Arduino notes: Serial communication

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

Official doc on Serial

https://www.arduino.cc/reference/en/language/functions/communication/serial/

Using the hardware serial port of a Teensy

On my Computer (with Linux) the Teensy shows with two items in the Serial port menu of Arduino:

*/dev/*ttyACM0 Serial (Teensy)
/dev/ttyACM0 Serial (Teensy 2.0)

Why? If someone knows, please tell me.

When debugging a sketch I had the impression that sometimes the first one worked, and sometimes the last. Anyway, I was tired of fumbling with the serial monitor and I attached the hardware serial port of the Teensy via a small interface to the serial port of my computer (yes, it still has got one and sometimes I am glad it has!)

But now I would have to change all Serial statements in my program. Really?

No, a little trick helped:

```
#define Serial Serial1
```

For the compiler this replaces every „Serial“ in the program with „Serial1“, at compile time.

Line input function

In BASIC there is a very useful function that you can use like this:

```
Input "What is your name?", userInput
```

It waits until the user hits <Enter> and returns the input.

Why can‘t I find this in the Arduino library?

There are Serial.readString() and Serial.readStringUntil(), but they do not behave exactly as I expected. There are solutions to be found in the forums that helped me build some funtions that behave approximately like the input function.

All of them use a buffer for the string that is filled as you type, until you hit the <Enter> button.

When this happens, the receiving loop is stopped and the string is returned to the caller of the function.

Here is the code:

```
String readLine(){

  String sdata = "";
  byte ch;

    while (1){
       if (Serial.available()) {
          ch = Serial.read();

          Serial.print((char)ch);   //echo
          sdata += (char)ch;
          if (ch=='\r') {  // end of line on CR
               sdata.trim();
               break;
          }
       }
    }
    Serial.println();
    return sdata;
}
```

Some details:

- The Serial.print((char)ch) is for echoing the incoming characters back to the remote terminal, so the user does not have to type blindly
- The Serial.println at the end is to do a line feed. Without it the cursor is moved to the beginning of the input line.
- In contrary to what I thought at first, the <Enter> key generates a ‚\r‘ (carriage return) and not a ‚\n‘ (line feed), at least my GTKTerm under Linux does this.
  Eventually the code could be adapted to another end of input character.

The function can be used like this:

```
String sdata =  readLine();
```

It is important to realize that, in contrary to the functions of the Arduino library, this funtion is blocking, it waits until the user enters something (this was exactly what I wanted!).

But if there is no entry, the whole program is blocked. It would be good to have a function with a timeout. This is not difficult:

```
String readLineWithTimeout(long timeoutms){
  String sdata = "";
  byte ch;
  long t1 = millis();

    while (1){
       if (Serial.available()) {
          ch = Serial.read();

          Serial.print((char)ch);   //echo
          sdata += (char)ch;
          if (ch=='\r') {  // end of line on CR
               sdata.trim();
               break;
```

```
            }

         }
         if (millis() - t1 > timeoutms){
                sdata.trim();
                break;
         }
      }
      Serial.println();
      return sdata;
}
```

The receiving loop is left if the user presses <Enter> or if a certain amount of milliseconds has passed

It is used like this:

```
String sdata =  readLineWithTimeout(5000);
```

for a timeout of 5 seconds.

The BASIC funtion allows a prompt to be passed to the function.

This can be done easily with an additional print:

```
String inputLine(String prompt, long timeoutms){
     Serial.print(prompt);
     return readLineWithTimeout(timeoutms);
}
```

It is used like this:

```
String sdata = inputLine("INPUT STRING:", 10000);
```

Clearing the input buffer

Sometimes, at program start, there is still some garbage in the input buffer.

This can be removed by this funtion:

```
void clearSerialInput() {
  uint32_t m = micros();
  do {
  if (Serial.read() >= 0) {
    m = micros();
    }
  } while (micros() - m < 10000);
}
```

(found on the net, I don‘t remember where)