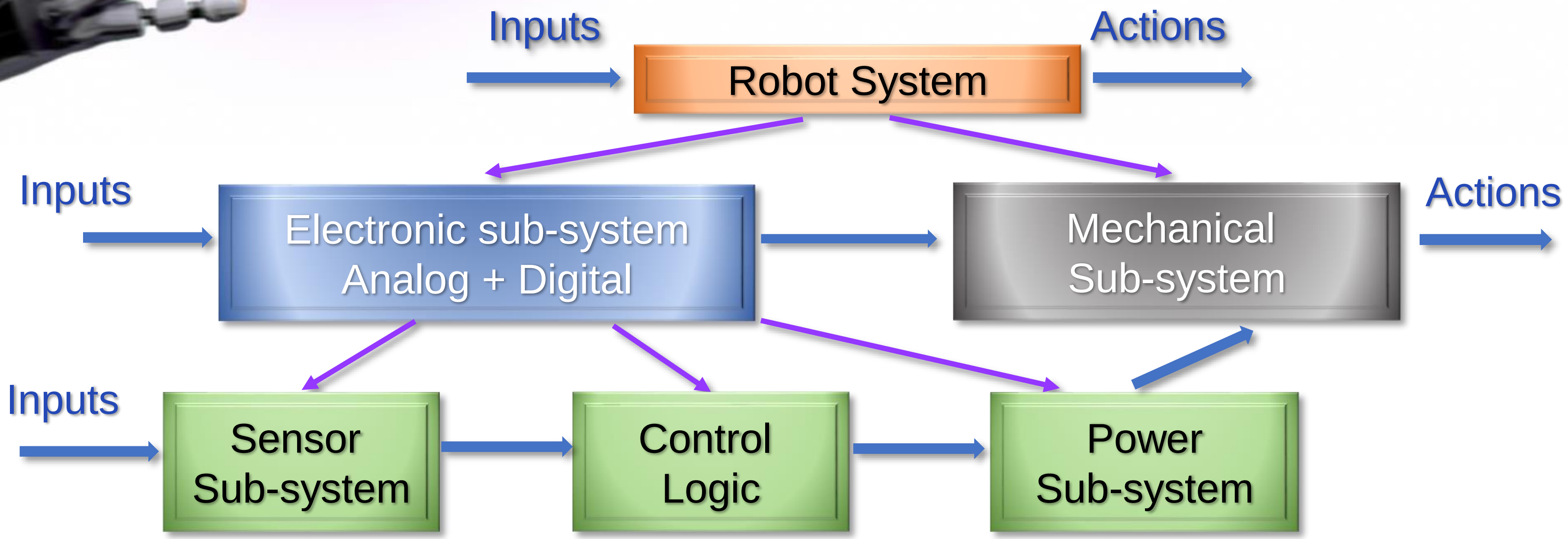
ELEC1100: Introduction to Electro-Robot Design

Lecture 15: Arduino Code (Part II) – Structure & Conditional Statement

SONG Shenghui and MURCH Ross, Dept. of ECE, HKUST

ELEC1100 ROADMAP



Sensor Basics:

Wk6: Sensor Basic –
Sensor/Line/ADC

Combinational/Sequential Logic:

Wk7: Robot Brain: Logic Gate and
Logic Operation

Wk8: MCU & Arduino

Wk9: Programming Language

Basic electronics:

Wk1: Basic Electronics -
Charge/Current/Voltage/Resistor

Wk2: Energy/Power and DC Sources

Motor Power Supply:

Wk3: Pulse Signal and PWM Control

Wk4: Transistor and H-Bridge

LAST LECTURE

- Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure.

  FUNCTIONS

  - For controlling the Arduino board and performing computations.

  VARIABLES

  - Arduino data types and constants.

  STRUCTURE

  - The elements of Arduino (C++) code.

- We will only introduce some key components for your project.

STRUCTURE

- Sketch
  - setup()
  - loop()

- Control Structure
  - for
  - if
  - while
  - break

- Further Syntax
  - #define (define)
  - /* */ (block comment)
  - / / (single line comment)
  - ; (semicolon)
  - {} (curly braces)

- Arithmetic Operators
  - % (remainder)
  - * (multiplication)
  - + (addition)
  - - (subtraction)
  - / (division)
  - = (assignment operator)

- Boolean Operators
  - ! (logical not)
  - && (logical and)
  - || (logical or)

- Comparison Operators
  - != (not equal to)
  - < (less than)
  - <= (less than or equal to)
  - == (equal to)
  - > (greater than)
  - >= (greater than or equal to)

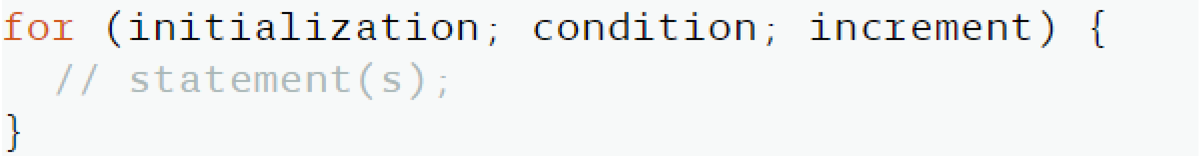
STRUCTURE: SKETCH

- setup()
  - This function is called when a sketch starts.
  - Use it to initialize variables, pin modes, start using libraries, etc.
  - It will only run once, after each power up or reset of the Arduino board.

- loop ()
  - After creating a setup () function, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond.
  - Use it to actively control the Arduino board.

STRUCTURE: CONTROL STRUCTURE [1]

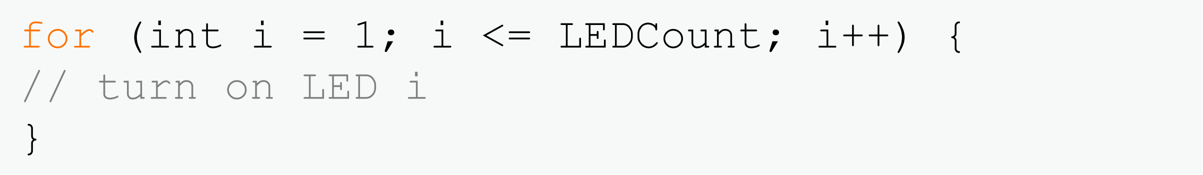
- The for loops
  - This statement is used to repeat a block of statements enclosed in curly braces.



```
for (initialization; condition; increment) {
  // statement(s);
}
```

"initialization" – happens first and exactly once.
"condition" – Each time through the loop, the "condition" is tested; if it's true, the statement block, and the "increment" is executed, then the "condition" is tested again. When the "condition" becomes false, the loop ends.
"increment" – executed each time through the loop when "condition" is true.

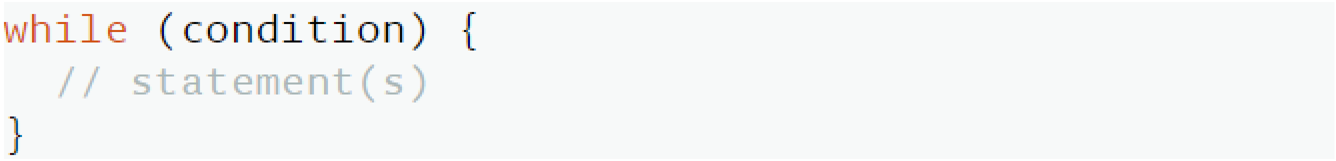
  - Example code: you have more than one LED to turn on response to the temperature changing. `LEDCount` is the number of LEDs.



```
for (int i = 1; i <= LEDCount; i++) {
// turn on LED i
}
```

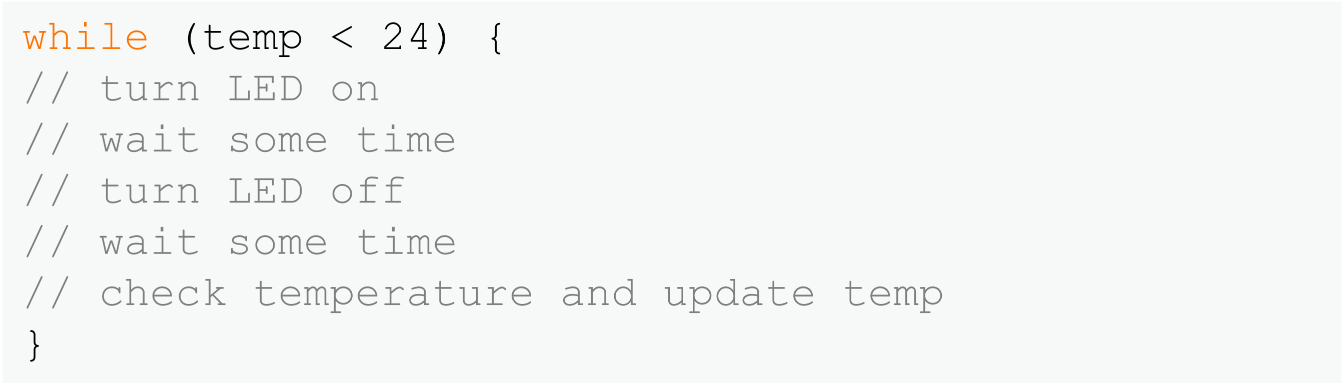
STRUCTURE: CONTROL STRUCTURE [2]

- The while loops
  - A while loop will loop continuously and infinitely until the expression inside the parenthesis () becomes false.



```
while (condition) {
  // statement(s)
}
```

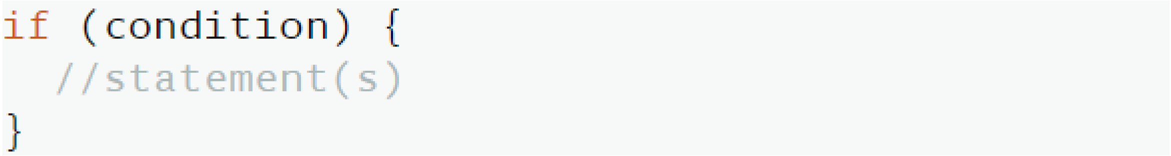
  - Example code: you want an LED to blink repeatedly so long as the temperature is less than 24 degree.



```
while (temp < 24) {
// turn LED on
// wait some time
// turn LED off
// wait some time
// check temperature and update temp
}
```

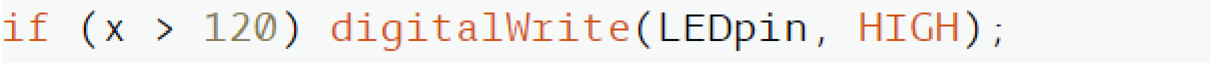
STRUCTURE: CONTROL STRUCTURE [3]

- The if statement
  - This statement checks for a condition and executes the proceeding statement or set of statements if the condition is true.



```
if (condition) {
  //statement(s)
}
```

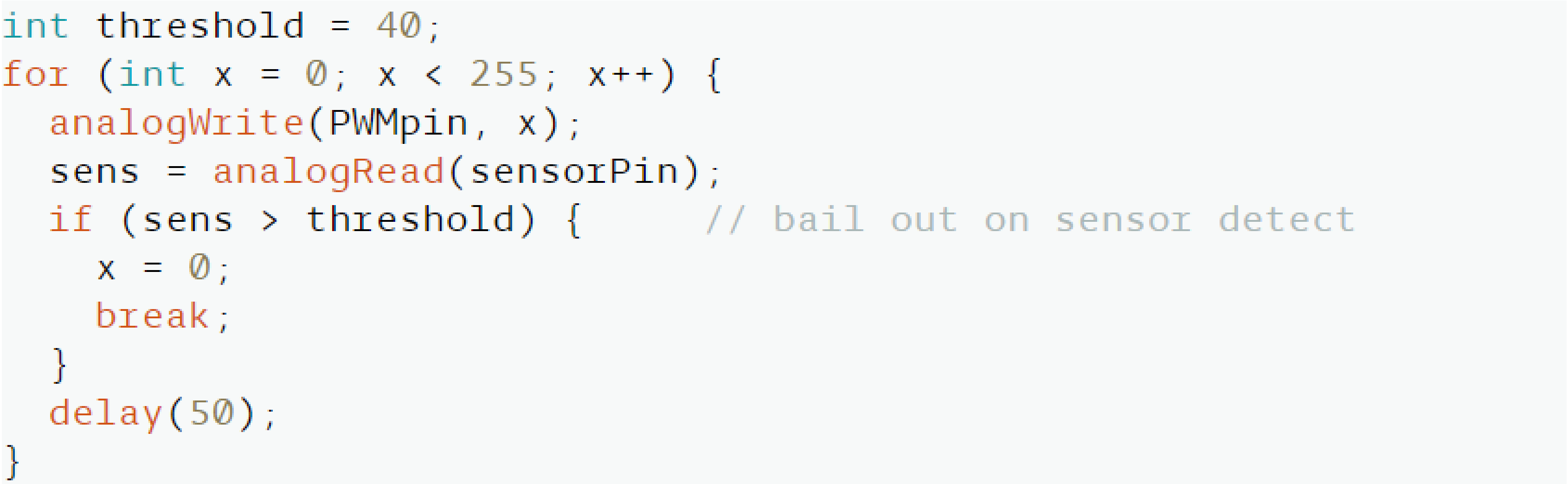
  - Example Code:



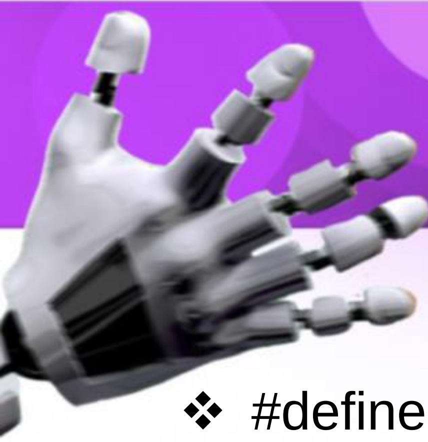
```
if (x > 120) digitalWrite(LEDpin, HIGH);
```

STRUCTURE: CONTROL STRUCTURE [4]

- The break statement
  - This statement is used to exit from a for loop, bypassing the normal loop condition.
  - Example Code: you want the control exits the for loop when the sensor value exceeds the threshold.



```
int threshold = 40;
for (int x = 0; x < 255; x++) {
  analogWrite(PWMpin, x);
  sens = analogRead(sensorPin);
  if (sens > threshold) {      // bail out on sensor detect
    x = 0;
    break;
  }
  delay(50);
}
```

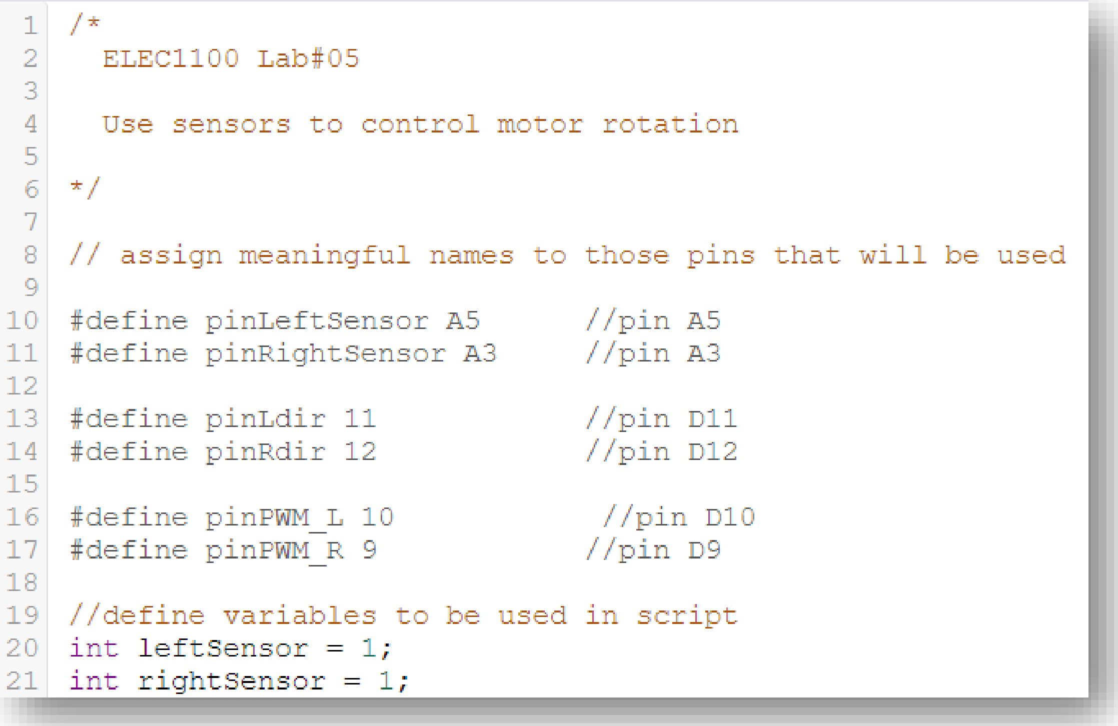


STRUCTURE: FURTHER SYNTAX [1]

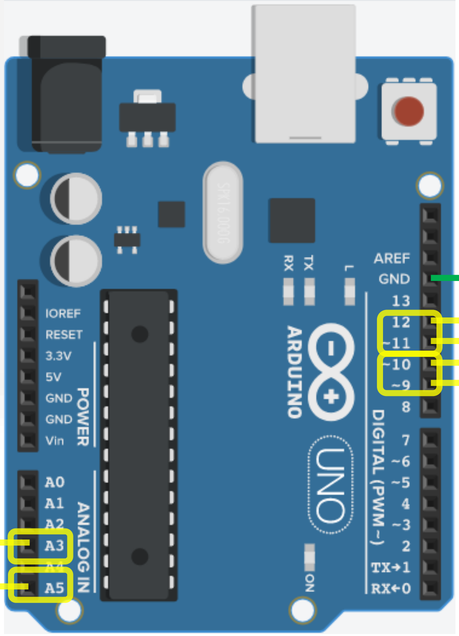
- #define
  - a useful C++ component that allows the programmer to give a name to a constant value before the program is compiled.
- /* */ and / /
  - Comments are lines in the program that are used to inform yourself or others about the way the program works. They are ignored by the compiler.
- ;
  - Used to end a statement.
- “{}”
  - A major part of the C++ programming language. Commonly used in functions, loops and conditional statements.

STRUCTURE: FURTHER SYNTAX [2]

- Example Code: your Lab#05



```
/*
  ELEC1100 Lab#05

  Use sensors to control motor rotation

*/

// assign meaningful names to those pins that will be used

#define pinLeftSensor A5        //pin A5
#define pinRightSensor A3       //pin A3

#define pinLdir 11              //pin D11
#define pinRdir 12              //pin D12

#define pinPWM_L 10              //pin D10
#define pinPWM_R 9              //pin D9

//define variables to be used in script
int leftSensor = 1;
int rightSensor = 1;
```



GND = 0V (connect to your breadboard GND)

D12 – R_DIR
D11 – L_DIR
D10 – PWM_L
D9 – PWM_R

Sensor R – A3
Sensor L – A5

STRUCTURE: ARITHMETIC OPERATORS

❖ % (remainder): x = 7%5,	x = 2
❖ * (multiplication) : x = 2*3,	x = 6
❖ + (addition) : x = 2+5,	x = 7
❖ - (subtraction) : x = 4-1,	x = 3
❖ / (division) : x = 50/5,	x = 10

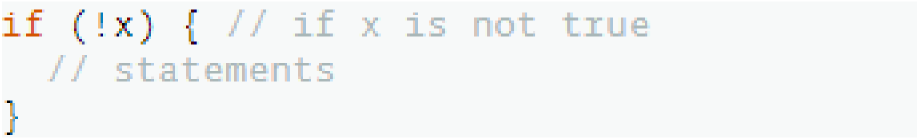
x = ?

❖ = (assignment operator)

The single equal sign “=” in the C++ programming language is called the assignment operator. It has a different meaning than in algebra class where it indicated an equation or equality. The assignment operator tells the microcontroller to evaluate whatever value or expression is on the right side of the equal sign, and store it in the variable to the left of the equal sign.

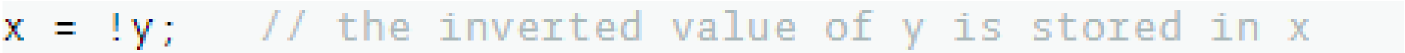
STRUCTURE: BOOLEAN OPERATORS [1]

- “!” Logical NOT
  - Can be used inside the condition of an if statement



```
if (!x) { // if x is not true
  // statements
}
```

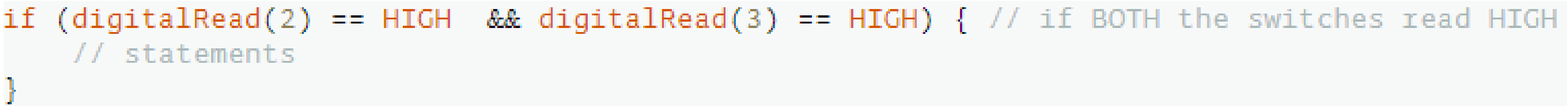
  - Can be used to invert the Boolean value



```
x = !y;    // the inverted value of y is stored in x
```

STRUCTURE: BOOLEAN OPERATORS [2]

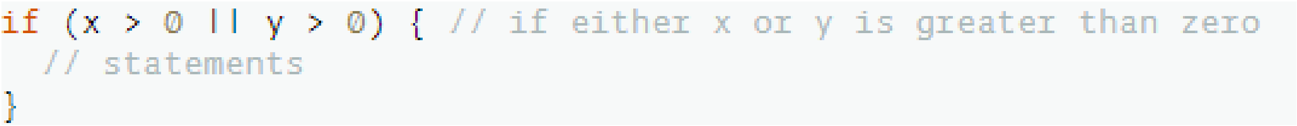
- “&&” Logical AND
  - Results in true only if both operands are true



```
if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) { // if BOTH the switches read HIGH
    // statements
}
```

Note: Here, == is the comparison operator. If the read value is HIGH, the expression “digitalRead(2)==HIGH” will return true.

- “||” Logical OR
  - Logical OR results in true only if either of the two operands is true



```
if (x > 0 || y > 0) { // if either x or y is greater than zero
  // statements
}
```

Note: Here, > is a comparison operator. If the value of x is greater than 0, the expression “x>0” will return true.

STRUCTURE: COMPARISON OPERATORS

- != (not equal to)
- < (less than)
- <= (less than or equal to)
- == (equal to)
- > (greater than)
- >= (greater than or equal to)

The result is true or false. It is recommended to compare variables of the same data type.

EXAMPLE: COMPARISON OPERATIONS

- “Yes or NO” questions about the relationship between two numbers.

Is 20 equal to 5?	No
Is 36 greater than 26?	Yes
Is 19 not equal to 90?	Yes

- Coding operator expressions.

20 == 5 ?	0
36 > 26 ?	1
19 != 90 ?	1

1 for “True”, 0 for “False”

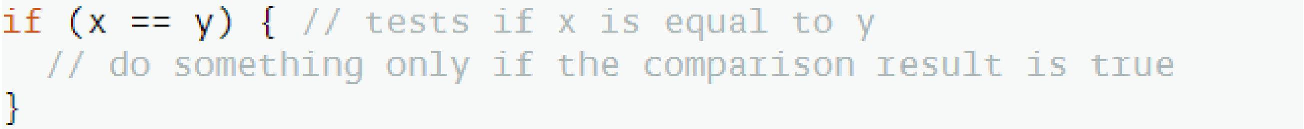
Note: The equals sign

“=“ is used to assign a value

“==“ / “!=” is used to compare values

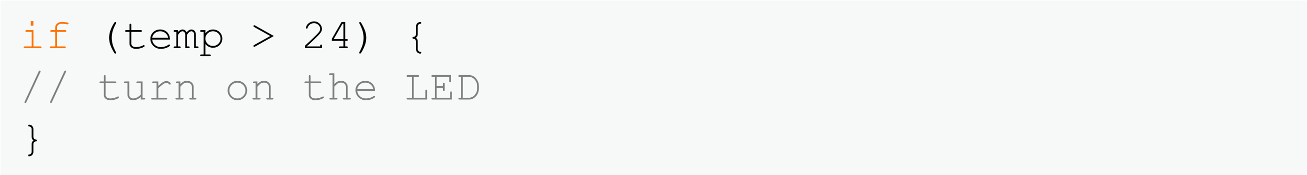
CONDITIONAL STATEMENT [1]

- **Comparison operators** are more useful with **conditional statements**.



```
if (x == y) { // tests if x is equal to y
  // do something only if the comparison result is true
}
```

- A conditional branch is lines of code that are executed only if some condition is true or not true.
  - Example code: you want to turn on an LED indicator light but only if the temperature in a room exceeds 24 degree.



```
if (temp > 24) {
// turn on the LED
}
```

CONDITIONAL STATEMENT [2]

- The if…else statement
  - Conditional branches can be made more complex by adding an else statement to an if statement.
  - Example code: you want to turn a green LED on if the temperature is less than or equal to 24 degree, but turn a red LED on if the temperature exceeds 24 degree.

```
if (temp <= 24) {
// turn on the green LED
}
else {
// turn on the red LED
}
```

CONDITIONAL STATEMENT [3]

- The if…else if statement
  - Conditionals can be made even more functional by using else if branches.
  - Example code: you want to turn on a blue LED if the temperature is less than or equal to 0 degree, a green LED if the temperature is between 0 and 24 degrees, and a red LED when exceeds 24 degree.

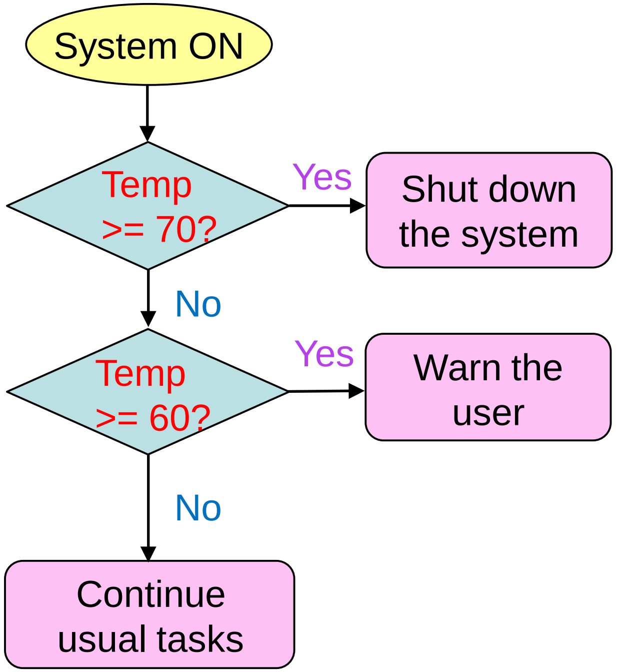
```
if (temp <= 0) {
// turn on the blue LED
}
else if (temp > 0 && temp <= 24) {
// turn on the green LED
}
else {
// turn on the red LED
}
```

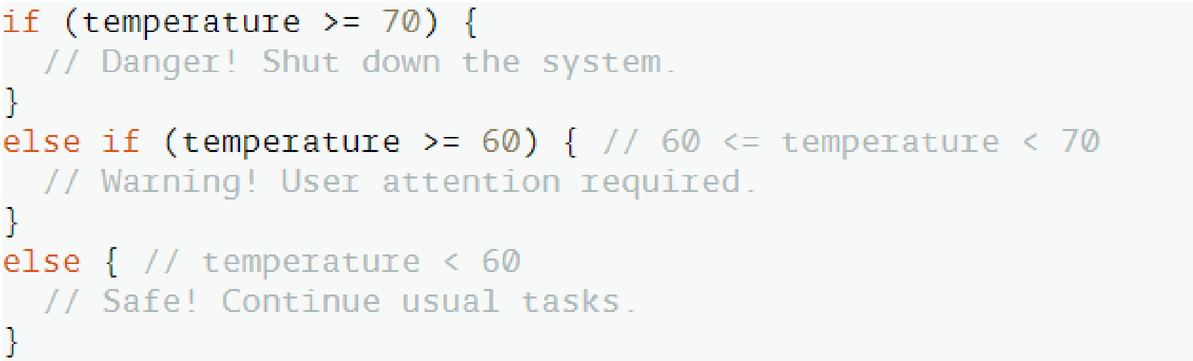
LOGIC FLOW IN PROGRAMMING

- Conditional statements and loops are powerful methods to perform different actions depending on the outcome of specific condition and to change the way of program flows.
- The structures introduced here (and others not mentioned like the *switch…case* structure) exist in most of the programming languages as they allow a programmer to translate the flow of the tasks performed by a human to code that a computer can execute.
- Remember that programming is less about learning specific syntaxes, but more about learning to think like a computer with logic flow steps.

EXAMPLE FLOW: A WARNING SYSTEM

- A **flowchart** is a type of diagram that represents a workflow or process, a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.





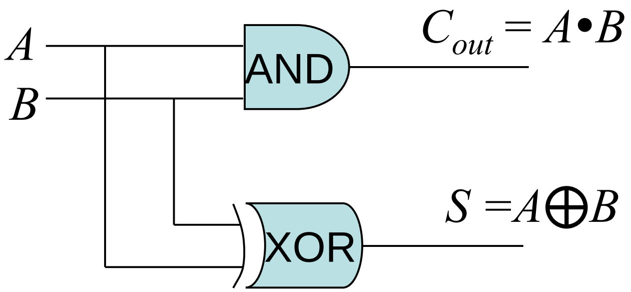
```
if (temperature >= 70) {
  // Danger! Shut down the system.
}
else if (temperature >= 60) { // 60 <= temperature < 70
  // Warning! User attention required.
}
else { // temperature < 60
  // Safe! Continue usual tasks.
}
```

LOGIC SYSTEM DESIGN

- The design procedure for a **logic circuits** starts with the problem specification and comprises the following steps:

  1) Determine required number of inputs and outputs
  2) Derive the truth table for each of the outputs based on their relationships to the input
  3) Simplify the Boolean expression for each output (Use K-map or Boolean algebra)
  4) Draw a logic diagram that represents the simplified Boolean expression
  5) Verify the design by analyzing or simulating the circuit

- **Arduino board** is used in this course to implement logic operation, replacing the basic logic gates system.



half adder implementation

TRUTH TABLE IN LOGIC DESIGN

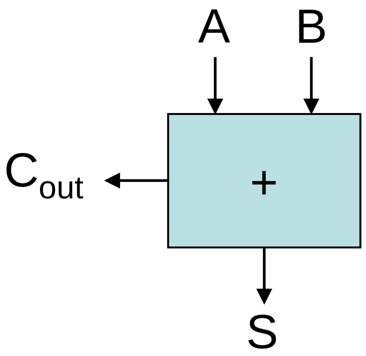
- Even the **Arduino board** is used in practice, it is important that a digital designer should learn how to build relationship between system inputs and desired outputs from a specification.
- Understanding this process allows the designer to better use **Arduino structures** (conditional statements & loops) in coding part to produce the correct output response for all input values.
- Thus, **truth table** is crucial to the success of system design.
- All the possible combinations of inputs are listed. If the truth table is known, we completely know how the circuit behave!

Truth table: half adder

A	B	S	C_{out}
0	0	**0**	**0**
0	1	**1**	**0**
1	0	**1**	**0**
1	1	**0**	**1**

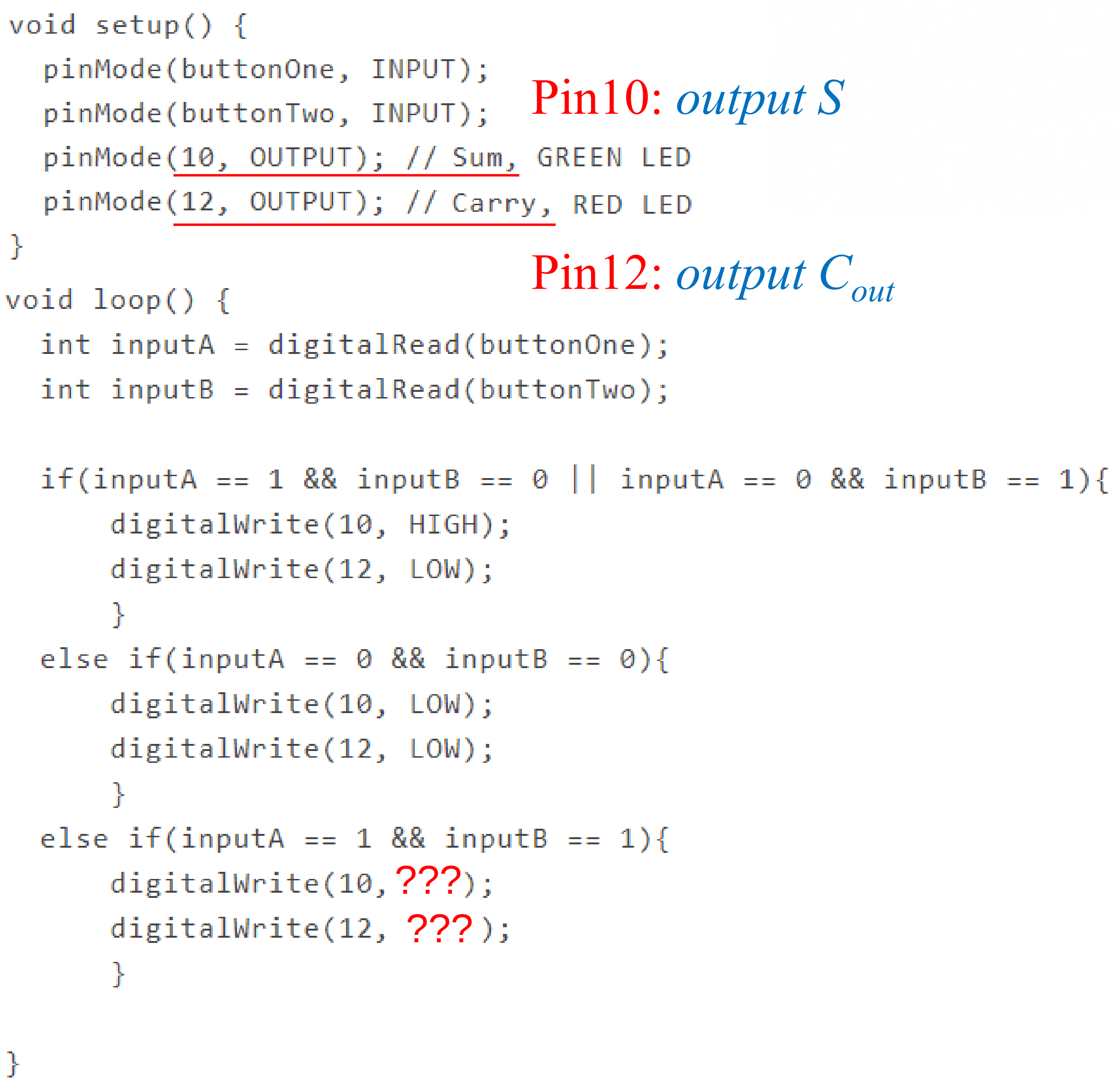
EXAMPLE CODE: HALF ADDER

- 1-bit half adder (from Lecture 13)



- Truth table

A	B	S	C_{out}
0	0	**0**	**0**
0	1	**1**	**0**
1	0	**1**	**0**
1	1	**0**	**1**



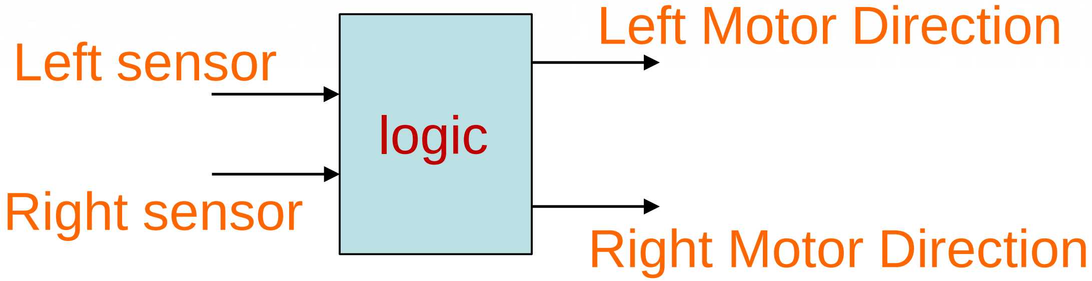
```
void setup() {
  pinMode(buttonOne, INPUT);
  pinMode(buttonTwo, INPUT);
  pinMode(10, OUTPUT); // Sum, GREEN LED
  pinMode(12, OUTPUT); // Carry, RED LED
}

void loop() {
  int inputA = digitalRead(buttonOne);
  int inputB = digitalRead(buttonTwo);

  if(inputA == 1 && inputB == 0 || inputA == 0 && inputB == 1){
      digitalWrite(10, HIGH);
      digitalWrite(12, LOW);
      }
  else if(inputA == 0 && inputB == 0){
      digitalWrite(10, LOW);
      digitalWrite(12, LOW);
      }
  else if(inputA == 1 && inputB == 1){
      digitalWrite(10,???);
      digitalWrite(12, ???);
      }

}
```

Pin10: *output S*

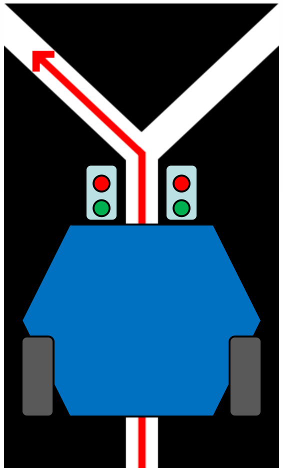
Pin12: *output* C_{out}

TWO SENSORS TRACKING

- In your Lab#06:



Turn left at the split



Sensor signal		Motor rotations		Left motor signal (to L293)		Right motor signal (to L293)	
Left	Right	Left	Right	Dir	Speed	Dir	Speed
0	0	reverse	forward	0	from your Uno-board	1	from your Uno-board
0	1	reverse	forward	0		1	
1	0	forward	reverse	1		0	
1	1	forward	forward	1		1	

(1-detect black surface)

(1-Dir forward)

SUMMARY

- Arduino programming main part: structure.
  - Sketch
  - Control Structure
  - Further Syntax
  - Arithmetic Operators
  - Comparison Operators
  - Boolean Operators

- Conditional statements and loops: powerful methods to perform different actions.

  - Comparison Operators

    != (not equal to)

    < (less than)

    <= (less than or equal to)

    == (equal to)

    > (greater than)

    >= (greater than or equal to)

  - Boolean Operators

    ! (logical not)
    && (logical and)
    || (logical or)

  - Control Structure

    for
    if
    while
    break

- Logic flow: a step-by-step approach to solving a task.
- Logic system design: your truth table.

NEXT LECTURE

- Your Final Project

QUESTIONS?