类 C 相关语法

数据类型及变量

1. 数据类型
   void unsigned char char boolean byte int unsigned int word
   long unsigned long float double string - char array String - object array -（数组）
2. 数据类型转换: char() byte() int() word() long() float()
   word()
   把一个值转换为 word 数据类型的值，或由两个字节创建一个字符。
   word(x) word(h, l)
   参数:
   x：任何类型的值
   h：高阶（最左边）字节
   l：低序（最右边）字节
3. 修饰符: static volatile(后文有提及) const
4. 变量

常量

1. 布尔常量：true false
2. 整数常量
3. 浮点常量
4. constants 预定义的常量
5. Arduino 里的常量：HIGH，LOW 等

控制语句与扩展语法

1. 选择语句：if、if…else、switch…case
2. 循环语句：while、do…while
3. 跳转语句：break、continue、return、goto
4. 扩展语法：;、{}、//、/* */、#define、#include

运算符

1. 算数运算符：=、+、-、*、/、%
2. 比较运算符：==、!=、<、>、<=、>=
3. 布尔运算符：&&（与）、||（或）、!（非）

4. 指针运算符：* 取消引用运算符 & 引用运算符
5. 位运算符：&（与）、|（或）、^（异或）、~（非）、<<（左移）、>>（右移）
6. 复合运算符：++、--、+=、-=、*=、/=、&=、|=

函数与头文件

Arduino 自带两个函数：setup()初始化函数，只运行一次；loop()循环体函数，重复运行。函数调用与 c 语言差不多，也支持头文件，但不能在头文件里使用 arduino 里的封装的函数，头文件写宏定义和数据处理比较好。

Arduino 基本函数

数字 I/O 函数

1. pinMode(pin, mode)
   用以配置引脚为输出或输出模式，它是一个无返回值函数，函数有两个参数 pin 和 mode，pin 参数表示所要配置的引脚，mode 参数表示设置的模式—INPUT（输入）或 OUTPUT（输出）。另外还有 INPUT_PULLUP 模式，用于上拉约 20k 的电阻，10k 左右差不多了。
   注意：Arduino 板上的模拟引脚也可以当做数字引脚使用，编号为 14（对应模拟引脚 0）到 19（对应模拟引脚 5）。
   函数原型：void pinMode(uint8_t pin, uint8_t mode)
2. digitalWrite(pin, value)
   它的作用是设置引脚的输出的电压为高电平或低电平。该函数也是一个无返回值的函数，函数有两个参数 pin 和 value，pin 参数表示所要设置的引脚，value 参数表示输出的电压—HIGH（1 高电平）或 LOW（0 低电平）。
   注意：在使用 digitalWrite(pin, value)函数设置引脚之前，需要将引脚设置为 OUTPUT 模式。
   函数原型如下：void digitalWrite(uint8_t pin, uint8_t val)
3. digitalRead(pin)
   用在引脚为输入的情况下，可以获取引脚的电压情况—HIGH（1 高电平）或 LOW（0 低电平），参数 pin 表示所要获取电压值的引脚，该函数返回值为 int 型，表示引脚的电压情况。
   函数原型如下：int digitalRead(uint8_t pin)

模拟 I/O 函数

1. analogReference(type)
   作用是配置模拟引脚的参考电压。在嵌入式应用中引脚获取模拟电压值之后，将根据参考电压将模拟值转换到 0～1023（2^10）。该函数为无返回值函数，参数为 type 类型，有 3 种类型（DEFAULT/INTERNAL/EXTERNAL），具体含义如下：
   DEFAULT：默认值，参考电压为 5V。

INTERNAL：低电压模式，使用片内基准电压源。
EXTERNAL：扩展模式，通过 AREF 引脚获取参考电压
注意：如果在 AREF 引脚加载外部参考电压，需要使用一个 5KW 的上拉电阻，这会避免由于设置不当造成控制芯片的损坏。

2. analogRead(pin)
用于读取引脚的模拟量电压值，每读一次需要花 100ms 的时间。参数 pin 表示所要获取模拟量电压值的引脚，该函数返回值为 int 型，表示引脚的模拟量电压值，范围在 0～1023。
函数原型：int analogRead(uint8_t pin)

3. analogWrite(pin, value)——PWM（（Pulse Width Modulation，脉冲宽度调制））
analogWrite 函数通过 PWM 的方式在引脚上输出一个模拟量，较多的应用在 LED 亮度控制、电机转速控制等方面。在 Arduino 中执行该操作后，应该等待一定时间后才能对该引脚进行下一次操作。Arduino 中的 PWM 的频率大约为 490Hz。该函数支持以下引脚：3、5、6、9、10、11。在 Arduino 控制板上引脚号旁边标注～的就是可用作 PWM 的引脚。
函数原型如下：void analogWrite(uint8_t pin, int val)

高级 I/O

1. pulseIn(pin,state,timeout)
用于读取引脚脉冲的时间长度，脉冲可以是 HIGH 或 LOW。如果是 HIGH，函数将先等引脚变为高电平，然后开始计时，一直到变为低电平为止。返回脉冲持续的时间长短，单位为 ms。如果超时还没有读到的话，将返回 0。
pulseIn 函数返回值类型为无符号长整型（unsigned long），3 个参数分别表示脉冲输入的引脚、脉冲响应的状态（高脉冲或低脉冲）和超时时间（这个参数可以不用，默认为 1 秒）。

2. shiftOut(dataPin, clockPin, bitOrder, val)
shiftOut 函数无返回值，能够将数据通过串行的方式在引脚上输出，相当于一般意义上的同步串行通信，这是控制器与控制器、控制器与传感器之间常用的一种通信方式。
dataPin：数据输出引脚，数据的每一位将逐次输出。引脚模式需要设置成输出。
clockPin：时钟输出引脚，为数据输出提供时钟，引脚模式需要设置成输出。
bitOrder：数据位移顺序选择位，该参数为 byte 类型，有两种类型可选择，分别是高位先入 MSBFIRST 和低位先入 LSBFIRST。
val：所要输出的数据值。
函数原型：void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val)

3. shiftIn(dataPin, clockPin, bitOrder)
将一个数据的一个字节一位一位的移入。从最高有效位（最左边）或最低有效位（最右边）开始。对于每个位，先拉高时钟电平，再从数据传输线中读取一位，再将时钟线拉低。注意：这是一个软件实现；Arduino 提供了一个硬件实现的 SPI 库，它速度更快但只在特定脚有效。
dataPin：输出每一位数据的引脚(int)
clockPin：时钟脚，当 dataPin 有值时此引脚电平变化(int)
bitOrder：输出位的顺序，最高位优先或最低位优先
函数原型：uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder)

4． tone()
在一个引脚上产生一个特定频率的方波（50%占空比）。持续时间可以设定，否则波形会一直产生直到调用 noTone()函数。该引脚可以连接压电蜂鸣器或其他喇叭播放声音。在同一时刻只能产生一个声音。如果一个引脚已经在播放音乐，那调用 tone()将不会有任何效果。如果音乐在同一个引脚上播放，它会自动调整频率。使用 tone()函数会与 3 脚和 11 脚的 PWM 产生干扰（Mega 板除外）。注意：如果你要在多个引脚上产生不同的音调，你要在对下一个引脚使用 tone()函数前对此引脚调用 noTone()函数。
tone(pin, frequency)
tone(pin, frequency, duration)
pin：要产生声音的引脚
frequency: 产生声音的频率，单位 Hz，类型 unsigned int
duration：声音持续的时间，单位毫秒（可选），类型 unsigned long

5． noTone(pin)
停止由 tone()产生的方波。pin: 所要停止产生声音的引脚

时间函数

1. millis( )
应用 millis 函数可获取机器运行的时间长度，单位 ms。大概 50 天溢出一次。函数返回值为 unsigned long 型，无参数。函数原型：unsigned long millis()
2. micros()
功能同 millis()，不过单位是 us，大概 70 分钟溢出一次，返回值也是 unsigned long 型。
3. delay(ms)
delay 函数是一个延时函数，在 Blink 程序中用到过，参数表示延时时长，单位是 ms。函数无返回值，原型如下：void delay(unsigned long ms)
4. delayMicroseconds(us)
delayMicroseconds 函数同样是延时函数，所不同的是其参数单位是 us（1ms=1000us）。函数原型如下：void delayMicroseconds(unsigned int us)

随机函数

1． randomSeed(seed)
用来设置随机数种子，seed 表示读模拟口 analogRead(pin)函数，随机种子的设置对产生的随机序列有影响。函数无返回值。
2． random(howSmall, howBig)
应用此函数可生成 howSmall 至 howBig 之间的随机数，函数参数和返回值都是 long 型。
3． random(max)
返回随机数据大于等于 0，小于 max。

数学函数

1. min(x, y)
   求最小值，函数原型：#define min(x, y) ((x)<(y)?(a):(b))
2. max(x, y)
   求最大值，函数原型：#define max(x, y) ((x)>(y)?(a):(b))
3. abs(x)
   计算绝对值，函数原型：#define abs(x) ((x)>0?(x):-(x))
4. constrain(x, low, high)
   约束函数，下限 low，上限 high，x 必须在 low 和 high 之间才能返回，否则返回 low（x<low 时）或 high（x>high 时）
   函数原型：#define constrain(x, low, high) ((x)<(low)?(low):((x)>(high)?(high):(x)))
5. map(x, in_min, in_max, out_min, out_max)
   将[in_min, in_max]范围内的 x 等比映射到[out_min, out_max]范围内。函数返回值为 long 型。
6. 三角函数：sin(rad),cos(rad),tan(rad) 返回值均为 double。
7. pow(base, exponent) 开方函数，base 的 exponent 次方

串口通信函数

用于 Arduino 控制板和一台计算机或其他设备之间的通信。所有的 Arduino 控制板有至少一个串口(又称作为 UART 或 USART)。Arduino Mega 有三个额外的串口：Serial 1 使用 19(RX)和 18(TX)，Serial 2 使用 17(RX)和 16(TX)，Serial3 使用 15(RX)和 14(TX)。Arduino Mega 特有:Serial1、Serial2、Serial3，其他的就用 Serial。

1. if (Serial) 表示指定的串口是否准备好。
2. Serial.begin(speed)
   串口定义波特率函数，speed 表示波特率（每秒传输数据的速率），如 300，1200，2400，4800，9600，14400，19200，28800，38400，57600，115200。常用为 9600
3. int Serial.available()
   判断缓冲器状态，回传有多少位元组（bytes）的资料尚未被 read()函数读取，返回值为 int 型，为 0 就没有数据在等待读取。
4. int Serial.read()
   读串口并返回收到参数，返回值是 int。 每次只能读取一个字节。
5. Serial.readBytes()
   从串口读字符到一个缓冲区。如果预设的长度读取完毕或者时间到了 (参见 Serial.setTimeout())，函数将终止.
   Serial.readBytes()返回放置在缓冲区的字符数。返回 0 意味着没有发现有效的数据。
   Serial.readBytes()继承自 Stream 类
   Serial.readBytes(buffer, length)
   buffer：用来存储字节（char[]或 byte[]）的缓冲区
   length：读取的字节数（int）
6. Serial.readBytesUntil()
   将字符从串行缓冲区读取到一个数组。如果检测到终止字符，或预设的读取长度读取完

毕，或者时间到了 (参见 Serial.setTimeout())函数将终止。
Serial.readBytesUntil()返回读入数组的字符数。返回 0 意味着没有发现有效的数据。
Serial.readBytesUntil()继承自 Stream 类
Serial.readBytesUntil(character, buffer, length)
character ：要搜索的字符（char）
buffer ：缓冲区来存储字节（char[]或 byte[]）
length:读的字节数（int）

7. Serial.setTimeout()
   设置使用 Serial.readBytesUntil() 或 Serial.readBytes()时等待串口数据的最大毫秒值. 默认为 1000 毫秒。
8. Serial.flush()
   清空缓冲器。等待超出的串行数据完成传输。（在 1.0 及以上的版本中，flush()语句的功能不再是丢弃所有进入缓存器的串行数据。）
9. Serial.find()
   从串行缓冲器中读取数据，直到发现给定长度的目标字符串。如果找到目标字符串，该函数返回 true，如果超时则返回 false
10. Serial.findUntil()
    从串行缓冲区读取数据，直到找到一个给定的长度或字符串终止位。如果目标字符串被发现，该函数返回 true，如果超时则返回 false。
11. Serial.parseFloat()
    查找传入的串行数据流中的下一个有效的浮点数。
12. Serial.parseInt()
    查找传入的串行数据流中的下一个有效的整数。
13. Serial.peek()
    返回传入的串行数据的下一个字节（字符），而不是进入内部串行缓冲器调取。也就是说，连续调用 peek()将返回相同的字符，与调用 read()方法相同。peek()继承自 Stream 类。
14. Serial.write()
    三种用法：Serial.write(val)、 Serial.write(str)、 Serial.write(buf, len)
    val: 以单个字节形式发的值
    str: 以一串字节的形式发送的字符串
    buf: 以一串字节的形式发送的数组
    len: 数组的长度

15. Serial.print(data)
    串口输出数据。 默认为十进制。其他进制输出格式如下：
    Serial.print(data, encoding) encoding 为编码格式，如下：
    DEC：十进制，默认　　HEX:十六进制
    OCT：八进制　　BIN：二进制
    BYTE：以 byte 进行传送，显示以 ASCII 方式
16. Serial.println(data)
    串口输出数据并带回车符。用法同 Serial.print(data)。
17. void serialEvent() 单独使用，构成伪事件，有点中断的意思。

中断函数

1. 中断使能函数：interrupts() 打开总中断 noInterrupts() 关闭总中断，无返回值，无参数
2. attachInterrupt(interrupt, function, mode)
   interrupt：中断源，其值可选 0 或 1，一般对应 2 号和 3 号数字引脚。中断可以再任何时候通过 attachInterrupt()命令进行改变。当重新使用 attachInterrupt()时，先前分配的中断就会从对应引脚上移除。
   function：中断处理函数，参数值为函数的指针（即函数名）
   mode：触发模式，有四种类型：LOW（低电平触发）、CHANGE（变化时触发）、RISING（沿上升延触发，即低电平变高电平时触发）、FALLING（沿下降延触发，即高电平变低电平时触发）
3. detachInterrupt(interrupt)
   用于删除中断，interrupt 表示中断源，可选 0 和 1

大多数的 Arduino 板有两个外部中断：0（数字引脚 2）和 1（数字引脚 3）。arduino Mege 有四个外部中断：数字 2（引脚 21），3（20 针），4（引脚 19），5（引脚 18）

注意：

1. 在中断函数中 delay 函数不能使用。
2. 使用 millis 函数始终返回进入中断前的值
3. 读取串口数据的话，可能会丢失。
4. 你应该声明一个变量来在未发生中断时储存变量，中断函数中使用的变量需要为 volatile 型，例如：volatile int state;

SPI（Serial Peripheral Interface）函数

SPI 接口：10（SS）11（MOSI）12（MISO）13（SCK）

概述：

SPI（Serial Peripheral Interface）是由摩托罗拉公司提出的一种同步串行外设接口总线，它可以使 MCU 与各种外围设备以串行方式进行通信以及交换信息，总线采用 3 根或 4 根数据线进行数据传输，常用的是 4 根线，即两条控制线（芯片选择 CS 和时钟 SCLK）以及两条数据信号线 SDI 和 SDO。

SPI 是一种高速、全双工、同步的通信总线。在摩托罗拉公司的 SPI 技术规范中，数据信号线 SDI 称为 MISO（Master-In-Slave-Out，主入从出），数据信号线 SDO 称为 MOSI（Master-Out-Slave-In，主出从入），控制信号线 CS 称为 SS（Slave-Select，从属选择），将 SCLK 称为 SCK（Serial-Clock，串行时钟）。在 SPI 通信中，数据是同步进行发送和接收的。数据传输的时钟基于来自主处理器产生的时钟脉冲，摩托罗拉公司没有定义任何通用的 SPI 时钟规范。

2. SPI接口数据传输

SPI以主从方式工作，允许一个主设备与多个从设备进行通信，主设备通过不同的SS信号线选择不同的从设备进行通信。典型的连接方式如图15所示。

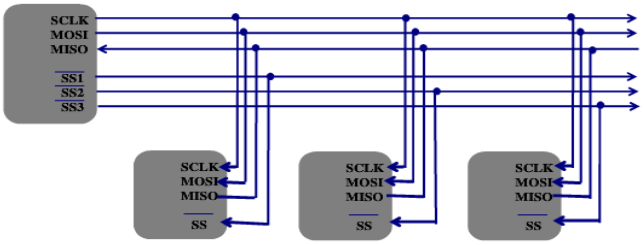

图15 Arduino SPI接口连接方式示意图

1. SPI.begin()
   初始化 SPI 总线，结果如下：
   pinMode(SCK, OUTPUT);//pin13　　digitalWrite(SCK, LOW);
   pinMode(MOSI, OUTPUT);//pin11　　digitalWrite(MOSI, LOW);
   pinMode(SS, OUTPUT);//pin10　　digitalWrite(SS, HIGH);
2. SPI.setBitOrder (bitOrder)
   作用是在设置串行数据传输时是先传输低位还是先传输高位，函数有一个 type 类型的参数 bitOrder，有 LSBFIRST（最低位在前）和 MSBFIRST（最高位在前）两种类型可选。函数无返回值。
3. SPI.setClockDivider (rate)
   作用是设置 SPI 串行通信的时钟，通信时钟是由系统时钟分频而得到，分频值可选 2、4、8、16、32、64 及 128，有一个 type 类型的参数 rate，有 7 种类型，对应 7 个分频值分别为 SPI_CLOCK_DIV2、SPI_CLOCK_DIV4、SPI_CLOCK_DIV8、SPI_CLOCK_DIV16、SPI_CLOCK_DIV32、SPI_CLOCK_DIV64 和 SPI_CLOCK_DIV128。函数默认参数设置是 SPI_CLOCK_DIV4，设置 SPI 串行通信时钟为系统时钟的 1/4。
4. SPI.setDataMode (mode)
   设置 SPI 的数据模式：时钟极性和时钟相位
   时钟极性：表示时钟信号在空闲时是高电平还是低电平
   时钟相位：决定数据是在 SCK 的上升沿采样还是在 SCK 的下降沿采样
   故有 mode 四种模式：
   SPI_MODE0 上升沿采样 下降沿置位 SCK 闲置时为 0
   SPI_MODE1 上升沿置位 下降沿采样 SCK 闲置时为 0
   SPI_MODE2 下降沿采样 上升沿置位 SCK 闲置时为 1
   SPI_MODE3 下降沿置位 上升沿采样 SCK 闲置时为 1
5. SPI.transfer (value)
   用来传输一个数据，由于 SPI 是一种全双工、同步的通信总线。所以传输一个数据实际上会发送一个数据，同时接收一个数据。函数的参数为发送的数据值，返回的参数为接收的数据值。函数原型：byte SPIClass::transfer(byte _data)
6. SPI.end ()
   停止 SPI 总线的使用（保持引脚的模式不改变）

位操作函数

1. lowByte() 取一个变量（例如一个字）的低位（最右边）字节。
2. highByte() 提取一个字节的高位（最左边的），或一个更长的字节的第二低位。
3. bitRead() 读取一个数的位。bitRead(x, n) X：想要被读取的数 N：被读取的位，0 是最重要（最右边）的位 该位的值（0 或 1）
4. bitWrite() 在位上写入数字变量 bitWrite(x, n, b) X：要写入的数值变量 N：要写入的数值变量的位，从 0 开始是最低（最右边）的位 B：写入位的数值（0 或 1）
5. bitSet() 为一个数字变量设置一个位 bitSet(x, n) X：想要设置的数字变量 N：想要设置的位，0 是最重要（最右边）的位
6. bitClear() 清除一个数值型数值的指定位(将此位设置成 0) bitClear(x, n) X：指定要清除位的数值 N：指定要清除位的位置，从 0 开始，0 表示最右端位
7. bit() 计算指定位的值（0 位是 1，1 位是 2，2 位 4，以此类推） bit(n) 需要计算的位

Arduino 控制器的 I2C/TWI 通讯

I2C 即 Inter－Integrated Circuit 串行总线的缩写，是 PHILIPS 公司推出的芯片间串行传输总线。它以 1 根串行数据线（SDA）和 1 根串行时钟线（SCL）（nano 是 A4-SDA, A5-SCL）实现了双工的同步数据传输。具有接口线少，控制方式简化，器件封装形式小，通信速率较高等优点。在主从通信中，可以有多个 I2C 总线器件同时接到 I2C 总线上，通过地址来识别通信对象。Arduino 已经为我们提供了 I2C 的库函数（Wire.h），这样我们就可以很轻松的玩 I2C 通讯了。

1. Wire.begin() //初始化 Wire 库，和设置 I2C 总线主从机
2. Wire.begin(address) //带地址参数就是从机，不带就是主机
3. Wire.requestFrom(address, count) //在启动 I2C 总线后，可以继续访问另一个地址，和访问次数
4. Wire.beginTransmission(address) //开始给从机发送地址
5. Wire.endTransmission() //结束本次 I2C 通讯，与上条函数成对使用
6. Wire.send() //发送数据
7. Wire.available() //用于判断数据是否有效，有效才开始接收，返回值为 byte 型
8. byte Wire.receive() //接收数据，返回值为 byte 型
9. Wire.onReceive(handler) //从机接收主机发来的数据，
10. Wire.onRequest(handler) //主机请求从机发送数据

编者注：

这个不是教程，只能算是一个小小的索引集，会通过具体编程实验来熟悉语法。本人能力有限，知识缺漏错误之处难免，欢迎读者指正！