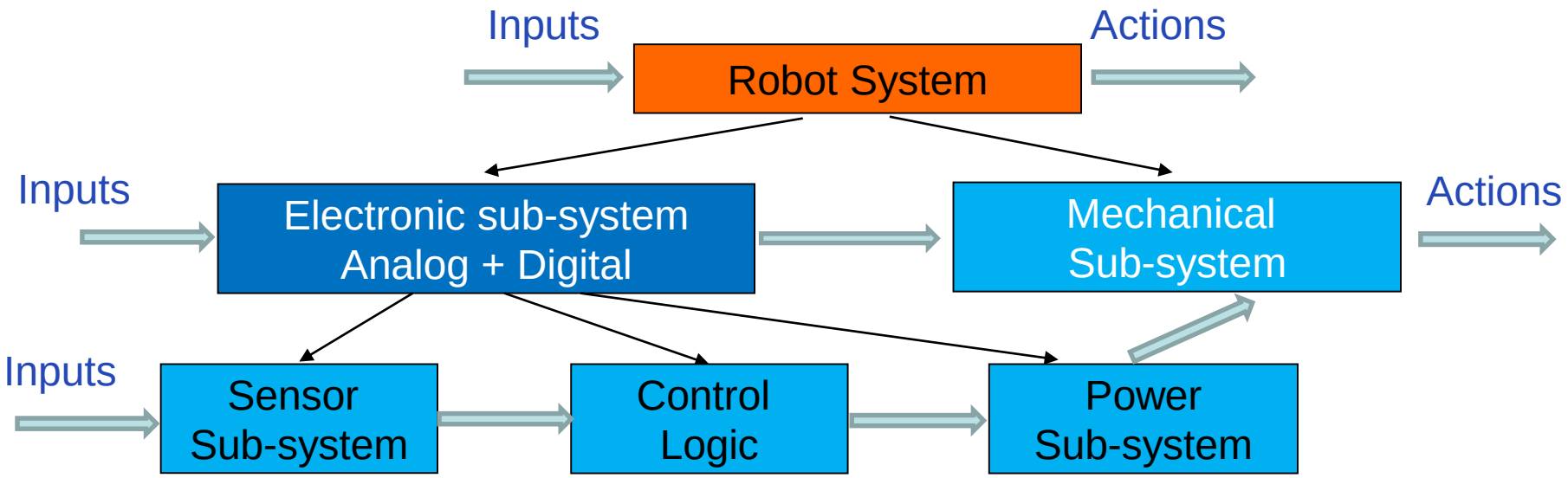
ELEC 1100: Introduction to Electro-Robot Design

Lecture 07: Programming Language part III&IV + Final Project

CHEN Wei, Dept. of ECE, HKUST

ELEC1100 ROADMAP



Sensor Basics:
Wk2: Sensor Basic

Combinational/Sequential Logic:
Wk3: Robot Brain: Logic Gate & Logic Operation
Wk4:Sequential Logic
Wk5:MCU & Programming Language

Basic electronics:
Wk1: Basic Electronics & DC Sources

Motor Power Supply:
Wk1: Pulse Signal & PWM Control
Wk2: Transistor and H-Bridge

STRUCTURE

- The elements of Arduino (C++) code.
  - Sketch
  - Control Structure
  - Further Syntax
  - Arithmetic Operators
  - Comparison Operators
  - Boolean Operators
  - Pointer Access Operators
  - Bitwise Operators
  - Compound Operators

SKETCH

❖ setup()

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

❖ loop ()

- The loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond.

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

CONTROL STRUCTURE

❖ if … else …

- Description: The if statement checks for a condition and executes the proceeding statement or set of statements if the condition is 'true'.
- Syntax:

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

- Example Code

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

CONTROL STRUCTURE

❖ for

- Description: The for statement is used to repeat a block of statements enclosed in curly braces.
- Syntax: for (initialization; condition; increment) { //statement(s);
  }

  - The initialization happens first and exactly once.
  - 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.
- Example Code

```
void loop()
{
   for (int i=0; i <= 255; i++){
      analogWrite(PWMpin, i);
      delay(10);
   }
}
```

CONTROL STRUCTURE

❖ break

- Description: break is used to exit from a for, while or do…while loop, bypassing the normal loop condition. It is also used to exit from a switch case statement.
- Example Code

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

FURTHER SYNTAX

- “# include” is used to include outside libraries in your sketch.
- 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.
  - block comment: “/* ….multiple lines…. */”
  - single line comment: “// single line”
- “;” semicolon is used to end a statement.

FURTHER SYNTAX

- "{}" curly braces are a major part of the C programming language with following main use
  - Functions

```
void myfunction(datatype argument){
  statements(s)
}
```

  - Loops

```
for (initialisation; termination condition; incrementing expr)
{
 statement(s)
}
```

  - Conditional Statements

```
if (boolean expression)
{
 statement(s)
}

else if (boolean expression)
{
 statement(s)
}
```

BOOLEAN OPERATORS

- "!" logic NOT
  - Logic NOT results in a True if the operand is false and vice versa
  - Example Code
    - Used inside the condition of an if statement

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

    - Used to invert the Boolean value

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

BOOLEAN OPERATORS

- "&&" Logical AND results in true only if both operands are true
  - Example Code

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

  - Here, == is the comparison operator. If the read value is HIGH, the expression "digitalRead(2)==HIGH" will return true.

BOOLEAN OPERATORS

- "||" Logical OR results in true only if either of the two operands is true
  - Example Code

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

  - Here, > is a comparison operator. If the value of x is greater than 0, the expression "x>0" will return true.

ARITHMETIC OPERATORS

- % (remainder) : x = 7%5, so x=?
- * (multiplication) : x = 2*3
- + (addition) : x = 2+5
- - (subtraction) : x = 4-1
- / (division) : x = 4/5
- = (assignment operator)

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.

COMPOUND OPERATORS

- &= (compound bitwise and)
- *= (compound multiplication)
- ++ (increment)
- += (compound addition)
- -- (decrement)
- -= (compound subtraction)
- /= (compound division)

++

- Syntax
  - x++; // increment x by one and returns the old value of x
  - ++x; // increment x by one and returns the new value of x
- Example Code

```
x = 2;
y = ++x;       // x now contains 3, y contains 3
y = x++;       // x contains 4, but y still contains 3
```

+=

- This is a convenient shorthand to perform addition on a variable with another constant or variable.
- Syntax
  - x +=y; // equivalent to the expression x = x+y
- Example Code

```
x = 2;
x += 4;       // x now contains 6
```

EXAMPLE

❖ LED Fading

```
int ledPin = 9;    // LED connected to digital pin 9

void setup() {
  // nothing happens in setup
}

void loop() {
  // fade in from min to max in increments of 5 points:
  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }

  // fade out from max to min in increments of 5 points:
  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }
}
```

FUNCTIONS

- Digital I/O
- Analog I/O
- Time
- Math
- Trigonometry
- Characters
- Random Numbers
- Bits and Bytes
- External Interrupts
- Interrupts
- Communication
- USB

DIGITAL I/O

- pinMode()
  - Configure the specified pin to behave either as an input or an output pin
  - Syntax: pinMode(pin, mode);
  - pin: the number of the pin whose mode you wish to set
  - mode: INPUT or OUTPUT
- digitalWrite()
  - Write a HIGH or a LOW value to a digital pin.
  - Syntax: digitalWrite(pin, value);
  - Example Code

```
void setup()
{
  pinMode(13, OUTPUT);          // sets the digital pin 13 as output
}
void loop()
{
  digitalWrite(13, HIGH);       // sets the digital pin 13 on
  delay(1000);                  // waits for a second
  digitalWrite(13, LOW);        // sets the digital pin 13 off
  delay(1000);                  // waits for a second
}
```

DIGITAL I/O

- digitalRead()
  - Reads the value from a specified digital pin, either HIGH or LOW
  - Syntax: digitalRead(pin)
  - Example Code

```
int ledPin = 13;    // LED connected to digital pin 13
int inPin = 7;      // pushbutton connected to digital pin 7
int val = 0;        // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin 13 as output
  pinMode(inPin, INPUT);        // sets the digital pin 7 as input
}

void loop()
{
  val = digitalRead(inPin);     // read the input pin
  digitalWrite(ledPin, val);    // sets the LED to the button's value
}
```

ANALOG I/O

- analogRead()
  - Reads the value from the specified analog pin, with 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit.
  - Syntax: analogRead(pin)
  - Returns: int(0 to 1023).
- analogWrite()
  - Writes an analog value (PWM wave) to a pin. The pin will generate a steady square wave of the specified duty cycle. The frequency of the PWM signal on most pins is approximately 490 Hz.
  - Syntax: analogWrite(pin, value)
    - Value: the duty cycle: between 0 and 255.

TIME

- delay()
  - Pauses the program for the amount of time (in milliseconds) specified as parameter.
  - Syntax: delay(ms)
- millis()
  - Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.
  - Syntax: time = millis()

MATH & TRIGONOMETRY

- abs()
- constrain()
- map()
- max()
- min()
- pow()
- sq()
- sqrt()

- cos()
- sin()
- tan()

USER-DEFINED FUNCTIONS

- Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program.
- Advantages:
  - Functions help the programmer stay organized. Often this helps to conceptualize the program.
  - Functions codify one action in one place so that the function only has to be thought out and debugged once.
  - This also reduces chances for errors in modification, if the code needs to be changed.
  - Functions make the whole sketch smaller and more compact because sections of code are reused many times.
  - They make it easier to reuse code in other programs by making it more modular, and as a nice side effect, using functions also often makes the code more readable.

USER-DEFINED FUNCTIONS

❖ Example code

```
void setup(){
  Serial.begin(9600);
}

void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = myMultiplyFunction(i, j); // k now contains 6
  Serial.println(k);
  delay(500);
}

int myMultiplyFunction(int x, int y){
  int result;
  result = x * y;
  return result;
}
```

USER-DEFINED FUNCTIONS

❖ Example code

```
void carMoveByDir(int carDir) {
  switch (carDir) {
    case FORWARD:
      LDir = HIGH;
      RDir = HIGH;
      break;
    case REVERSE:
      LDir = LOW;
      RDir = LOW;
      break;
    case LEFT:
      LDir = LOW;
      RDir = HIGH;
      break;
    case RIGHT:
      LDir = HIGH;
      RDir = LOW;
      break;
  }
  setMotorDir();
}
```

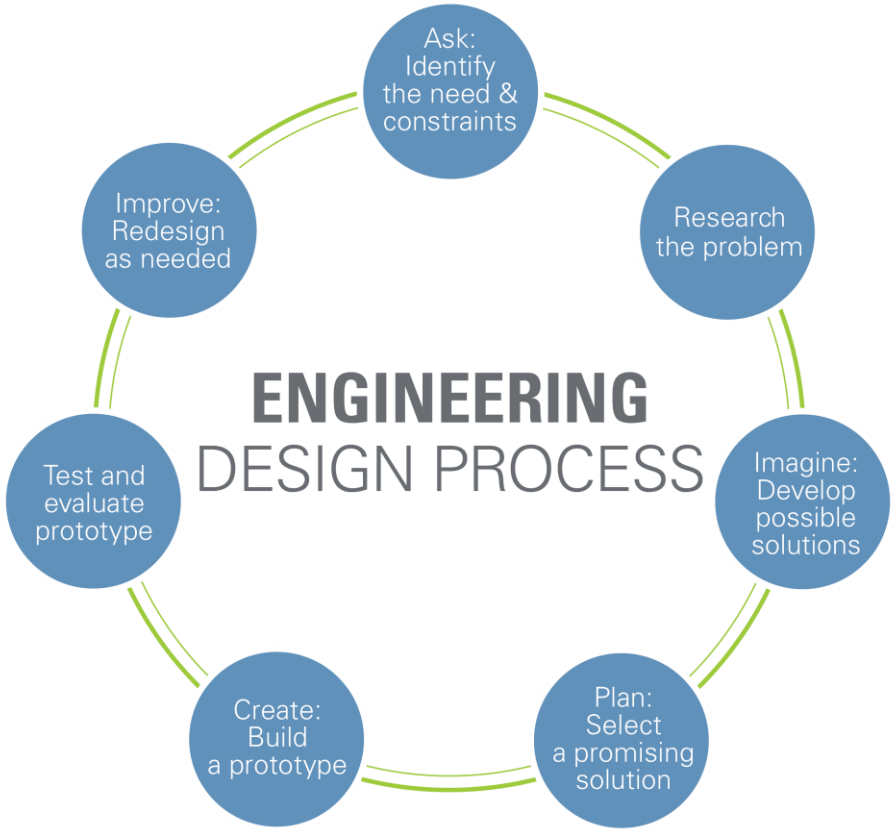
```
void setMotorDir() {
  digitalWrite(pinLDir, LDir);
  digitalWrite(pinRDir, RDir);
}
```

PROJECTS

- Project Design
- Hardware Design
- Software Design

ENGINEERING DESIGN PROCESS

- Our project is not that complex, but it gives a good opportunity to learn engineering design.



ENGINEERING DESIGN PROCESS

- Read carefully about the project requirement.
- Think about possible solutions (hardware & software).
- Pick one and build a prototype.
- Test and evaluate.
- Improve and re-design as needed.

HARDWARE DESIGN

- Freedoms we have:
  - Car design
  - Use of sensors
  - Use of Arduino
- Questions to think about:
  - Should the sensors be close or far away from the wheels?
  - How many sensors should I use? 2 or 3?
  - How many I/O ports should I use from Arduino?
  - What are the advantages and disadvantages for different options?

SOFTWARE DESIGN

- Things we cannot change
  - setup()
  - loop()
- Freedoms we have:
  - Everything inside the loop()!
  - The map has several sections, so
    - Should I use the same speed for all sections?
    - Should I use the same logic for all sections?
    - Are there any operations I should functionalize?

SOFTWARE DESIGN: SOME REMINDERS

❖ Turning

- There are several splits on the map where the car should turn.
- Image that in the first split, the car senses sth like “white white” or “white black white”, depending the number of sensors you are using. How can we guarantee that the car will turn left and leave the split, i.e., not bouncing back to the split.
- For the right angle turn, how can we guarantee that the car has sufficient time to turn before running out of the track?
- Between the 2-sensor and 3-sensor design, which one is more stable at turning point?

SOFTWARE DESIGN: SOME REMINDERS

- Moving back
  - To the end of the track, the car needs to move back after sensing the wall.
  - How can we guarantee the car will move back with the required distance?
  - If we use sensors, what if the car is not moving straightly?
  - Other solutions?

Questions ?!