Arduino notes

This is not a tutorial, but a collection of personal notes to remember the essentials of Arduino programming.
The program fragments are snippets that represent the essential pieces of code, in an example.
jean-claude.feltes@education.lu

Arduino web site:
http://arduino.cc

IDE

- Tools – Board – select board
- Tools – Serial port – select port (normally /dev/ttyUSBx under Linux, COMx under Windows)

Remember that under File – Examples you find many inspiring examples!

Help: Right klick – find in reference

The Serial monitor helps to do debugging

Serial commands

```
Serial.begin(9600);
Serial.print(x);
Serial.println(“HELLO“);
```

Arduino C language

Basic Arduino C:

https://www.arduino.cc/reference/en/

Libraries:
https://www.arduino.cc/en/Reference/Libraries

Program structure

Each program has 3 or more parts:

- At the beginning you find:
  - Library include statements
  - Global variable declarations

- In the first function setup() port or sensor pins are initialized
  Take care: as setup is a function, variables declared here are not „seen“ outside this function! Global variables must be declared before this, outside a function!

- The function loop contains the main program loop

- Additional functions for special tasks like reading a sensor etc.

Variables

https://www.arduino.cc/reference/en/#variables

Variables are declared this way:

```
<type> <name> = startvalue;
```

like

```
int analogPin = 0;
```

Details are found here: https://www.arduino.cc/reference/en/

The scope (visibility) of a variable is the function where it is declared. Outside, it is invisible. Only variables declared at the beginning of the program (before the setup function) are global, that means accessible from everywhere.

Some special remarks:

boolean x = true (not True!) is the same as x=1
For boolean variables, print uses 0 and 1, not false and true.

char

```
char a = 'a';           // correct, is a
char b = "b";           // wrong!  Use single quotes!
char c = 99;            // correct, is c
```

Astonishingly, char is a signed datatype (with values from -128 to +127).
To use ASCII characters > 127, use the byte variable!

byte

The value of a Byte can be assigned in decimal, hex or binary notation

```
byte x = 0x56;
byte y = B01011;
```

or even like this:

```
byte z = 'a';
```

Serial.print prints char variables as characters, and bytes as numbers.

Arrays

```
float x[5];         // declares an array of 5 floating point values
```

Strings

https://cdn.arduino.cc/reference/en/language/variables/data-types/stringobject/

Strings are objects. In contrary to the other variables, strings are declared by using „String“ with a capital S (All objects are noted with capital letters).

```
String s = “Hello world“;
```

Strings have methods (like all objects) that are very practical to use.
For all search and replace operations it is important to remember that strings are zero based arrays. That means that the first character has the index 0.

Another speciality of C is that s.substring(2,4) means the substring from the third to the fourth character, not as you could be tempted to think, from the 2nd to the 4th, or the 3rd to the 5th.
Why?
As the index starts with 0, index 2 means the 3rd character.
Probably because the C implementation of the substring algorithm works internally with a
for (int i, i<lastindex, i++) construction, the last index is not taken into account and the loop stops before i=4 in our case.
This is similar to Python where s[2:4] = 'll' if s = 'hello world'

Examples:

```
s = "Hello world";

L= s.length();              //   11
s.toUpperCase();            //   HELLO WORLD
s.toLowerCase();            //   hello world
s.replace(' ', '-');        //   hello-world

x = s.charAt(1);            //   101 = 'e'
y = s.indexOf('e');         //   1    first occurence of e at index 1
z = s.lastIndexOf('l');     //   9    last occurence of l
j = s.startsWith("hel");    //   1 (true)
k = s.endsWith("LD");       //   0 (false)
t = s.substring(2,4);       //   ll
u = s.substring(2);         //   llo world
```

Static variables
A normal variable declared inside a function is local to that function and loses it's value once the function is left.
A static variable declared inside a function is local to that function, but „remembers“ it's value between function calls.

Example code to test this:

```
void loop() {
    static int x;
    int y;
    x += 1;
    y += 1;
    Serial.println(x);
    Serial.println(y);
    delay (1000);
}
```

The value of x increases with every loop, whilest y remains at the value 1.

Volatile variables
These are used inside interrupt service routines (ISR) and reside in RAM space. Normal variables could get corrupted when used in an ISR.

Operators

Some special C operations:

```
x += 2;           // abbreviation for   x = x+2
x++;             //  abbreviation for   x = x+1
x--;             //  abbreviation for   x = x-1


c = (x==y)       //  c = 1 if   x is equal to y
c = (x != y)           //  c = 1 if  x is not equal to y
```

&& = boolean AND
|| = boolean OR
! = boolean NOT

Conditions

If … else
One condition:

```
if (x>5){
        Serial.println(x);
        }
```

2 conditions:

```
if (x>5){
        Serial.println(x);
        }
    else {
        Serial.println('*');
        }
```

Multiple conditions:

```
 if (x>5){
        Serial.println(x);
        }
    else if (x==1) {
        Serial.println('*');
        }
    else {
        Serial.println('-');
    }
```

Switch … case … break
This is easyer for multiple conditions:

```
switch(x){
      case 2:
         Serial.println('*');
         break;
      case 3:
         Serial.println("***");
         break;
      default:
         Serial.println(x);
     }
```

For Loops

Ascending loop 0...9:

```
for (int i=0; i<10; i++){
    Serial.print(i);
    }
```

Descending loop 10 … 1:

```
for (int i=10; i>=1; i--){
    Serial.println(i);
    }
```

Ascending loop with increment 2: 1, 3, 5, 7, 9

```
for (int i=1; i<=10; i+=2){
    Serial.println(i);
    }
}
```

Any operation for the index can be done! This gives the possibility to generate any list of values

This gives the numbers 1, 10, 100, 1000, 10000:

```
for (long i=1; i<=10000; i*=10){
    Serial.println(i);
    }
}
```

This gives 2.00, 3.56, 6.34

```
for (float x=2.0; x<=10; x=x*1.78) {
    Serial.println(x);
    }
}
```

While and Do .. while loops

While generates 0...4:

```
int i=0;
while (i <= 4) {
    Serial.println(i);
    i++;
    }
```

The same can be done with a do … wile:

```
int i=0;
do {
    Serial.println(i);
    i++;
    }
    while (i <= 4);
```

What is the difference between a while loop and a do...while loop?
The do … while structure checks the condition at the end. This means that the do...while loop is executed at least once. The while loop is not executed, if the condition at the beginning is never met.

Breaking a loop

Sometimes a loop must be left prematurely, depending on an event.

The following snippet shows how to leave a while loop when the signal on digital input 0 turns LOW:

```
int i=0;
while(1){
    Serial.println(i);
    i++;
    if (digitalRead(0)==0) {
      break;
      }
    }
```

Functions

This example calculates the squares of natural numbers:

```
long x=0;
long y;

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

void loop() {
    y = mysquare(x);
    Serial.println(y);
    x++;
    delay(500);
}

int mysquare(int x){
  return x*x;
  }
```

The function is defined by this:

```
int mysquare(int x){
  return x*x;
  }
```

with the syntax

```
<return var type> <function name> (typed arguments) {
      statements
      return >return var>
      }
```

If no value is returned we omit the return statement and use „void“ as type:

```
void mylinefeed(void){
    Serial.println();
    Serial.println();
    }
```

Function prototypes

In C, normally the compiler must be informed about the functions if they are not declared before they are called. This is done by function prototypes at the beginning of the code, like this in our example:

```
// function prototypes:
int mysquare(int x);
void mylinefeed(void);
```

These prototype statements can also be put into separate files called **header files**.
(This is usually done if the functions form a library).

In this case, the header file must be included at the beginning of the code:

```
      #include "myheader.h"
or
      #include <myheader.h>
```

The first kind of include is used for header files in the project path, the second for the standard path of libraries.

When all functions are included in one file, Arduino automatically does the job without the declaration of function prototypes, but if the code is spread over several files, the prototyping must be done.

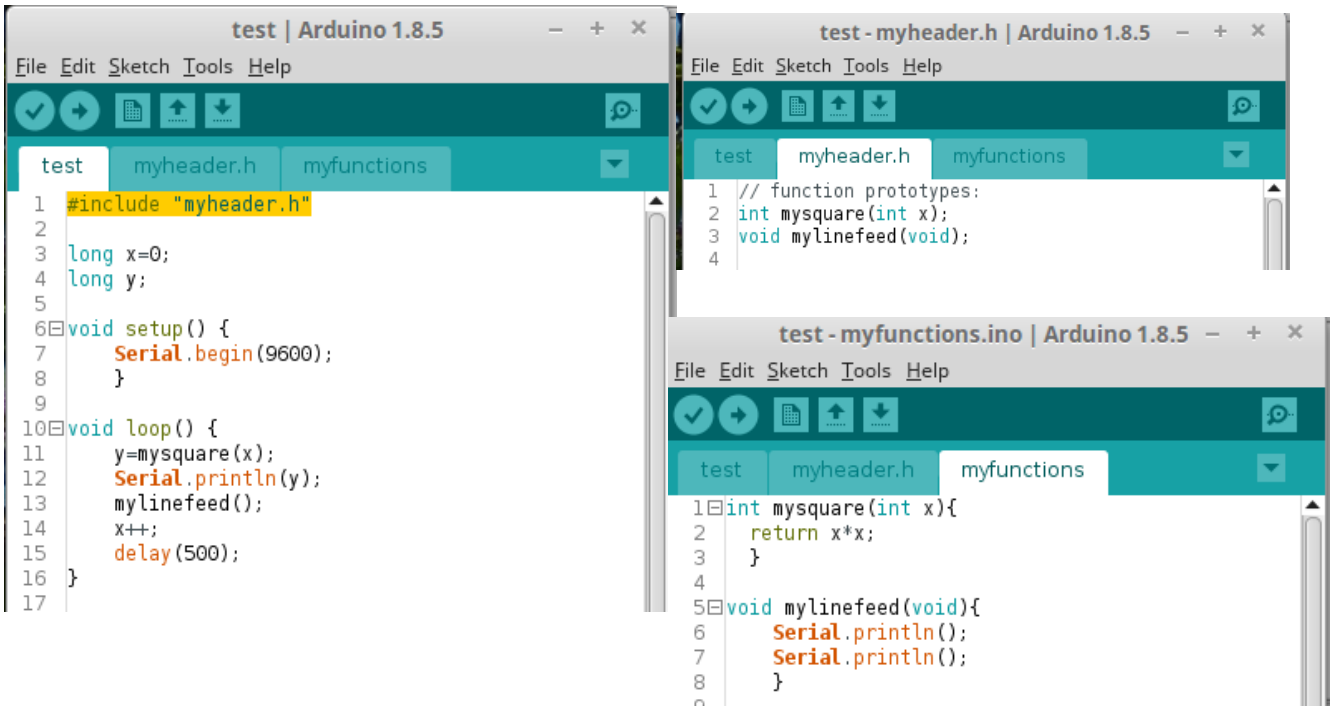
Using tabs in the Arduino editor

https://liudr.wordpress.com/2011/02/16/using-tabs-in-arduino-ide/

For bigger projects it may be useful to spread the code over several files so it is easyer to overview and to edit.

The tab menu is available by klicking on the small triangle at the right in the Arduino IDE.

A small example:



Mathematical functions

- All **trigonometric functions** (sin, cos, tan) and their reverse functions (asin, acos, atan) are available and **use radians**, not degrees.
  The data type is double.

- Other functions:
  - pow(x,y) = x^y
  - exp(x) = e^x
  - log(x) = ln(x), natural logarithm
  - log10(x) = log(x), to the base 10
  - square(x) = x^2
  - sqrt(x) = $\sqrt{x}$
  - fabs(x)§ = |x|

Binary functions

- x & y = x AND y
- x | y = x OR y
- ~x = NOT x

- x << 3 = x shifted 3 bits to the left
- x >> 2 = x shifted 2 bits to the right

Time

To delay the program flow we have

```
delay(x);               // in ms
delayMicroseconds(x);   // in us
```

During the delay, the program is stopped.

Arduino provides the time since the board has started with the **millis()** function:

```
unsigned long t;

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

void loop() {
      t=millis();
      Serial.println (t);
      delay(500);
}
```

This little program prints the uptime every 500ms.

Beware:
The variable t (unsigned long) has an overflow after t = 2^{32} ms = 4294967295ms
t = 1193h = 49.7 days !
There is even a shorter delay (only 9 hours!) when things can go wrong:
http://www.cibomahto.com/2008/04/analysis-of-the-millis-function/

For longer intervals a variable days could be used that would be incremented every 24h. If the millis function could be resetted, but that is not the case.
https://www.baldengineer.com/arduino-how-do-you-reset-millis.html
So other tricks must be used, for example react to the rollover of the unsigned long variable, and counting the 49.7days intervals. Fortunately such long times are rarely needed.

For more precise timing, a function micros() that counts the microseconds is available.
The same program with **micros()**

```
unsigned long t;

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

void loop() {
    t=micros();
    Serial.println (t);
    delay(500);
}
```

gives for example this result:

```
8
500076
1000368
1500716
2001056
2501392
3001740
```

Timing without delay

The millis() function can be used to time without any dead time of the processor:

```
unsigned long previousMillis = 0;
const long interval = 1000;


void setup() {
      …
}


void do_it(){
   // here comes what you want to do every interval milliseconds
   ...
}

void loop() {
     unsigned long currentMillis = millis();
     if (currentMillis - previousMillis >= interval) {
        do_it();
        previousMillis = currentMillis;
        }
}
```

Literature:

[1] Julien Bayle: C Programming for Arduino, Packt.
This is an excellent and very practical introduction