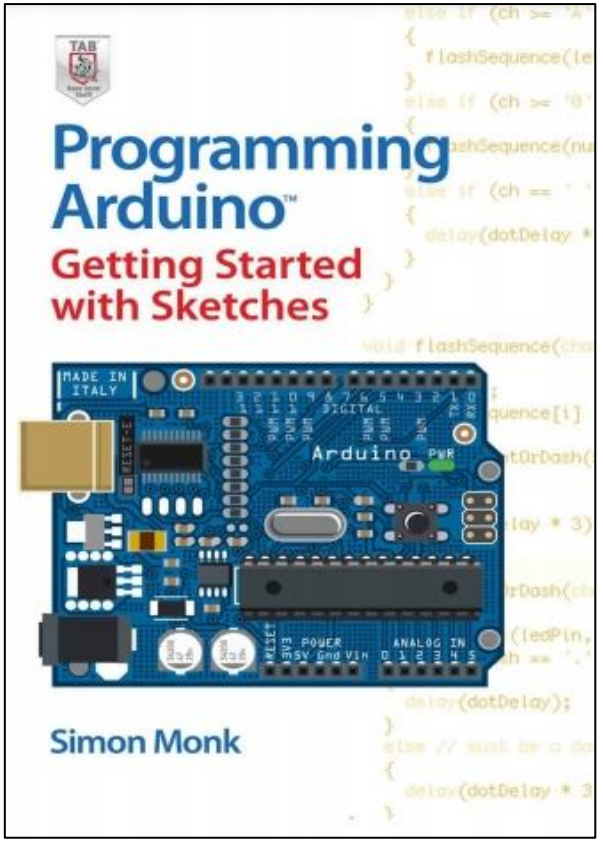
FabLab
SINGAPORE POLYTECHNIC

EP1000

Embedded Systems 2 Programming

Arduino System

- Programs written for the Arduino system are text files called sketches (extension .ino)
- The programming language used is based on Processing, which is loosely based on the C++ syntax.



A simple and easy introduction to Programming Arduino. Available in the SP Library

Sketches

- All Arduino sketches have 2 functions:
  - setup()
    code is executed only once
    used for initialisation and setup of I/O
  - loop()
    code is executed continuously
    application code is placed here

```
14
15  /*
16      Basic Sketch
17  */
18
19
20  void setup()
21  {
22      // code is executed only once
23  }
24
25  void loop()
26  {
27      // code is run continously
28  }
29
30
```

Variables

- Variables are memory set aside to hold changing data.
- Variables use different amounts of memory depending on the data type used.
- Common data types are
  - char 8-bit
  - int 16-bit
  - float 32-bit
  - String stores a sequence of characters
- We use identifiers (names) to name the variable and locate it.
- Use conversion functions to convert between them

Data Types

- array
- bool
- boolean
- byte
- char
- double
- float
- int
- long
- short
- size_t
- string
- String()
- unsigned char
- unsigned int
- unsigned long
- void
- word

Operators

Arithmetic Operators

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

Boolean Operators

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

Bitwise Operators

- & (bitwise and)
- << (bitshift left)
- >> (bitshift right)
- ^ (bitwise xor)
- | (bitwise or)
- ~ (bitwise not)

Compound Operators

- %= (compound remainder)
- &= (compound bitwise and)
- *= (compound multiplication)
- ++ (increment)
- += (compound addition)
- -- (decrement)
- -= (compound subtraction)
- /= (compound division)
- ^= (compound bitwise xor)
- |= (compound bitwise or)

Use the KISS principle!

Operators work only with similar data types

Use conversion functions to help.

Boolean

- Boolean variables have only 2 values: True/False
- Boolean expressions have only 2 results: True/False
- Comparison operators give a Boolean result
- Boolean operators work only with Booleans

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)

Used in **Conditionals, Loops**

Conditional: if … else

- Control Structure which checks the condition expression and if true executes the following code block.
- Condition Expression is a Boolean expression.
- When used with the else, control transfers to the else block.
- Can have nested or have multiples else-if conditionals for more granular control.

```
1
2   if (temperature < 30)
3   {
4       // increase heat
5       ...
6   }
7   else
8   {
9       // maintain
10      ...
11  }
12
13
```

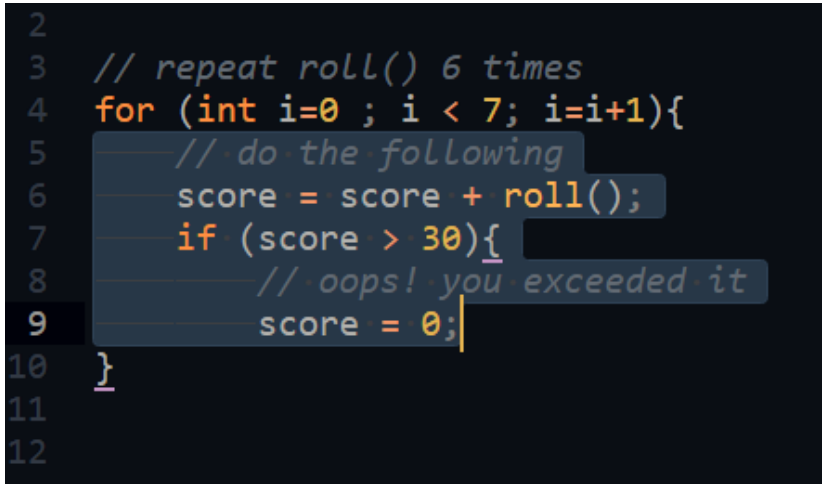
Conditional: switch … case

- Control Structure which checks the value in **switch** (preferably ordinal) and transfers control to the matching **case** code block.
- Each case code block must be terminated with a **break**.
- Each case must match exactly.
- Control is transferred to the **default** code block (if any) if there is no match.

```
1
2   die = roll();        // rolls a die
3
4   switch(die){
5       case 6:
6           // scores and rolls again
7           score = score + 6 + roll();
8           break;
9       case 1:
10          // forfeits turn (no score)
11          break;
12      case 4:
13          // deducts points
14          score = score - 4;
15          break;
16      default:
17          // all other values
18          score = score + die;
19          break;
20  }
21
```

Loop: for

- Structure of for
  - Initialization
  - Conditional expression
  - Increment
- Used when we know exactly the number of times we wish to loop.



```
2
3   // repeat roll() 6 times
4   for (int i=0 ; i < 7; i=i+1){
5       // do the following
6       score = score + roll();
7       if (score > 30){
8           // oops! you exceeded it
9           score = 0;
10  }
11
12
```

Loops a **fixed** number of times

Loop: while

- Tests conditional expression, if true the code block is executed.
- Indefinte loop, code block is executed zero, once or many times.
- If the condition results always in True, we have an endless loop.

```
2
3   // execute code when switch is pressed
4   while ( ispressed(key1) )
5   {
6       // key is pressed
7       count = count + 1
8       ...
9   }
10
11  // what will be the default
12  // condition state?
13
14
```

Loops a 0, 1 … n number of times

Loop: do … while

- Executes the code block before testing the conditional expression.
- If conditional expression is true the code block is repeated.
- Indefinte loop, code block is executed once or many times.
- If the condition results always in True, we have an endless loop.

```
2
3   // execute code when switch is pressed
4   do
5   {
6       // control light intensity
7       value = value - 1;
8       light(value);
9       // check brightness
10      lightIntensity = measureBrightness();
11  }
12  while ( lightIntensity > 25 )
13
14  // what happens if the lightIntensity
15  // was originally 10 before the loop
16  // was entered
17
18
```

Loops a 1 … n number of times

Control: break

- When used in a loop, break exits the loop, control transfers to next statement after loop.
- break is also used to transfer control out of a matching case in a switch statement.
- Control: continue ignores the remaining statements and transfers control to the loop condition.
  (not commonly used)

```
1
2  // read the sensor
3  while (readSensor() > 25){
4      // adjust the settings
5      value = value - 1;
6      result = setLight(value);
7
8      if (result == -1)
9      {
10         // error has occured
11         break;
12     }
13 }
14
15 // what is the state here?
```

Functions

- A function is identified using ()
- A function is a block of code that can accept parameters.
- Executes the code when called, returns a single value as it's name.
- return is used to return the value in the indicated data type.

```
1
2   /* function
3       name: cube
4       parameters: int value
5       returns: int
6   */
7   int cube(int value){
8       result = value * value * value;
9       return result
10  }
11
12  ...
13
14  // execution
15  myAnswer = cube(4);
16
17  int data = 6;
18  yourAnswer = cube(data);
19
20
```

Class

- A user-defined data type that is used to create objects.
- An object has
  - attributes (constants, variables)
  - methods (functions)
- An object's attributes and methods are accessed using the dot (.) operator
- Classes are predominantly used in code libraries

Classes should be identified using a starting uppercase character e.g. Serial

Directive: #define

- #define is a compiler directive and not a code statement.
- Does not end with a semi-colon
- Used to name a constant and assign the value
- const is the preferred method of defining constants

```
1
2  // Replace all occurences of LED_RED
3  // with the value 7
4  #define  LED_RED  7
5
6  // alternative
7  // const is a keyword
8  const int LED_RED = 7;
9
```

Constants should be identified using uppercase

Directive: #include

- #include is a compiler directive and not a code statement.
- Does not end with a semi-colon
- Instructs the compiler to read and insert code from the target file

```
1
2  // include the system library
3  // (found in a folder on the
4  // path)
5  #include <wire.h>
6
7  // include the local library
8  // found in the local folder
9  #include "mystepper.h"
10
11 // libraries are external code
12 // to help in your projects
```

- < file > indicates system library, found along the library path
- " file " indicates local file in same folder

EP1000

Embedded Systems 2 Programming

End