数据类型

C 语言一共定义了 32 个关键词，其中数据类型关键词占 20 个。可见数据类型在 C 语言中的复杂程度。

1 数据类型关键字

A.基本数据类型（5 个）

void：声明函数无返回值或无参数，声明无类型指针，显式丢弃运算结果

char：字符型类型数据，属于整型数据的一种

int：整型数据，通常为编译器指定的机器字长

float：单精度浮点型数据，属于浮点数据的一种

double：双精度浮点型数据，属于浮点数据的一种

B .类型修饰关键字（4 个）

short：修饰 int，短整型数据，可省略被修饰的 int。

long：修饰 int，长整形数据，可省略被修饰的 int。

signed：修饰整型数据，有符号数据类型

unsigned：修饰整型数据，无符号数据类型

C .复杂类型关键字（5 个）

struct：结构体声明

union：共用体声明

enum：枚举声明

typedef：声明类型别名

sizeof：得到特定类型或特定类型变量的大小

D .存储级别关键字（6 个）

auto：指定为自动变量，由编译器自动分配及释放。通常在栈上分配

static：指定为静态变量，分配在静态变量区，修饰函数时，指定函数作用域为文件内部

register：指定为寄存器变量，建议编译器将变量存储到寄存器中使用，也可以修饰函数形参，建议编译器通过寄存器而不是堆栈传递参数

extern：指定对应变量为外部变量，即标示变量或者函数的定义在别的文件中，提示编译器遇到此变量和函数时在其他模块中寻找其定义。

const：与 volatile 合称“cv 特性”，指定变量不可被当前线程/进程改变（但有可能被系统或其他线程/进程改变）

volatile：与 const 合称“cv 特性”，指定变量的值有可能会被系统或其他进程/线程改变，强制编译器每次从内存中取得该变量的值

2. 解析

这里分析一下在 32 位系统里，这些关键词的含义。考虑到 Cortex 里基本不用浮点数，这里先不分析 **float** 和 **double** 类型。

基本的数据类型就是 **char** 和 **int**。

其中 char 占一个字节，int 默认与处理器位宽相同，在 Cortex M 系列等 32 位 CPU 里，其占 4 个字节。

short 和 **long** 用来修饰 int ，short 指定为两个字节，long 指定 4 个字节。

这样就有了以下几种数据类型：

char ： 占一个字节；

short int （通常只写作 short , int 可省略） ： 占两个字节 ；

long int （通常写作 long） ： 占四个字节；

int ： Cortex M 里默认为 4 字节，等同于 long int。

现实生活中的数都是有正负符号的，可以表示正数和负数。计算机里为了可以表示正数和负数，把最高位设置成符号位，其中 0 代表正数，1 代表负数。这种表示方法结合补码可以快速的对数字进行数学运算。这里牵扯到最多的是移位运算，想了解更多可以去查逻辑右移和算术右移的相关资料。

同时计算机里还需要另外一种数据，比如我想把单片机的某一路 IO 拉高，用来点亮 8 个 LED 灯 ，我们肯定更乐意去写“ P0OUT = 0xFF ；”这样的语句。这里的 0xFF 8 个 bit 都是有意义的。同样当我们设置流水灯时，肯定希望“P0OUT <<= 0x01 ;”时符号位能够以正常的数据形式来参与左移。

基于前面两种情况，C 语言在设计时采取了有符号数和无符号数两种数据类型模式，即 **signed** 和 **unsigned** 。

signed 型代表有符号数，最高位是符号位。移位运算时是算术移位。

unsigned 型代表无符号数。最高位是数据位。移位时是逻辑移位。

这样 C 语言中，数据类型就变成了 6 种：

Signed char	Unsigned char
Signed short int	Unsigned short int
Signed long int	Unsigned long int

另外：C 语言标准中写到对于未加修饰词的 char 和 int，编译器自行规划。

这句话造成了很大的问题，在 gnu 的编译器中， 未添加 修饰词的都作为 unsigned char 处理；但是 VC 的编译器中一律按照 signed char 处理。对于其他编译器，呵呵，我也不知道。

为了处理在不同编译器之间代码移植造成数据处理出错问题。在 C99 版本里，重新定义

了数据类型，其对应的文件是 stdint.h 和 stdbool.h 。下面是 stdint.h 里的一段：

```
    /* 7.18.1.1 */

    /* exact-width signed integer types */
typedef   signed          char int8_t;
typedef   signed short     int int16_t;
typedef   signed           int int32_t;
typedef   signed       __INT64 int64_t;

    /* exact-width unsigned integer types */
typedef unsigned          char uint8_t;
typedef unsigned short     int uint16_t;
typedef unsigned           int uint32_t;
typedef unsigned       __INT64 uint64_t;

    /* 7.18.1.2 */

    /* smallest type of at least n bits */
    /* minimum-width signed integer types */
typedef   signed          char int_least8_t;
typedef   signed short     int int_least16_t;
typedef   signed           int int_least32_t;
typedef   signed       __INT64 int_least64_t;

    /* minimum-width unsigned integer types */
typedef unsigned          char uint_least8_t;
typedef unsigned short     int uint_least16_t;
typedef unsigned           int uint_least32_t;
typedef unsigned       __INT64 uint_least64_t;

    /* 7.18.1.3 */

    /* fastest minimum-width signed integer types */
typedef   signed           int int_fast8_t;
typedef   signed           int int_fast16_t;
typedef   signed           int int_fast32_t;
typedef   signed       __INT64 int_fast64_t;

    /* fastest minimum-width unsigned integer types */
```

这样编程时使用 uint8_t 或者 int8_t 来代替 unsigned char、char 和 signed char。这样统一了不同编译器的代码移植问题。

另外 C99 规定了 long long 来定义 8 字节的数 ，当然考虑到代码移植问题，可以直接写 uint64_t 或者 int64_t 。

另外还有一些和硬件相关的修饰词：

auto 不用管理，我们定义的数据默认都是 auto 类型。有编译器自动生成和释放。

static 定义变量为静态变量，该变量和全局变量一样，不会在超出作用域之后不会被释放。这个用得多，不细说了。

extern 调用外部变量，也不细说了。

register 定义数据到寄存器，这个不会用就不要用。除了增加点执行效率，别的没什么用。而那点执行效率，呵呵。真的就只差那点效率的话，换个更好的 CPU 吧。考虑到代码移植什么的最好不要用这个。

volatile 指定每次读写数据时去内存读写，不使用缓存区。这个在做跨线程时很重要，一定要加。在多线程里不确定要不要加时，就加吧。肯定不会错。

const 在 Cortex M 中，const 修饰的变量会被编译成内容保存在 flash 里，而不是 ram 中。需要时再读取到 ram 中。所以这个词限定的 变量是不可更改的。

const uint8_t a = 5 ; a 就永远等于 5 了 ， 再写 a=10 ； 是非法的，编译器会报错。

struct union enum 没什么好讲的 ；

typedef 用来自定义数据类型，是一种语法，本身不是数据类型。也不说了。

sizeof 用来获取数据长度，也是一种语法，本身没什么好说的。以后说数据对齐的问题时，会经常说到这个玩意，不明白用法的可以先 google 一下。