EduCake 的 Serial 通讯使用

一、 Serial 通讯介绍

前面介绍了各种讯号输出入控制功能，已经可以利用多样的传感器输入、马达、LED 等零件搭配出不同的花样；这些都是在控制板上就可以达成的功能，那如果控制板要跟其他装置，例如另一块控制板、智能型手机、计算机等等进行互动呢？读者应该已经发现，之前的程序练习范例中，已经利用 IDE 上的「Serial Monitor」做过控制板往计算机端回传数值或讯息进行除错了，这就是控制板跟外界的主要通讯功能，也是控制板使用上的一大重点；本文将详细介绍 86Duino EduCake 在「串行通讯端口」方面的功能，并练习相关程序的写法。

谈到「串行通讯端口」(Serial Port，也称为串行通讯端口)，就得跟「并列通讯端口」(Parallel Port)一起做个比较，如图 1 的示意图，串行通讯的概念一般是经由单一条通讯线路，以「分时」的方式一位一位依序传送数据，而并列通讯则以多条线路「同时」传送多个位数据；优点当然是并列通讯带宽较大，而串行通讯较省线路空间，但同频率下需较长的时间才能跟并列通讯传送一样多的数据，另外并列通讯在长距离传输时也会有讯号同步的问题；目前计算机硬件相对刚起步的年代已经可以做到高频率的情况下，串行通讯已经可以达到相当高的传输速度，因此计算机周边通讯接头已经淘汰并列埠很长一段时间了，例如常见的 USB、SATA、RS232、RS422、RS485、I^2C、IEEE 1394、PCI-E、Thunderbolt 等等都是采用串行通信的方式，接头与插槽也较不占空间。或许有读者曾看过早期打印机的 printer port(或 parallel port)的 DB25 接头，以现在消费性产品讲究轻薄短小的标准来看，那可是相当占空间呢。

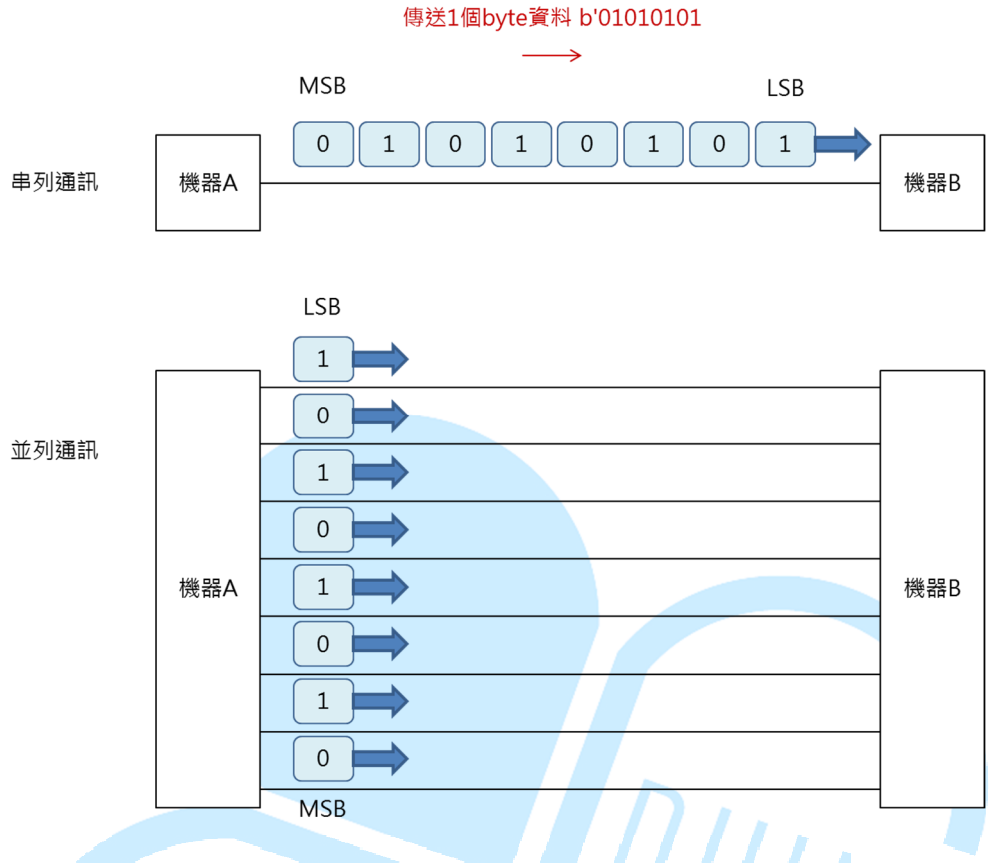


图 1. 串行埠与并列埠传输概念比较图

串行式通信也不只有单一条线，只是线路的确相对并列通信要少得多，图 1 部分仅是概念示意，实际上串行通信仍需配合一些功能线才能正常运作（如同步讯号之类）。举个例来说，之后章节会提到的 I^2C 通信方式，就有一条频率线（SCL）与一条资料线（SDA），需靠 SCL 的辅助才能取得正确的 SDA 资料时序；而 RS485 则是数据线（D+）与数据线反向讯号（D-）各一条，用以减少噪声干扰，各种通信规格各有不同的特色。像 RS232、RS422、RS485 这些通信方式各有不同的电器及机械定义标准，但都属于 UART（通用异步收发传输器，Universal Asynchronous Receiver/Transmitter），UART 仅是通称而已。以 86Duino EduCake 的通讯接口也是属于 UART 的通讯方式，有 TX、RX 线路各一条（也有些地

方会标示成 TxD/RxD，是一样的东西），TX 负责从控制板送出讯号，而 RX 则负责接收外界传来的讯号。这种 TX/RX 各自有专用线路且可以同时收送的，又称为「全双工」式通讯，表示此线路可允许连接两端的机器同时收送数据；「半双工」则是线路上同时只能有传送或接收，不能同时进行，像 I^2C、RS485 通讯就是半双工的机制；而 86Duino 的 UART、计算机上常见的 RS232 则是全双工的机制。「全双工」与「半双工」可参考图 2 的说明。

TX/RX 这两个脚位在这边使用的是一种称为 TTL 5V 的电压位准，表示在通信在线的电压 LOW/HIGH 在 0~5V 之间；但像 RS232 的逻辑 HIGH(1)则是定义成-3~-15V，逻辑 LOW(0)则是 3~15V。所以当两个这类型装置以 UART 互相连接时，需注意通讯脚位的电压范围是否兼容，且机器 1 的 TX 需连接机器 2 的 RX，机器 1 的 RX 需连接机器 2 的 TX。以 TTL 的 UART 要跟 RS232 通讯为例，通常得用上 MAX232 之类的芯片进行电压准位转换才行。

通常具备 UART 的机器，在 TX/RX 端会有各自的暂存缓冲区，且为 FIFO（First in first out，先进入的数据先取出）的机制，缓冲区用在暂存即将送出以及刚接收到的数据，以免 CPU 无法及时处理，如图 2 所示。此机制之后也会在范例程序中使用，读者可先了解有此机制的存在；不同机器上缓冲区的大小可能也不同，甚至可能没有缓冲区，得看各种机器的规格而定。

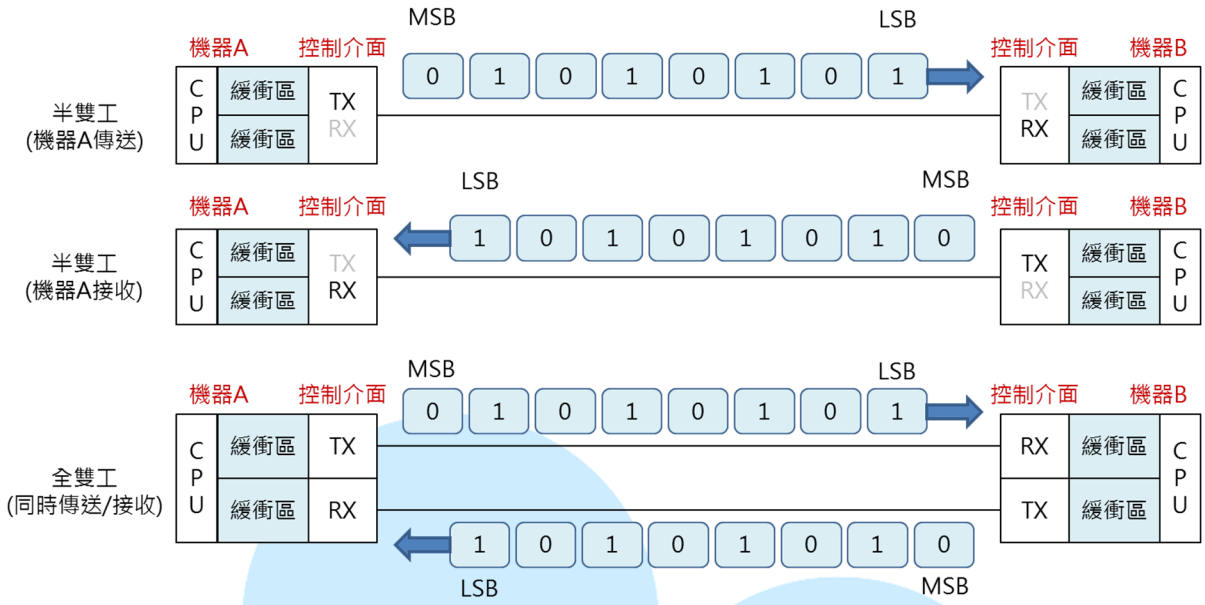


图 2. UART 全双工/半双工机器互连及缓冲区示意图

UART 还有另一个特点是通讯协议的处理，传送/接收双方机器都必须遵守共通的通讯协议以及参数设定，才能正确无误地与对方沟通。UART 通讯协议如图 3 所示，一开始会先送出起始位，接着是数据位，之后是同位检查与结束位，每传输一个数据单位（Data Frame）都须遵守这个规则。起始位用在让接收方机器判断何时为 Frame 的起始点，UART 的「异步 Asynchronous」就是这个意思；其他名词则在下方作介绍。

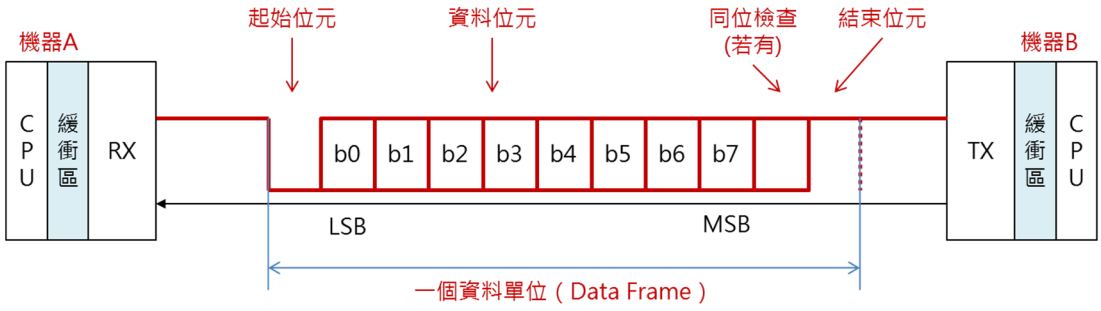


图 3. UART 通讯协议结构

以一般计算机为例，若读者已经将控制板接上计算机的 USB 插槽，便能从我的计算机→设备管理器看到如图 4 的画面，由于控制板上已有芯片将 TTL UART 转换成 USB 讯号，因此计算机便能将控制板辨识为一个虚拟 COM Port（计算机上通用串行端口的一般称呼）并对他传输/接收数据。鼠标在装置上右键单击→内容，便可看到串行埠的参数设定。

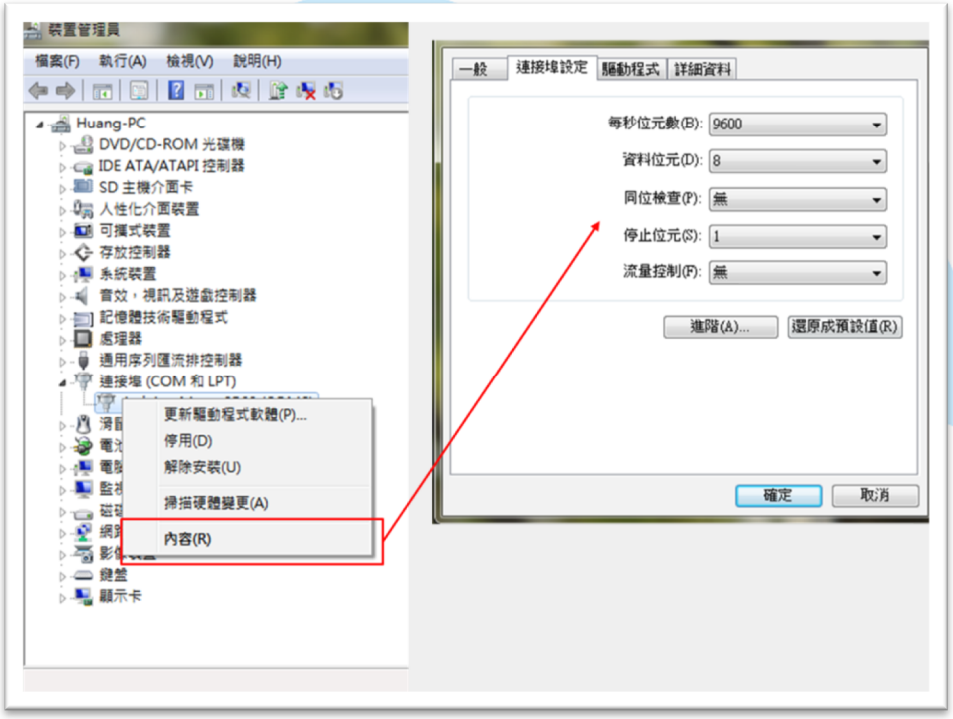


图 4. 设备管理器 COM Port 设定画面

各项参数简介如下：

- **每秒位数(也称为 Baudrate、鲍率)**

  设定串行传输的「传输速度」，单位为 bit/second（bps）。由于 UART 本身没有数据同步讯号，仅观察起始位作为传输开始的信号，因此通讯两边机器必须设定为相同的数值，才能用相同的速度进行传输及收讯采样，否则接收方可能收到错误讯息或根本收不

到东西；例如 Serial Monitor 的设定跟控制板不一致时，传回讯息可能看到乱码这种情况。数字越大表示每秒传输的数据量越多，但使用时须注意机器本身的可设定上限，运作速度（频率）较低的机器，可能只支持 4800 或 9600 的速度。部分较新的机器可能支持自动侦测 Baudrate（Automatic baud rate detection，简称 ABR、autobaud）的功能，以机器所载规格为准。

- **数据位**

  设定一个传输单位里面含有几个位的数据，通常以传输 1 个 byte（=8bits）当作基本单位，所以这里设定为 8 即可；传输时数据位以先 LSB（最低位）后 MSB（最高位）的方式进行传输。

- **同位检查(也称为奇偶校验、Parity check)**

  此字段用在检查数据本身的正确性，有时候当环境本身干扰多、通信线路因老化或接触不良导致传输质量不佳时，可能 A 机器传了数据位 b'01010101，结果 B 机器收到了 b'01000101，数据便出现了错误。同位检查可设定为：

  1. 无（No Parity）：不进行检查，讯号中便没有这个位。
  2. 奇同位（Odd Parity）：当数据位中有偶数个 1，则奇偶校验位为 1，反之为 0；数据位与奇偶校验位全部带有奇数个 1。

3. 偶同位（Even Parity）：当数据位中有奇数个 1，则奇偶校验位为 1，反之为 0；数据位与奇偶校验位全部带有偶数个 1。

4. 标记（Mark）：少用的设定方式，不管数据位状况，都把奇偶校验位设定为 1。

5. 空格（Space）：少用的设定方式，不管数据位状况，都把奇偶校验位设定为 0。

- **结束位**

  标示一段数据传输的结尾，若传输过程中双方机器出现些微的不同步状况，也可让在这段时间内进行时钟的同步校正。

- **流量控制**

  当数据传输时，接收方由于处理速度不足、缓冲区域太小等等原因，导致来不及接收数据，便可以启动这个功能。当接收端收完资料便会告知传送端自己已经收完资料，让传送端可以继续下个资料的传送。

一般预设参数通常是：数据位：8、同位检查：无、停止位：1、流量控制：无，至于每秒位数就得要用户自己设定了。使用 UART 通讯时，需特别注意接线是否正确，以及双边参数是否都设定正确，通讯有问题时可先从这边着手解决。上述通信机制看起来是不是有点复杂呢？没关系，幸好 86Duino EduCake 的 Code100 IDE 里面已经有包装好的函式库可以方便直接使用，也就是接下来范例程序练习会使用到的「Serial Library」，以下程序范例练习，参数都设定为：数据位 8、无同位检查、停止位 1、无流量控制、鲍率 115200 bps。接下来就让我们以范例进行练习吧~

二、 第一个程序 – 练习 Serial.write( )

前面提到的数据位，虽然在电路上都是 HIGH(1)/LOW(0)的讯号，但程序在传送跟接收数据时其实可以用不同的解读（或解碼）方法，分为「数字」跟「字符」两种。数字的解读方式便是直接以二进制、十进制、十六进制等方法直接解读；「字符」的编码/译码方法则是依照「ASCII 表」的定义，十进制 0~127 各代表不同的「符号」，ASCII 表的数据在网络上很容易找到，读者可以直接搜寻相关资料阅读。例如数据位传送 b'01000001，以「数字」方式解读就是十进制的数字 65，以「字符」方式解读则是符号「A」。

第一个范例程序，我们先做 86Dino EduCake 与计算机双方进行单一 byte，且以「数字」作编码/译码的练习。以 86Duino EduCake 的程序写作为例，读者请打开 86Duino Coding 100 IDE，输入以下程序代码：

```
void setup()
{
  Serial.begin(115200);// 設定 Serial Port 的 Baudrate
}

void loop()
{
  if(Serial.available())
  {
    int value = Serial.read();// 以數字方式解讀收到的資料
    delay(100);
    Serial.write(value + 1);// 以數字方式回傳收到的資料
  }// end if(Serial.available()

}
```

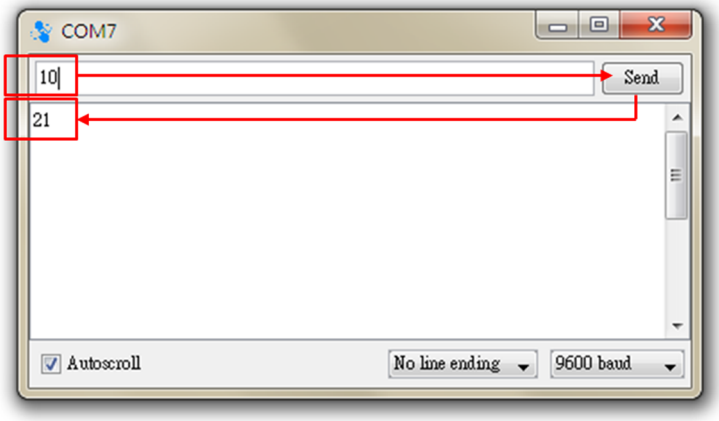
此范例程序功用为：当用户传输一个 byte 的数据给 86Duino EduCake 时，EduCake 会将收到的数值以数值方式解释，加上 1 之后回传。

一开始在 setup( )里，先以 Serial.begin(115200)进行 Serial Port 的 Baudrate 设定；接着 loop( )里以 Serial.available( )检查是否 Serial port 有接收到数值，若有接收到数据时，Serial.available( )会回传目前缓冲区内暂存的资料 byte 数量。

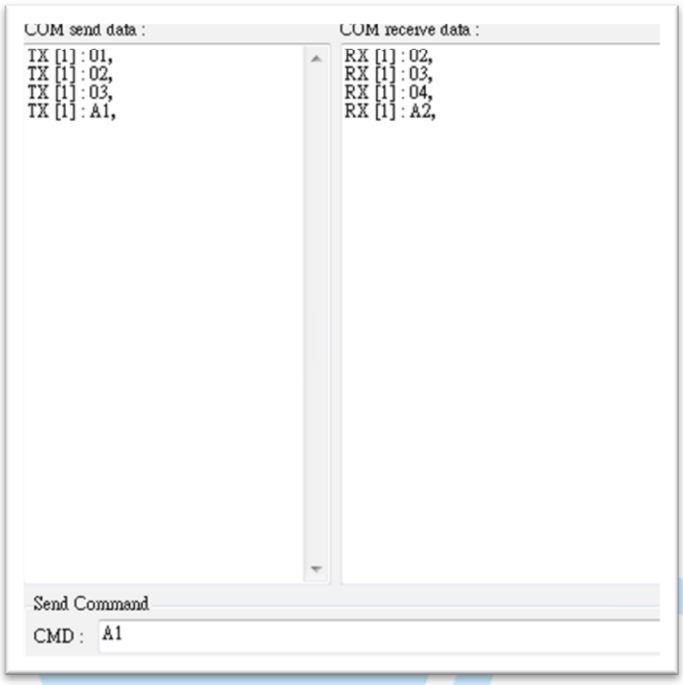
由于以「数值」方式做译码，因此宣告 byte value 变量进行接收（也可以使用 unsigned int、int 等进行宣告，视使用者数值传换状况而定），以 Serial.read( )做读取。读取完数据后，暂停一小段时间把收到的数值加上 1 再送出（此处 delay(100)只是用来看出延迟传送的效果，也可以不加延迟），传输资料则是使用 Serial.write( )指令。注意 Serial.read( )、

Serial.write(byte)指令都是以 1 个 byte 当作单位，传送与接收数据时须留意数据内容，以免超出 1 个 byte 可表示的数字范围（0~255）。

此范例暂时不使用 Serial Monotor，因为 Serial Monitor 默认会使用「ASCII 字符」方式进行传送与接收译码，当用户从上方输入框输入任何数字或文字，Serial Monitor 会将输入的数据分开为各别字元传送出去，例如当输入「数字 10」，实际上 Serial Monitor 会送出代表'1'、'0'字符的两个数据，在 ASCII 表中分别为十进制 49 跟 48，因此会看到 EduCake 分别回传了 49+1=50、48+1=49 两个资料回来，被 Serial Monitor 解读为'2'、'1'，跟原本预期的数字 10+1=11 大不相同，如下图所示：



这边以一个计算机端 UI 来作范例如下：此接口纯粹以数值方式做传送与接收（以 16 进位方式呈现），当传送 1 时收到 2，传送 2 收到 3...传送 A1 收到 A2，依此类推。



三、 第二个程序 – 练习 Serial.print( )

接着这个范例程序我们使用字符的方式做练习，请在 Coding 100 IDE 输入以下程序：

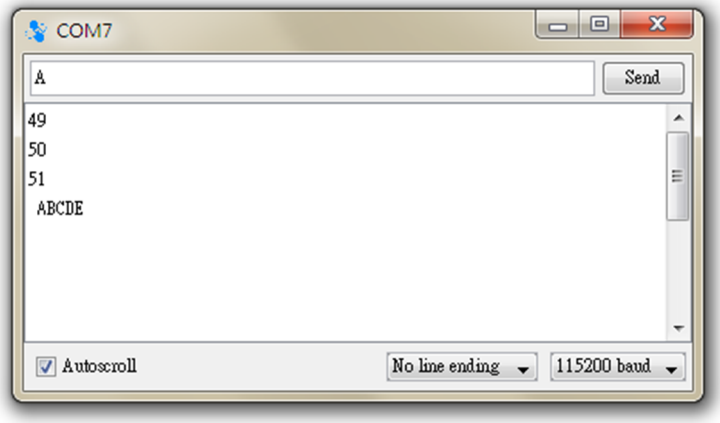
```
void setup()
{
  Serial.begin(115200);// 設定 Serial Port 的 Baudrate
}

void loop()
{
if(Serial.available())
  {
    byte value = Serial.read();// 以數字方式解讀收到的資料
    delay(100);

    if(value == 'A')
    {
```

```
            Serial.write(32);// space
            Serial.write(65);// A
            Serial.write(66);// B
            Serial.write(67);// C
            Serial.write(68);// D
            Serial.write(69);// E
            Serial.write(10);// \n
        }
        else
    {
            Serial.println(value);//  以數字方式回傳收到的資料
        }
      }// end if(Serial.available()
    }
```

接着上传程序至 EduCake 后，开启 Serial Monitor，此范例便可使用 Serial Monitor 做通讯。此程序功能为，当用户由输入框输入字符'A'，则 EduCake 回传十进制 32、65~69、10，而 Serial Monitorg 收到数据后会以字符方式解释这几个数据，这几个数据在 ASCII 表中代表了符号'空白'、'A'、'B'、'C'、'D'、'E'、'换行'显示在窗口中，注意 ASCII 表格中十进制 0~31、127 是「控制字符」，表示这些符号是无法显示的，而是执行某些动作，例如范例中 Serial.write(10)便可让 Serial Monitorg 窗口光标换到新的一行（读者也许会在其他程序语言书中看到'\n'，就是此处的换行控制字符），表中 32~126 则是「可以显示的字符」，像是'0'~'1'、'a'~'z'、'A'~'Z'等符号。如果用户不是传送字符'A'，则传送其他数据都会让 EduCake 执行 Serial.println(value)指令，此指令会将「value 的十进制值」以字符方式回传给 Serial Monitor，且换到新的一行（Serial.print( )功能相同只是不换行）；

Serial.print( )、Serial.println( )也是之前章节用来除错或印出变量内容的语法。当使用者由 Serial Monitor 送出 1，实际上 EduCake 执行 byte value = Serial.read( )后收到的是 ASCII 表中代表字符 1 的十进制数值「49」，而 Serial.println(49)则会传出代表十进制值 49「字符」的两个 byte 十进制「52」、「57」给 Serial Monitor，因此窗口上会显示 49 两个字符；这个范例程序也可以用来观察各个「可显示字符」实际在 ASCII 表中对应的十进制值是多少。

实际测试如下图，传送'1'、'2'、'3'后分别收到字符对应的十进制值 49、50、51，传送'A'则收到空白 ABCDE 并换到下一行。



由上面两个范例应该不难看出 Serial.write(value)跟 Serial.print(value)两个指令的差别，Serial.write( )是传送出 value 这个 byte，接收端收到的一样是 value 这个数据，只是接收端选择用数字或符号去解释这个数据而已；而 Serial.print( )则是把 value 所代表的数值符号以 ASCII 对应码传送出去，接收端也要以 ASCII 字符方式解释才能看出收到数据的意义。

Serial.print( )也有不同的参数可用，例如 Serial.print(value, DEC)是以 10 进位显示 value 的数值字符（默认就是十进制，因此第二个参数栏可以

不输入 DEC）、Serial.print(value, BIN)是二进制显示、Serial.print(value, OCT)是八进制显示、Serial.print(value, HEX)是以十六进制显示等等，读者可以尝试看看不同参数的效果。Serial.println( )不加任何参数则是单纯换行，对于以字符解释数据时与 Serial.write(10)功能相同

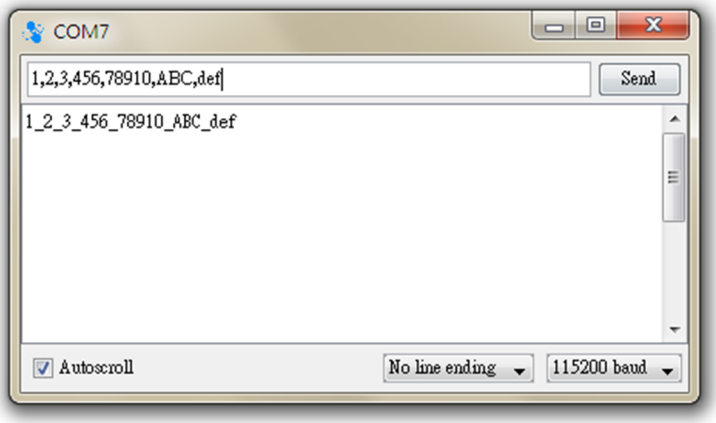
四、 第三个程序 – 传送/接收多个 byte(数字)

练习完传送单一个 byte 数据后，接着来练习传送/接收多个 byte 的状况，也是多数专题应用时会碰到的状况。请在 Coding 100 IDE 输入下列程序代码：

```
void setup()
{
  Serial.begin(115200);// 設定 Serial Port 的 Baudrate
}

void loop()
{
  int byteCount = Serial.available();// 取得緩衝區內的資料 byte 數量
  if(byteCount > 0)
  {
    byte RX_buff[byteCount];// 宣告陣列，用來複製緩衝區內的資料
    for(int i = 0; i < byteCount; i++)
    {
      RX_buff[i] = Serial.read();// 將收到的所有資料存到陣列裡
    }

    for(int i = 0; i < byteCount; i++)
```

```
        {
            if(RX_buff[i] != 44)//  當暫存區的數值不是 44(字元',')
            {
               Serial.write(RX_buff[i]);
            }
            else
            {
               Serial.write(95);// '_'
            }
          }

          Serial.println();//  換行

        }// end if(byteCount > 0)

    }
```

接着上传程序至 EduCake 后，开启 Serial Monitor。此程序功能为，当用户在输入框输入一连串字符，其中可包含逗号','，则 EduCake 会将符号','替换为符号'_'并依原本顺序传回给 Serial Monitor，如下图。



loop( )中先使用 int byteCount = Serial.available( )检查目前 Serial 缓冲区内有多少资料，如果 byteCount 大于 0 表示缓冲区内有数据，则宣告一个用来接收 Serial Port 暂存区数据的 byte 数组名为 RX_buff，数组长度为 byteCount，然后使用 for 循环配合 Serial.read( )将缓冲区数值复制进数组里，其实此处的 for 循环也可以用 Serial.readBytes(RX_buff, byteCount)指令取代，可以让程序代码变得更简洁。前面提到 Serial Port 有 FIFO（先入先出）的暂存缓冲区，运作机制是每执行一次 Serial.read( )这个指令，缓冲区内就会少一个 byte 的数据。接着使用另一个 for 循环并检查 RX_buff 内容，当内容不是十进制值 44（表示原本收到字符','），则回传原本内容，反之以 Serial.write(95);// '_'替换为底线字符。

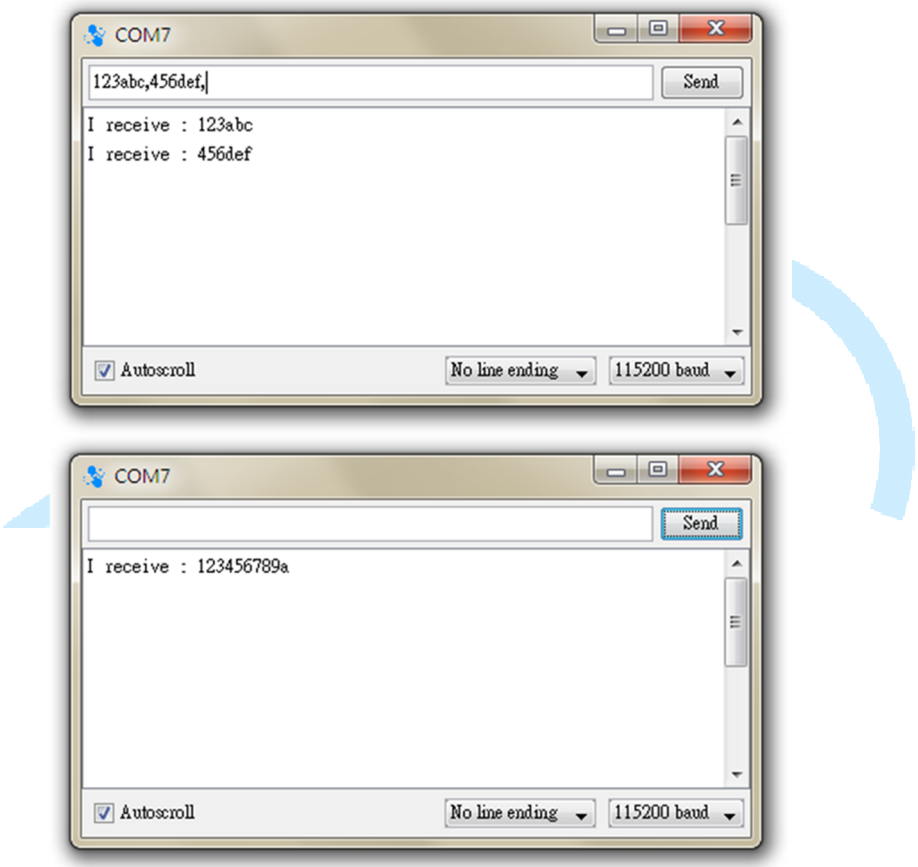
此程序需特别注意一个地方，由于 loop( )内程序代码不多运行时间相当快，当 loop( )没有加上适当的延迟时间时，若用户以较缓慢的速度传送了 10 个 byte，实际 byteCount = Serial.available( )不一定会得到 10，可能好几个 loop( )才会收完总共 10 个 byte，但不影响目前这个范例的执行效果。

五、 第四个程序 – 传送/接收多个 byte(字符串)

86Duino EduCake 的 Serial Port 还有一些其他函式可以用，适合用于双方以字符串互相传输的状况，请读者输入以下程序：

```
void setup()
{
  Serial.begin(115200);// 設定 Serial Port 的 Baudrate
  Serial.setTimeout(2000);// 設定 Serial.readBytesUntil();最長等待時間，單位 ms
}

void loop()
{
  char RX_buff[10];// 宣告陣列，用來複製緩衝區內的資料
  int byteCount = Serial.readBytesUntil(',', RX_buff, 10);// 讀到','或讀了 10 個才結束等待
  //Serial.print("byteCount : ");
  //Serial.println(byteCount);

  if(byteCount > 0)
  {
    Serial.print("I receive : ");
    for(int i = 0; i < byteCount; i++)
    {
      Serial.write(RX_buff[i]);
    }
    Serial.println();// 換行
  }// end if(byteCount > 0)

  delay(100);// 等回傳完
  Serial.flush();// 清空暫存區域
}
```

上传程序至 EduCake 后，开启 Serial Monitor。此程序功能为，当用户输入一连串字符，其中包含字符','时，EduCake 便会读取到此字符前为止，并回传「I receive：读到的字符串」；或者如果没有读到','但先接收到了 10 个字符，也会中止读取并回传，执行效果如下图。



由于程序代码里使用了 Serial.readBytesUntil(',', RX_buff, 10)指令，此行程序会读取 Serial Port 至读取到字符','，或读取了 10 个 byte 为止，如果没有设定适当的截止时间，当外部装置没有送出字符','，或送出至少 10 个 byte 的数据，那么程序可能就卡在这一行再也不会往下执行了。因此一开始在 setup( )里先使用 Serial.setTimeout(2000)用来设定 Serial.readBytesUntil( )最长等待时间，单位是 ms。Loop( )里先宣告 char 型态的矩阵 RX_buff，长度 10（长度大于 readBytesUntil 的设定即可）；

int byteCount = Serial.readBytesUntil(',', RX_buff, 10)可以取得实际上读到了几个 byte 的数据，如果 byteCount 大于 0 才做回传。

Serial.print("I receive：")指令可以把一连串字符以 ASCII 编码传送出去，注意程序中字符串需用双引号头尾包夹供 IDE 辨识为字符串数据，实际传送的数据不会有双引号在内。或者读者也可以把这串数据拆开成一个个字符传送，字符则是用单引号包夹：

```
Serial.print('I');
Serial.print(' ');
Serial.print('r');
Serial.print('e');
Serial.print('c');
Serial.print('e');
Serial.print('i');
Serial.print('v');
Serial.print('e');
Serial.print(' ');
Serial.print(':');
Serial.print(' ');
```

也是一样的效果，但就麻烦很多了对吧?

程序代码最后面使用了 Serial.flush( )指令，由于用户可能一次传送超过 10 个 byte，为了下一次不会读到剩下的数据，因此使用这个指令来清空缓冲区，须注意传送与接收缓冲区都会被清空，所以在执行这一行之前会先加上一点延迟时间，让前面的 Serial.write( )、Serial.print( )可以传完数据。若没有加上些微延迟时间，会发现应该回传的数据没有传完就中断了，读者不仿试试看。

六、 第五个程序

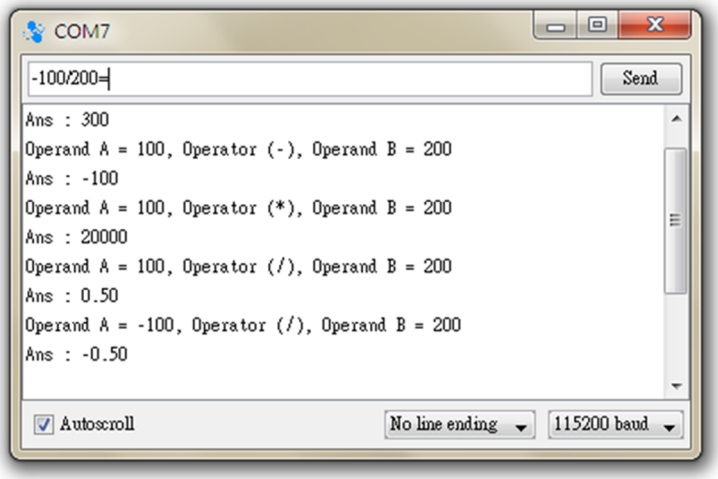
最后一个范例程序使用上面的几个函式配合几个新函式做点有趣的功能，让 86Duino EduCake 变成一个以串行通讯指令操作的计算器，读者请输入以下程序代码：

```
int Op_A = 0;// 運算元 A
int Op_B = 0;// 運算元 B

void setup()
{
  Serial.begin(115200);// 設定 Serial Port 的 Baudrate
  Serial.println("Please enter format : Operand A +(or -) Operand B =");// 傳送指示字串
  Serial.setTimeout(1000);// 設定 Serial.parseInt ();最長等待時間，單位 ms
}

void loop()
{
  // 讀取運算元 A
  Op_A = Serial.parseInt();
  Serial.print("Operand A = "); Serial.print(Op_A);

  // 讀取運算子
  char opr = Serial.read();
  Serial.print(", Operator ("); Serial.write(opr); Serial.print(")");

  // 讀取運算元 B
  Op_B = Serial.parseInt();
  Serial.print(", Operand B = "); Serial.println(Op_B);
```

```
        if(Serial.read() == '=')
        {
          Serial.print("Ans : ");
          switch(opr)
          {
            case '+':
              Serial.println(Op_A + Op_B);
              break;

            case '-':
              Serial.println(Op_A - Op_B);
              break;

            case '*':
              Serial.println(Op_A * Op_B);
              break;

            case '/':
              Serial.println((float)(Op_A) / (float)(Op_B));
              break;

            default:
              Serial.println("illiegal operator !!!");
              break;
          }// end switch
        }// end if(Serial.read() == '=')
    }
```

上传程序至 EduCake 后，一样开启 Serial Monitor。此程序功能为，当用户从 Serial Monitor 输入此格式：[整数 A][+-*/运算符号][整数 B][=]

（其中不含空格符号）然后按下传送，则 EduCake 会将收到指令及运算结果回传，整数可以是正或负数；如下图所示：



程序代码一开始先宣告两个 int 型态变量 Op_A 跟 Op_B，用来储存操作数的数值。

setup( )部分与范例四相同，loop( )流程一开始读取操作数 A，这里使用了 Serial.parseInt( )这个指令，此指令会以「字符解释」方式依序检查 Serial 缓冲区的数据，直到找到不是数字的字符为止（所以跟 Serial.readBytesUntil 一样要设定 Serial.setTimeout），然后将找到的一连串「数字字符」转换成 int 型态；例如收到了字符'1'、'2'、'3'、'+'、'1'，则会转换出十进制整数「123」，缓冲区剩下'+'、'1'，注意如果是收到'-'、'1'、'2'、'3'、'+'，则 Serial.parseInt( )指令会把第一个'-'字符算在一起，变成十进制整数「-123」。

中间使用 char opr = Serial.read( )读取运算符的符号，接着一样用 Op_B = Serial.parseInt( )读取操作数 B 的整数值。最后剩下一个符号，使用 if(Serial.read() == '=')检查是否是'='符号，如果是等于符号就可以开始

做数学运算。这个范例程序支持「加减乘除」四则运算，为了程序阅读方便这里使用 switch-case 语法做运算符号的判断，如果前面收到的运算符号是'+'，则把操作数 A、B 相加后回传，其他算法依此类推；运算符'/'的部分计算处理稍有不同，是因为整数相除可能出现小数点，所以计算前先把操作数 A、B 转成 float 型态再做除法，不然整数直接相除后小数点可是会消失喔。Switch-case 注意要写 default，以免使用者输入了不支持的运算符号。

读者也可以试着把操作数 A、B 用 float 型态宣告，然后把 Serial.parseInt( )改成 Serial.parseFloat( )，这样程序就可以支持小数点运算啰。

http://arduino.cc/en/Reference/Serial

http://arduino.cc/en/Serial/ParseFloat

http://arduino.cc/en/Serial/ParseInt

http://arduino.cc/en/Serial/ReadBytes

七、 第六个程序 GPS 的实作

本章节最后，我们来实作一个 GPS 的简单应用，看看现在已经普及到生活中的 GPS 到底是如何结合 EduCake 来发挥他的功能。

全球定位系统（Global Positioning System，简称 GPS），又称全球卫星定位系统，这是美国国防部研制和维护的中距离圆型轨道卫星导航系统。涵盖地球表面绝大部分地区（98%），可以提供准确的定位、测速和高精度的时间，搭载非常精准的原子钟对卫星时间作校准，至少远超过我们平时使用的手表和时钟，也可以顺便用来帮 EduCake 作定时器和取得时钟时间的功能。详细的 GPS 原理就跳过了，有兴趣的读者可以 GOOGLE 或是到维基百科查询 GPS 或是全球定位系统，就可以找到很多相关的原理和知识，这里直接看来实作。

首先，想办法取得一个 GPS 模块，一般拍卖都有很多，现在市面上的 GPS 模块的输出讯号多数都 RS232/TTL 规格，也有 I2C 规格或是 SPI，但不管哪个规格，EduCake 都能处理，笔者选用最简单的 TTL 规格版本 (NEO-6M)，如图 1 所示

图 1. TTL 讯号输出的 GPS 模块

这种 GPS 模块一个约数百台币就可买到，接线很单纯，由右至左分别是：NC(悬空不接)、VCC(给 3.6~5.2V，这可直接接 EduCake 的 5V)、RX(UART 接收命令用，像是设定传输速度之类的指令)、TX(UART 数据输出)、GND、1PPS(定频率波，周期一秒)，输出数据的速度是 1~5Hz。

GPS 模块有几个状态需要了解一下，第一个是冷起动，这是指从完全断电的情况下通电，GPS 模块必须初始化、抓到卫星、重新计算目前的相关信息等等，这些动作是需要一点时间的，通常至少是二三十秒的时间，实际的行车导航会远超过这个时间，因为还要重新产生附近的图资、路径规画等信息，会有数十秒~一两分钟以上，且这是在有抓到 3 颗以上卫星的情况下才行(这跟 GPS 运作原理有关)；第二个是温启动，待机情形下继续工作，约需要 1~数秒；第三种情况是重新捕捉卫星，这最容易发生在城市大楼密集处、桥下、进出地下停车场、进出隧道时，因为卫星讯号在这些地方会微弱或是根本收不到，但再次收到卫星信息时，GPS 会和 buffer 里面存下的之前的信息来比对，迅速修正目前位置，通常这个动作约一秒内。

接线方式如图 2，主要只接 5V、GND、TX 就好，其他的不需要接，若需要取得精确的一秒钟，那就多接 1PPS 那条线就好，可以获得周期一秒，HIGH 时间 100ms 的固定脉波，板子上的标示很清楚，应不会有接错的疑虑才是

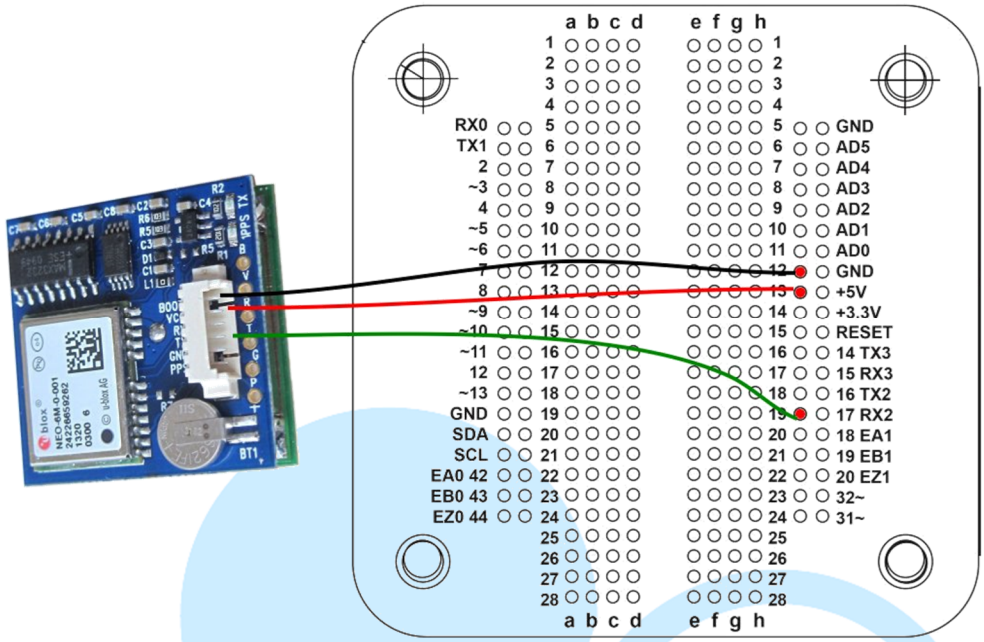


图 2. GPS 模块接线方式

接好线以后，Educake 里面放入以下程序代码

```
void setup()
{
  Serial.begin(9600);
  Serial2.begin(9600); //GPS 接在這裡
  Serial.println("Begin");
  delay(2000); //  這裡需要頓一下，GPS 開機初始化需要時間
}

void loop(){
    char ch;
    if ( Serial2.available()) //  有收到東西時
    {
     Serial.println("Get:");
     while (Serial2.available()) //就開始一直收，直到沒東西
     {
      ch=Serial2.read();
      Serial.print(ch); //並且一直印到畫面上
     }
    }
}
```

这个程序只是用来把 GPS 接收到的东西原封不动印出来 Serial Monitor 看，有助于我们先了解到底会出现什么，结果会看到一堆类似底下字符串的东西，这种东西就是 GPS 模块抓到的卫星定位信息，有很多种，我们需要用程序一行一行来解析这些字符串以便取得想要的信息，底下是 GPGGA 类型的封包

$GPGGA,073849.00,2301.37777,N,12012.94773,E,1,09,0.098,41.2,M,-2.5,M, ,*42

这行字里面的每一个信息都用小写的逗号"，"隔开，分别代表以下意思

$GPGGA	GGA 规范的讯息开头
073849.00	代表时时分分秒秒.秒秒，所以这是说国际标准时间早上 7 点 38 分 49.00 秒的 UTC 时间，台湾来说，必须加上八小时，所以这代表台湾时间 15 点 38 分 49 秒才对
N	代表北半球，南半球就会是 S
2301.37777	代表纬度 度度分分.分分分分分
E	代表东半球，西半球就是 W
12012.94773	代表经度 度度度分分.分分分分分
1	定位代号，0 代表无效、1 代表已定位、2 代表偏差修正、3 是军用格式(一般这是收不到的，收到也解不出来)
09	抓到 9 颗卫星，一般低于 3~5 颗就根本没准确度可言了
0.098	水平精确度，实测觉得这应该不可能，哪会这么准，这数值笔者测试约会在 0.08~15 左右跳来跳去，当作参考就好，主要是卫星数量，数量越少就越不准；附近是否空旷造成的影响最多，若有很多高楼大厦，会因为卫星讯号的反射折射等因素，就算抓到很多颗也会有比较大误差
41.2	离海平面的高度
M	单位是公尺
-2.5	地表平均高
M	单位公尺
*42	checksum 检查码，确认这则 GPS 讯息是否有误

基本上有收到这个已经所有想要的信息都有了，但 GPS 还有很多别的讯息是可以使用的

以下的这两个就是

$GPGLL, 2301.37777,N, 12012.94773,E, 073849.00,A,A*62

这是 GLL 封包，里面主要只有经纬度和 UTC 时间的信息，因为解释方式都和上面一样就不重复，请自行查询上面的表格。

$GPRMC,073141.00,A,2301.37777,N,12012.94773,E,0.016,0,190214,,,A*77

这是 RMC 封包，跟上面的也是很像，但其中比较不一样是第三个参数 A，代表这讯息是可用的，若为 V 就代表这个讯息不正确，可能是卫星数太少或是讯号微弱不足以算出位置；经度信息后面那个 0.016 代表速度，单位是节(1 节=1 海浬/hr=1.852 公里/hr)；"0" 代表对地方向；190214 是日期，代表 2014 年 2 月 19 号；再来的参数是磁极的偏移量和方向，不过这里都没信息；A*77 是 checksum 检查码。

GPS 其实很有很多种别的封包，不过因为有些信息牵涉很复杂的数学验证，不适合这里讨论，有兴趣的读者可以参考 http://goo.gl/QSfSsS，另外，经纬度的定义参考 http://goo.gl/rDvgzl ，

.E7.B6.93.E7.B7.AF.E5.BA.A6.E8.A1.A8.E7.A4.BA.E5.8F.8A.E8.BD.89.E6.8F.9B， 1 度是 60 分，1 分是 60 秒，截取到的经纬度必须经过这个换算才能变成真正 GOOGLE MAP 能接受的坐标。组合以上封包能获得的信息，其实就能截取到目前所在位置的: 经度、纬度、速度、高度、日期、原子钟等级的精确时间、方向等等的信息，底下就来写一个真正的范例，把这些信息从 GPS 传回的封包里面抓出来吧。

另外为了能比较直觉的获得信息，也使用 16*2 LCD 来协助显示到液晶模块上面，这样即使不接计算机，也可直接使用。

```
#include <LiquidCrystal.h>

// rs 接 digital 3
// rw 接 digital 4
// enable 接 digital 5
// d4, d5, d6, d7 接 digital 6、7、8、9
LiquidCrystal lcd(3,4,5, 6,7,8,9);
int LightBri = 32;   //對比控制接 digital 32

String data="";
int mark = 0;
boolean Start_Flag=false;
boolean valid=false;
String GGAUTCtime,GGAlatitude,GGAlongitude;
String GPStatus,SatelliteNum,HDOPfactor,Height;
String PositionValid,RMCUTCtime,RMClatitude;
String RMClongitude,Speed,Direction,Date,Declination,Mode;

void setup()
{
  pinMode(LightBri, OUTPUT);
  analogWrite(LightBri, 40);   // 設定對比，越低螢幕顯示的字會越黑
  lcd.begin(16,2);
  lcd.clear();
  lcd.setCursor(0,0);   //設定游標在第一列開始印
  lcd.print("DMP Demo");

  Serial.begin(9600);
  Serial2.begin(9600); // GPS   連接在這裡
```

```
        Serial.println("Begin");
      // 這裡最好先停頓 1~10 秒,主要是 GPS 開機後需要一點時間作初始化和抓取衛星的動作
        // 這個動作需要多少時間得看附近的地形是否容易抓到夠多的衛星而定
        // 若數量不夠，GPS 會一直沒辦法輸出正確訊息
        delay(2000);
      }

    void loop()
    {
      while (Serial2.available()> 0)
      {
        if(Start_Flag){   //開始旗標是 true 就可以開始抓取相關欄位的資料
            data=readGPS(); // 第一，先把封包種類資訊抓出來，可能是 GPGGA、GPGSV 等等
            Serial.println(data);
            if(data.equals("GPGGA")){
              // 抓到 GGA 的封包，這裡就是參照前面談過的表格內容
              // 每隔一個逗號都是一項資訊,呼叫 readGPS 函數按順序把欄位內容讀取出來
              GGAUTCtime=readGPS();
              GGAlatitude=readGPS();
              GGAlatitude+=readGPS(); // N 或 S，直接加在緯度後面
              GGAlongitude=readGPS();
              GGAlongitude+=readGPS(); // W 或 E，直接家在經度後面
              GPStatus=readGPS(); // 定位狀態，0 無效，其他數字都是有效定位
              SatelliteNum=readGPS(); // 抓到的衛星數量
              HDOPfactor=readGPS(); // 精確度
              Height=readGPS(); // 距離海平面高度
```

```
                Start_Flag =false;

                if (GPStatus>0) //  必須是有效定位才可以設定
valid=true，以利後面判斷
                    valid=true;
                else
                    valid=false;
                data="";
            }
            else if(data.equals("GPGSA") ){   // GSA 封包跳過不處理
                Start_Flag =false;
                data="";
            }
            else if(data.equals("GPGSV")){   // GSV 封包跳過不處理
                Start_Flag =false;
                data="";
            }
            else if(data.equals("GPRMC")){
                //  抓到 RMC 的封包，這裡就是參照前面談過的表格內容
                //  每隔一個逗號都是一項資訊，呼叫 readGPS 函數按順序
把欄位內容讀取出來
                RMCUTCtime=readGPS();
                PositionValid=readGPS(); //  讀取到 A 才是有效封包
                RMClatitude=readGPS();
                RMClatitude+=readGPS();
                RMClongitude=readGPS();
                RMClongitude+=readGPS();
                Speed=readGPS(); //速度
                Direction=readGPS(); //方向
                Date=readGPS(); //  日期
                Declination=readGPS();
                valid=true;
```

```
        Start_Flag =false;
                data="";
            }
            else if(data.equals("GPVTG")){
                Start_Flag =false;
                data="";
            }
            else{
                Start_Flag =false;
                data="";
            }
          }

          if(valid){
            if(PositionValid=="A")
            {
                Serial.println("Get a valid position");
                output(); // 印出前面抓到的正確訊息到 LCD 和 serial
port
            }
            else
                Serial.println("Not a valid position");
            valid=false;
            PositionValid="";// 重新把檢查旗標清空，才能確實判斷下
次是否有抓到正確資訊
          }
          if(Serial2.find("$")){ // 若從 GPS 來的資料裡面含有$，代表開
始抓到資訊了
            Serial.println("Get GPS string...");
            Start_Flag =true; // 就可以把開始旗標設定為 true 準備開始
處理
          }
        }
    }
```

```
    //    從整串的 GPS 傳回字串裡面讀取被逗號隔開的資料
    //    範例 $GPGGA,073849.00, 2301.37777,N,
12012.94773,E,1,09,0.098,41.2,M,-2.5,M, ,*42
    String readGPS(){
      String value="";
      int temp;
    startRead:
      if (Serial2.available() > 0)
      {
        temp=Serial2.read(); //   一個字一個字讀取出來
        if((temp==',')||(temp=='*')) //需要一直讀取，直到讀到逗號或
是*號才是一段完整資訊
        {
          if(value.length()>0)   // value 裡面的長度比 0 大代表有確實
讀到資訊，直接傳回
            return value;
          else
            return "";   // 否則就傳回空字串
        }
        else if(temp=='$')   // 讀到$代表這是一段訊息的最開頭，準
備開始截取資訊
          Start_Flag =false;
        else
          value+=char(temp); // 把每一個讀到的字都加入 value 字
串
      }

      // 這個 delay()的數值需要稍大或稍小調整，1~3 都可以
      // 等待一下，避免讀取速度過快，GPS 根本沒資訊傳回
      delay(1);
      goto startRead; // 跳回 if 開頭，不斷的讀取，直到確認能讀到
完整資訊
    }
```

```
void output() //  輸出相關訊息的程式都包裝在這裡
{
  //  底下這一大段 serial.print 純粹是用來把訊息印出來看
  Serial.print("Date:");
  Serial.println(Date);
  Serial.print("UTCtime:");
  Serial.println(GGAUTCtime);
  Serial.print("Latitude:");
  Serial.println(GGAlatitude);
  Serial.print("Longitude:");
  Serial.println(GGAlongitude);
  Serial.print("GPStatus:");
  Serial.println(GPStatus);
  Serial.print("SatelliteNum:");
  Serial.println(SatelliteNum);
  Serial.print("HDOPfactor:");
  Serial.println(HDOPfactor);
  Serial.print("Height:");
  Serial.println(Height);
  Serial.print("Speed:");
  Serial.println(Speed);
  Serial.print("Direction:");
  Serial.println(Direction);
  Serial.print("Mode:");
  Serial.println(Mode);

  //  顯示相同的訊息到 LCD 上面
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("time:");
  lcd.print(GGAUTCtime);
  lcd.setCursor(0,1);
  lcd.print("Satellite:");
```

```
        lcd.print(SatelliteNum);
        delay(600); //  因為 16*2 LCD 螢幕只有兩行，需要頓一個時間以
後清除畫面，重新顯示新資訊
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("lat:");
        lcd.print(GGAlatitude);
        lcd.setCursor(0,1);
        lcd.print("lon:");
        lcd.print(GGAlongitude);

        delay(600); //  再度停頓一段時間後清畫面
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Height:");
        lcd.print(Height);
        lcd.print("m");
        lcd.setCursor(0,1);
        lcd.print("Speed:");
        lcd.print(Speed);
        lcd.print("km");
        delay(550); //  最後還是要頓一段時間，避免立刻回主迴圈，被清
掉畫面，最後兩行資訊來不及看到
    }
```