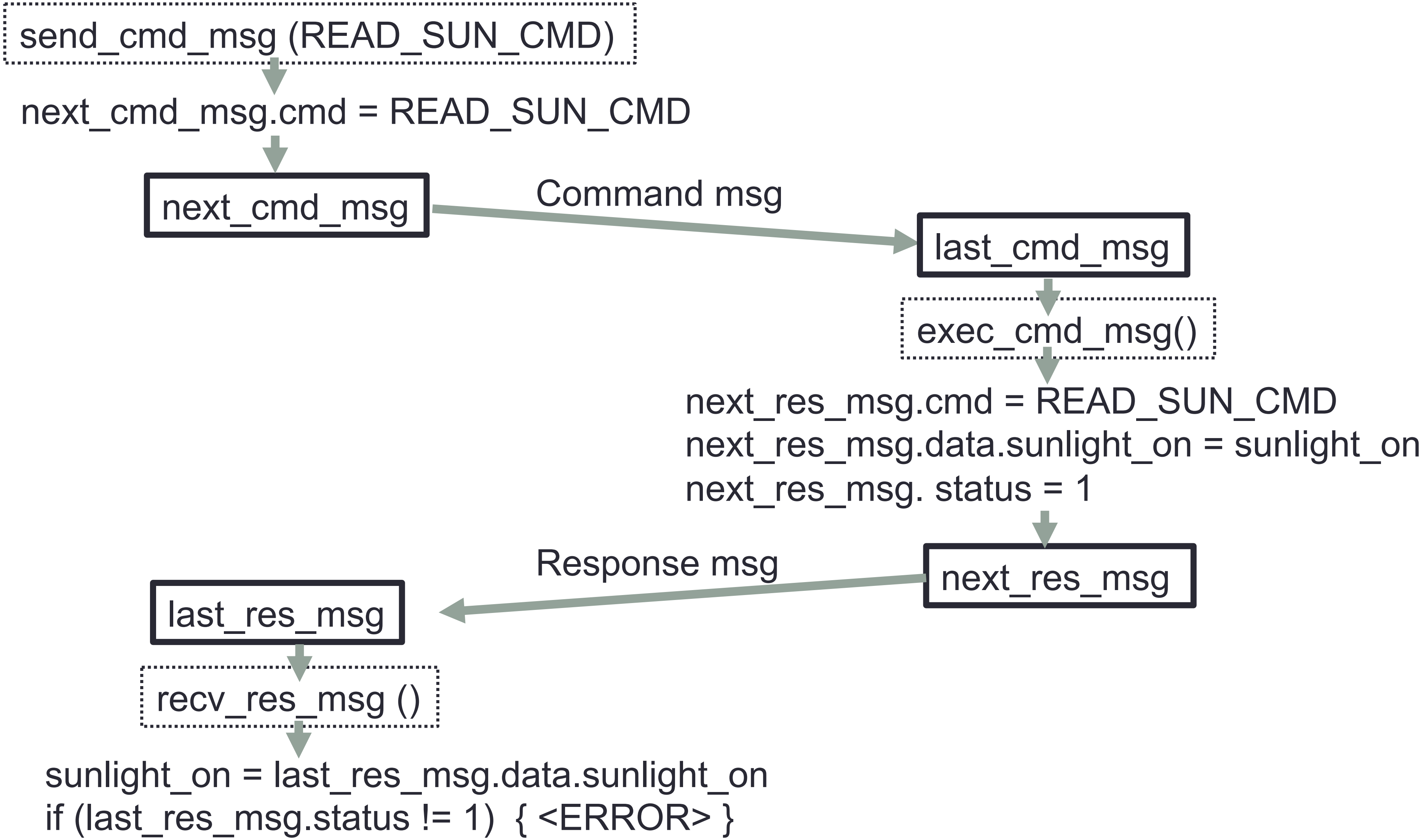
Response msg

last_res_msg

recv_res_msg ()

If (last_res_msg.status != 1) { <ERROR> }

Protocol example: Sensor



send_cmd_msg (READ_SUN_CMD)

next_cmd_msg.cmd = READ_SUN_CMD

next_cmd_msg

Command msg

last_cmd_msg

exec_cmd_msg()

next_res_msg.cmd = READ_SUN_CMD
next_res_msg.data.sunlight_on = sunlight_on
next_res_msg. status = 1

next_res_msg

Response msg

last_res_msg

recv_res_msg ()

sunlight_on = last_res_msg.data.sunlight_on
if (last_res_msg.status != 1) { <ERROR> }

Protocol messages: Commands and responses

Message name	Command data	Response data
NO_CMD	-----	-----
SET_HEAT_CMD	set_heater	status
READ_SUN_CMD	-----	data.sunlight_on status
READ_TEMP_CMD	-----	data.temperature status
READ_POS_CMD	-----	data.position status

Functions interface: Arduino

- Function: get_temperatura()
- Inputs variables taken from the global state:
  1. **heater_on**
  2. **sunlight_on**
  3. **temperature**
  4. **time_temperatura**
- Outputs variables modified on the global state:
  1. **temperature**
  2. **time_temperatura**
- Constants values used (from #define):
  1. **SHIP_SPECIFIC_HEAT**
  2. **SHIP_MASS**
  3. **HEATER_POWER**
  4. **SUNLIGHT_POWER**
  5. **HEAT_POWER_LOSS**

Functions interface: Arduino

- Function: get_position ()
- Inputs variables taken from the global state:
  1. **init_time_orbit**
- Outputs variables modified on the global state:
  1. **position**
- Constants values used (from #define and static global variables):
  1. **ORBIT_POINTS_SIZE**
  2. **ORBIT_TIME**
  3. **orbit_points**

Functions interface: Arduino

- Function: exec_cmd_msg ()
- Inputs variables taken from the global state:
  1. **last_cmd_msg**
  2. **sunlight_on**
  3. **temperature**
  4. **position**
- Outputs variables modified on the global state:
  1. **next_res_msg**
  2. **heater_on**
- Constants values used (from *enum command*):
  1. **NO_CMD**
  2. **SET_HEAT_CMD**
  3. **READ_SUN_CMD**
  4. **READ_TEMP_CMD**
  5. **READ_POS_CMD**

How to communicate both subsytems

- Both subsystems communicate using a master/slave message protocol defined for this project.

  1. The underlying hardware is a UART serial communication hardware.

- Using the Arduino Framework to control the UART serial communication

Arduino Serial functions

- Init the board to send/rceive data using the serial port
  - `Serial.begin(speed)`
    - **speed** is the trnasference speed in bouds (usually 9600)

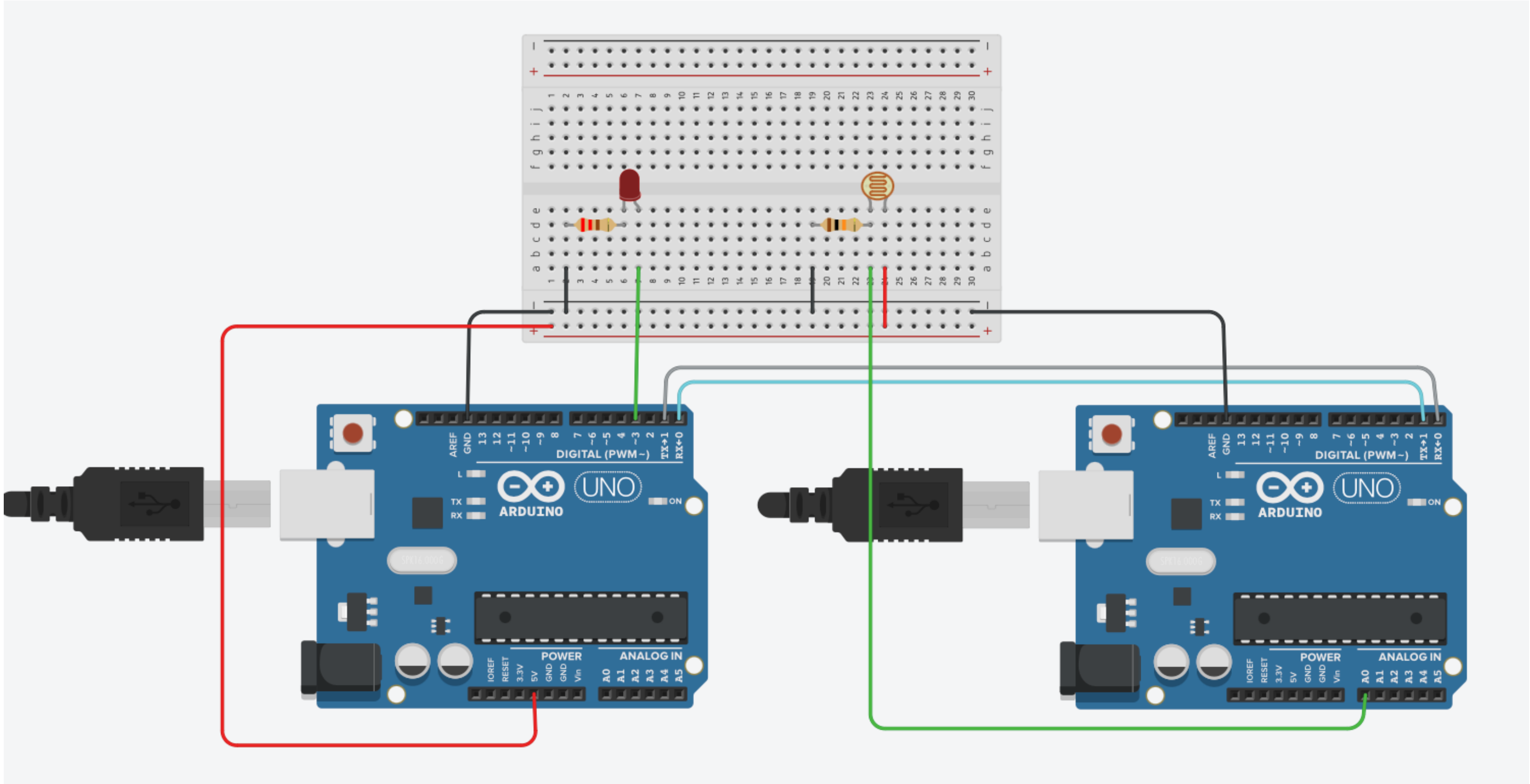
- Send dat using the serial port (Arduino → PC)
  - `Serial.print(data)`
  - `Serial.println(data)`
    - **data** is a variable (string, int, etc).
  - `Serial.write(data, size)`
    - **data** is an array of bytes (unsigned char)
    - **data** is the size of the array

Arduino Serial functions

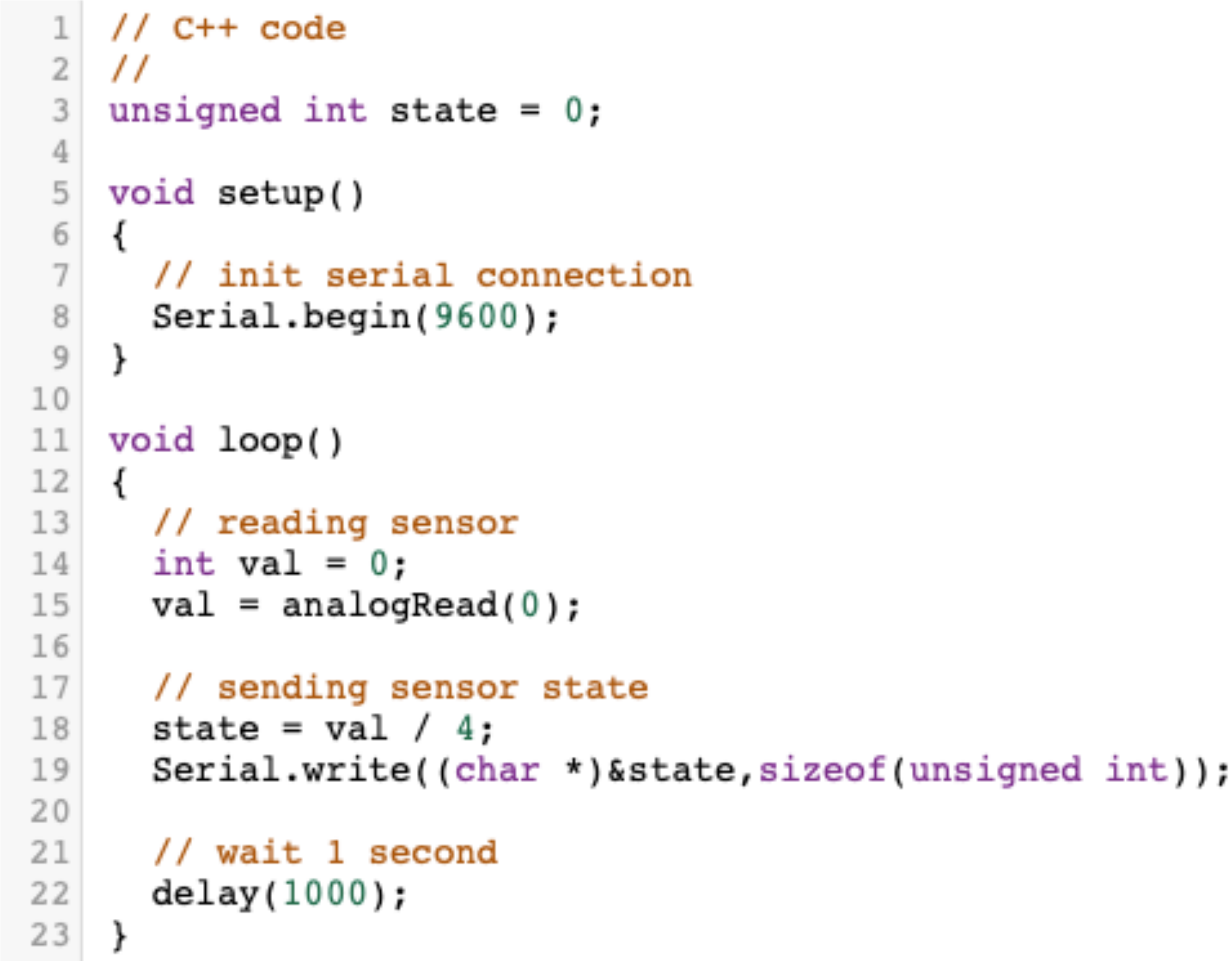
- Read data from the serial port(Arduino ← PC)
  - `val Serial.available()`
    - Send the number of bytes already read and in the inner buffer
  - `val = Serial.read()`
    - Read one byte form the buffer.
  - `val = Serial.readBytes(data, size)`
    - Read an array of bytes in data of size ”size”.

- Clean the inner buffer
  - `Serial.flush()`

Example:

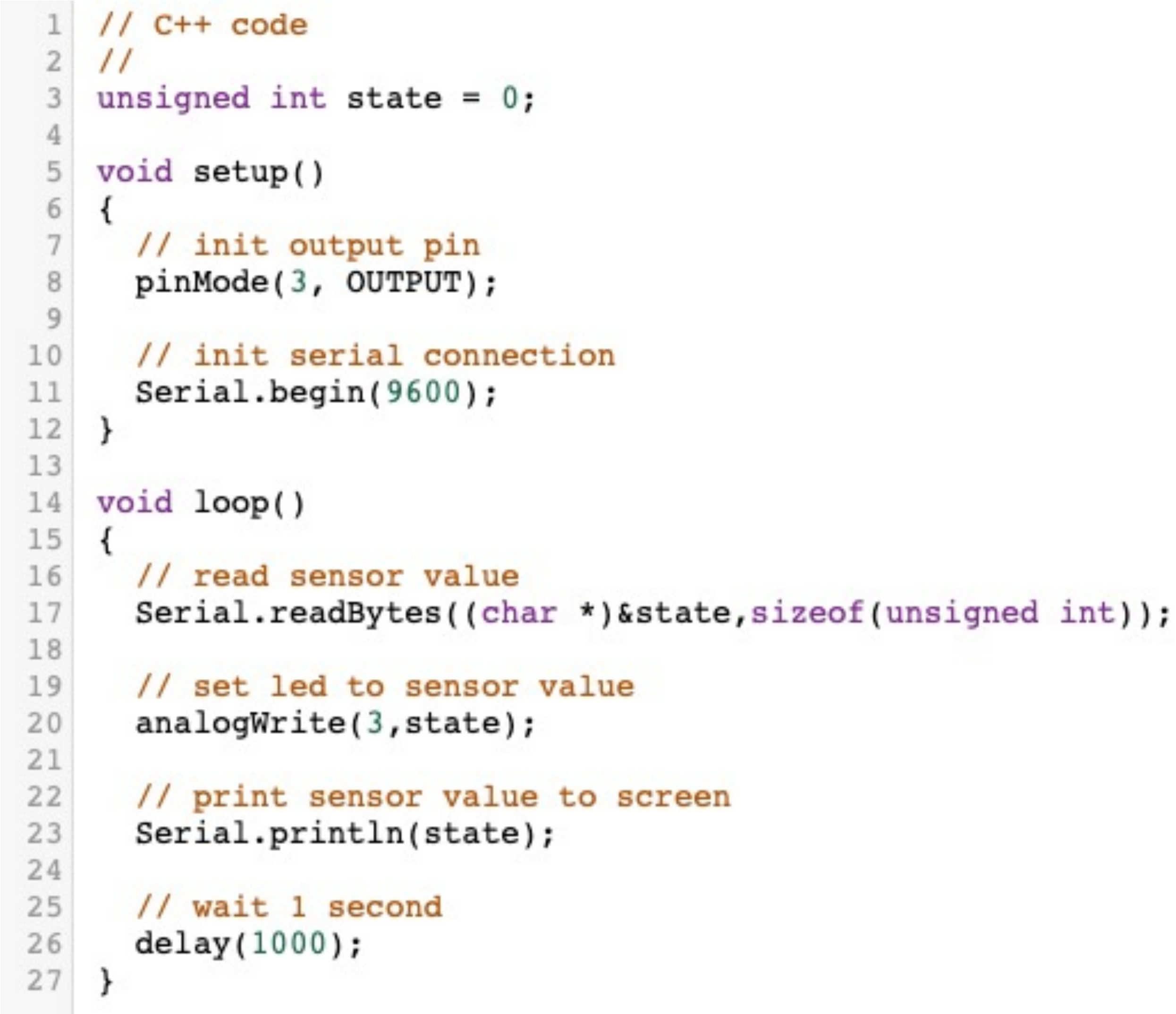

AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX▸1
RX◂0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
ANALOG IN
A0
A1
A2
A3
A4
A5
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX▸1
RX◂0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
ANALOG IN
A0
A1
A2
A3
A4
A5

Sensor Controller Code:



```cpp
// C++ code
//
unsigned int state = 0;

void setup()
{
  // init serial connection
  Serial.begin(9600);
}

void loop()
{
  // reading sensor
  int val = 0;
  val = analogRead(0);

  // sending sensor state
  state = val / 4;
  Serial.write((char *)&state,sizeof(unsigned int));

  // wait 1 second
  delay(1000);
}
```

Led Controller Code:



```
// C++ code
//
unsigned int state = 0;

void setup()
{
  // init output pin
  pinMode(3, OUTPUT);

  // init serial connection
  Serial.begin(9600);
}

void loop()
{
  // read sensor value
  Serial.readBytes((char *)&state,sizeof(unsigned int));

  // set led to sensor value
  analogWrite(3,state);

  // print sensor value to screen
  Serial.println(state);

  // wait 1 second
  delay(1000);
}
```