###### 链接（多文件编译各C文件的函数名，变量名等不可出现重名问题）

编译成目标文件.o即是可用来做可链接的库文件的，这时候有以下两种gcc编译方式：第一种是用gcc将主程序.o和库文件.o一起编译成目标文件；第二种是在主程序源码里加一个include文件（一般是函数名的声明，一些变量的定义，宏定义这些），把用到的库文件.o里面的函数名包含进来，然后编译只编译主程序同时告诉链接的时候该去那里找链接文件.o（如果.o文件没有对应的函数名的定义，则会报链接错误）。

C语言中两种方式使用#include命令, #include <>与#include ””的区别

一种是在包含指令#include后面”<>”将头文件名括起来。这种方式用于标准或系统提供的头文件，到保存系统标准头文件的位置（具体位置取决与不同的编译器）查找头文件。标准函数库也是编译器的一部分，安装编译器时，库的也会跟着被装到特定的目录下并与编译器的相关参数产生关联。比如gcc有一个命令参数-lm意思是加入函数库libm.so

另一种是在包含指令#include后用双引号””将头文件包括起来。这种方式常用与程序员自己的头文件。用这种格式时，C编译器先查找当前目录是否有指定名称的头文件，然后再从标准头文件目录中查找。

C标准主要由两部分组成，一部分描述C的语法，另一部分描述C标准库。C标准库定义了一组标准头文件，每个头文件中包含一些相关的函数、变量、类型 声明和宏定义。要在一个平台上支持C语言，不仅要实现C编译器，还要实现C标准库，这样的实现才算符合C标准。不符合C标准的实现也是存在的，例如很多单 片机的C语言开发工具中只有C编译器而没有完整的C标准库。

动态函数库与静态函数库

静态函数库的特色：

扩展名：（扩展名为 .a）

这类的函数库通常扩展名为 libxxx.a 的类型；

编译行为：

这类函数库在编译的时候会直接整合到执行程序当中，所以利用静态函数库编译成

的文件会比较大一些喔；

独立执行的状态：

这类函数库最大的优点，就是编译成功的可执行文件可以独立执行，而不需要再

向外部要求读取函数库的内容 （请参照动态函数库的说明）。

升级难易度：

虽然可执行文件可以独立执行，但因为函数库是直接整合到可执行文件中， 因此

若函数库升级时，整个可执行文件必须要重新编译才能将新版的函数库整合到程序

当中。 也就是说，在升级方面，只要函数库升级了，所有将此函数库纳入的程序

都需要重新编译！

动态函数库的特色：

扩展名：（扩展名为 .so）

这类函数库通常扩展名为 libxxx.so 的类型；

编译行为：

动态函数库与静态函数库的编译行为差异挺大的。 与静态函数库被整个捉到程序

中不同的，动态函数库在编译的时候，在程序里面只有一个“指向 （Pointer）”的位

置而已。也就是说，动态函数库的内容并没有被整合到可执行文件当中，而是当可

执行文件要使用到函数库的机制时， 程序才会去读取函数库来使用。由于可执行

文件当中仅具有指向动态函数库所在的指标而已， 并不包含函数库的内容，所以

他的文件会比较小一点。

独立执行的状态：

这类型的函数库所编译出来的程序不能被独立执行， 因为当我们使用到函数库的

机制时，程序才会去读取函数库，所以函数库文件“必须要存在”才行，而且，函数

库的“所在目录也不能改变”，因为我们的可执行文件里面仅有“指标”亦即当要取

用该动态函数库时， 程序会主动去某个路径下读取，所以动态函数库可不

能随意移动或删除，会影响很多相依的程序软件喔！

升级难易度：

虽然这类型的可执行文件无法独立运行，然而由于是具有指向的功能， 所以，当

函数库升级后，可执行文件根本不需要进行重新编译的行为，因为可执行文件会直

接指向新的函数库文件 （前提是函数库新旧版本的文件名相同喔！）。

目前的 Linux distribution 比较倾向于使用动态函数库，因为如同上面提到的最重

要的一点， 就是函数库的升级方便！由于 Linux 系统里面的软件相依性太复杂了，如果

使用太多的静态函数库，那么升级某一个函数库时， 都会对整个系统造成很大的冲

击！因为其他相依的可执行文件也要同时重新编译啊！ 这个时候动态函数库可就有用

多了，因为只要动态函数库升级就好，其他的软件根本无须变动。

那么这些函数库放置在哪里呢？绝大多数的函数库都放置在：/lib64, /lib 目录

下！ 此外，Linux 系统里面很多的函数库其实 kernel 就提供了，那么 kernel 的函数库放

在哪里？呵呵！就是在 /lib/modules 里面啦！里面的数据可多着呢！不过要注意的是，

不同版本的核心提供的函数库差异性是挺大的，所以 kernel 2.4.xx 版本的系统不要想将

核心换成 2.6.xx 喔！ 很容易由于函数库的不同而导致很多原本可以执行的软件无法顺利

运行呢！（可以用ldd命令查询一个程序用到的动态链接库）。

链接 = 链接函数和全局变量：编译的时候只要函数的声明，全局变量的声明正确，编译器便可以生成目标文件（.obj），链接的时候需要指明需要链接的文件名，链接器会根据名称（函数名称，全局变量名称）在各个指明的obj文件中找到对应的函数实现（定义）以及全局变量定义；

如果有多个.obj文件，一般将这些文件打包，这样就不用一个一个的指明链接，打包后的文件便是库文件，在Windows下叫.lib文件，在Unix下叫.a文件（archive file）；C文件调用这些程序的函数接口是可以使用a库或so库，在程序文件中只需要include含有库中提供的函数接口声明的头文件即可；

（如何生成.so和.a文件即如何将.obj文件打包成.so和.a文件？如何使用.so和.a文件？）可以这么理解两者的区别：静态链接库的函数实现的指令等嵌入到了程序文件中，而动态链接库是程序在运行的过程中才去调用，并将其当作临时的指令文件执行，执行完后就不管（不像静态链接库，其调用部分一直存在于主程序中）

1. 静态链接库在被链接完就可以删除掉，不影响主程序运行，而动态链接库不行；
2. 如果静态链接库有改动，那么主程序也要重新编译链接一次，而动态链接库不需要；

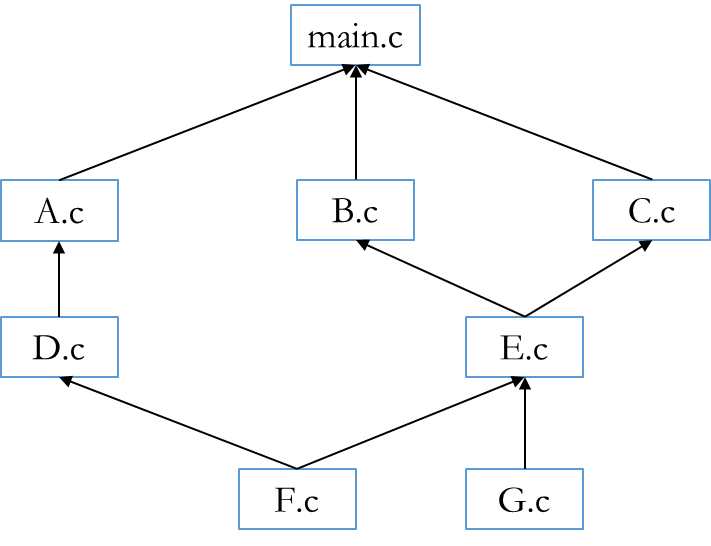
如何生成a库：

#### 代码风格借鉴

（先记一些零星片段，最后找时间与前面的整合）

###### C语言接口写法之一

“这种方法需要.c文件提供全局变量，尽量只在相关联的，组合在提供一套服务的.c文件之间提供，比如xxx\_api.c提供的底层接口”



1. 同级之间不要包含（“方案架构时逻辑上的同”，比如A.c和E.c不是同级）；
2. 不要越级包含（比如main.c不要直接包含底层的F.c）；
3. 不要被包含太多，也不要包含太多（除了main.c）；

以上C程序架构可方便扩充和裁剪（功能结构清晰，耦合性低），可移植性暂时不考虑；以后写C程序先按上面的架构来，慢慢摸索和完善（比如这种架构如何与多线程编程结合？线程函数应该在那里定义？同样看设计方案架构吧，一般在main.c的下一个层级定义）。

###### C语言main函数的主要形式有两种：

1. int main(void)
2. int main(int argc,char\*argv[]或者int main(int argc,char\*\*argv)

其中参数argc和argv用于运行时，把命令行参数传入主程序，argc是传给main函数的命令行参数个数（包括可执行程序名，argc=1时表示只有一个程序名称，存储在argv[0]中，而argv[argc]=NULL），具体如下：

argv[0]指向程序运行时的全路径名（执行C程序时写“./程序名”的原因）

argv[1]指向程序在DOS命令中执行程序名后的第一个字符串

argv[2]指向执行程序名后的第二个字符串

```

argv[argc]为NULL

以上基本对应Linux命令格式，在写程序时一般从argv[1]开始分析处理。

###### 枚举类型enum

enum类型变量是将变量的可能取值限定在声明时限定的有限几个。枚举变量不是那一串枚举的值的集合，而是集合中的某一个，就像无符号char型变量一样，它不可能是0~255所有整数的集合，而是其中的某个值。

枚举类型的声明是enum weekday{ sun,mou,tue,wed,thu,fri,sat };枚举类型变量的定义是enum weekday a,b,c;它和结构等自定义类型一样的格式。另外对于int、char等这些基本类型，我们一般只接触到它们的变量定义，没有见到它们的声明。

#### 回调函数（Call back function）

回调函数是作为其他函数的参数，用于在函数体中调用回调函数（在C语言中的实现方式是用函数指针）。其分为两种类型：同步阻塞型（立即执行或者在其调用者返回之前执行）和异步非阻塞型（不立即执行或在其调用者返回之后执行）。

“回调函数一般用在事件驱动型系统中”，比如点击一个按钮，触发相对应的事件，这些事件便由回调函数去处理。“返回值相同，形参列表相同”，名字（地址）不同的多个回调函数可以被一个事件驱动（调用者根据不同的情况选择调用那个回调函数）。良好的编程习惯是用typedef简化函数指针的定义。

还有一个用到回调函数（函数指针）的地方，就是排序算法函数，我们可以向其传入一个定义了排序规则的函数供排序算法函数调用（这也可以视为回调函数的一种应用）。

#### 指针常量和常量指针的区别

指针是一种变量，它里面存储的内容是内存地址。常量是指里面存储的内容不能发生改变的量。

###### 概念

**指针常量**就是指针本身是常量（指针不能被改变，因为它是常量），换句话说，就是指针里面所存储的内容（内存地址）是常量，不能改变。但是，内存地址所对应的内容是可以通过指针改变的。

**常量指针**就是指向常量的指针（常量不能被改变），换句话说，就是指针指向的是常量，不能通过该指针来修改它指向的内容。但是指针自身不是常量，它自身的值可以改变，从而指向另一个常量。

###### 声明

指针常量：Type \*const var

常量指针：const Type \*var（或者Type const\*var）

#### 条件编译

常用#ifdef、#endif的组合（如果后面的宏有定义就编译中间的代码），#ifndef和#endif与之正好相反。

###### 应用场景

1、防止头文件重包含：

#ifndef GPIO\_DRIVE\_H

#define GPIO\_DRIVE\_H

头文件内容

#endif

2、用#ifdef选择性编译代码（可解决跨平台问题或者其他需要条件编译的情况）

#### break语句

break语句用于switch语句和循环语句当中（包括for循环和while循环），前者是跳出switch，不再判断后面的case，直接执行switch后面的语句；后者是跳出循环。

###### 函数指针与typedef

**函数指针**：

1. 定义函数指针变量：int(\*p)(int, int);函数指针变量名字是p，指针变量类型是int(\*)(int, int)即“指向返回值是int输入参数是两个int的函数”的指针类型；如果不使用typedef，那么每次定义新函数指针都要int(\*x1)(int, int)，int(\*x2)(int, int)，int(\*x3)(int, int)，而有了typedef可以直接p x1，p x2，p x3。
2. 赋值方法：p = Func；
3. 使用方法：(\*p)(a, b)或者p(a, b)，因为函数名编译后就是一个函数指针（地址）；(\*p)(a, b)调用才是和函数名调用不一样，把它看作“函数指针调用的专属写法，然后编译器对这种写法的变量也稍有不同”；
4. 变量在编译的编译的背后都是对应一个地址，各种操作都是针对地址，但是变量不是地址，指针是存储变量地址的变量；“变量不是地址”应该从行为上理解：比如我把这片内存区的内容成为变量，虽然在编译的时候，操作这个变量用的是它的地址，但是这片区的内容不是它的地址；
   1. 从程序员的角度看，一个变量名代表是一片内存区的内容；而编译器会把变量名编译成一个地址，因为编译器需要该地址操作变量内存区（程序员角度）以完成程序员对变量的操作指示（**“程序员角度”的背后实现**）；对于指针变量，从程序员的角度看，指针变量的对应的内存区内容是一个地址；

**typedef**：

typedef int (\*SendFun)(unsigned char\*Buff, int Len);

typedef int (\*RecFun)(unsigned char\*Buff, int Len);

typedef void (\*ClearFun)(void);

typedef struct

{

SendFun Send;

RecFun Recv;

ClearFun Clear\_write;

ClearFun Clear\_read;

}mfcard\_interface;//**接口：方法的集合**

调用方法：

static mfcard\_interface card\_interface;

card\_interface.Send(SendBuf, Len);//等价于函数指针形式的调用，而不是函数形式的调用

###### 关于数组初始化

1. 如果要想数组初始化为0，那么建议使用memset，因为绝大多数编译器看到{0}都会将数组初始化为全0，但是不能保证所有的编译器都是这样；
2. 如果只初始化数组的前面几个元素，不能保证所有编译器的行为都是讲后面的默认为0，所以还是显式的令某个数组元素等于自己想要的值，比如字符数组的0；

###### 字符串处理'\0'的问题

对于C语言标准库相关函数，其一般会在新字符串末尾补'\0'，比如sprintf（转其他类型变量成字符也会在后面补0，坏处是会覆盖掉原来的内容），但是对于第三方就不一定了，'\0'可能只作为字符串末尾的判断而已，并不会主动地发送/加入/复制/…字符'\0'。

###### 指针数组的定义

dataType \*arrayName[length];

arrayName是一个数组，包含了length个元素，每个元素的类型为dataType \*；

#### C语言FILE文件操作

C语言标准规定了函数fopen，它的底层实现（如何处理缓冲，缓冲与磁盘之间如何交互等等，是操作系统编译器需要考虑的事），标准只封装了接口，编译器要屏蔽具体的硬件实现，编译出来的代码能完成描述的接口功能（完整的C语言编译器既要支持其语法编译，也要有其标准库的实现，C语言除了它的语法还包括它的标准库）。

\_CRTIMP FILE \* \_\_cdecl fopen(const char \*, const char \*);

第一个参数是文件名（绝对路径），如果不指定路径则默认是当前工作目录；第二个参数打开模式，比如"r"表示只读，"w"表示只写，"a"向文件新增数据（不删除原有数据），打开时位置指针移到文件末尾。

\_CRTIMP int \_\_cdecl fclose(FILE \*);关闭成功返回0，不成功返回非零值。

###### 文本文件操作相关

\_CRTIMP int \_\_cdecl fputc(int, FILE \*);

\_CRTIMP int \_\_cdecl fgetc(FILE \*);

字符读取函数fgetc()可从文件数据流中一次读取一个字符，然后读取光标移动到下一个字符，并逐步将文件的内容读出。如果字符读取成功，则返回所读取的字符，否则返回EOF(end of file)。EOF是表示数据结尾的常量，真值为-1。另外，要判断文件是否读取完毕，可利用feof()进行检查。未完返回0，已完返回非零值。

其他的文本文件操作函数还有fprintf、fscanf、fgets、fputs等等。

\_CRTIMP size\_t \_\_cdecl fread(void \*, size\_t, size\_t, FILE \*);

\_CRTIMP size\_t \_\_cdecl fwrite(const void \*, size\_t, size\_t, FILE \*);

###### 二进制文件操作相关

当要求一次存取一组数据（如一个数组、一个结构体变量的值），fread和fwrite函数可以解决该类问题。它们的调用形式一般为：

fread(buffer, size, count, fp);

fwrite(buffer, size, count, fp);

buffer：对于fread来说，指的是读入数据的存放地址；对于fwrite来说，是要输出数据的地址。

size：读写数据时，每笔数据的大小

count：读写数据的笔数

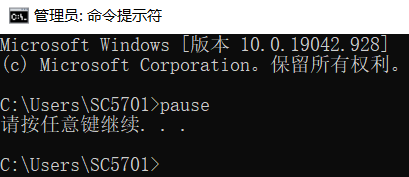
fp：文件指针

#### C语言常用标准库函数

C语言包括其语法和标准库；

###### system函数（stdlib.h）

Windows下是int system(char \*command);功能是执行一个dos指令，如system("pause")，直接在cmd界面输入pause，效果是：



Linux下是int system(const char \* string);功能是执行一个shell指令。

###### malloc函数（stdlib.h）

全称是memory allocation，用于申请一块连续的指定大小的内存块区域。函数原型是void \*malloc(unsigned int size);由于返回的是void\*型指针，所以必要时需进行类型转换。当该内存区不在使用时，一定要记得使用free函数释放。

如果分配成功则返回指向被分配内存的指针，否则返回空指针NULL；调用时需要进行强制类型转换，比如(char \*)，(int \*)，malloc返回的指针可以复制给一个单独的指针（用法就按单独指针那样用），也可以给指向数组的首地址的指针（用法就按数组那样用）；

因为malloc要么是单个变量大小，要么是变量数组；

###### memset（string.h）

函数原型：void \*memset(void \*s,int ch,size\_t n);//typedef unsigned int size\_t

功能：将s指向的地址开始的n个字节用ch替换并返回s；该函数是对较大的结构体或者数组进行清零操作的一种最快方法。

注意：ch的范围应该是0~255，因为memset是按字节替换，如果ch超范围了，那么它只能提取ch的后8位去替换。

###### scanf函数（stdio.h）

用法1（推荐用法）：scanf("%d%s",&i,str)输入时多个变量之间用空格、回车或者回车键隔开，多少个空格键等没有限制；

用法2：scanf("%d,%s",&i,str)输入时变量之间要用逗号隔开（英文输入法），而且要严格对应，即不能有多个逗号，逗号旁边也不能有空格这些东西。输入的是变量的地址？那字符串要不要取地址，应该不要，输入的要求应该不是变量的地址，而是通知输入的内容要存在那个地方，由于整型变量名并不能通知要存在那个地方（不是地址，不是指针变量），所以要取地址，而字符串变量本身是一个指针变量，可以通知内容要存在那个地方，所以不用取地址操作。

不过printf没有这些麻烦事。

char s[5];

scanf( "%4s", s );//限制字符串输入的长度（如果不限制，那么输入超出会报stack error）

###### strcpy函数（string.h）

即string copy（字符串复制），头文件是string.h，功能是把含有'\0'结束符的字符串复制到另一个地址空间。函数原型是char \*strcpy(char\* dest, const char \*src); 。

src和dest所指内存区域不可以重叠且dest必须有足够的空间来容纳src的字符串；返回值是指向dest的指针

###### strncpy（string.h）

strncpy函数用于将指定长度的字符串复制到字符数组中，是 C语言的库函数之一，来自 C语言标准库，定义于 string.h。语法形式为：char \*strncpy(char \*dest, const char \*src, int n)，表示把src所指向的字符串中以src地址开始的前n个字节复制到dest所指的数组中，并返回被复制后的dest。

复制字符串source中的内容（字符，数字、汉字....）到字符串destinin中，复制多少由maxlen的值决定。如果source的前n个字符不含NULL字符，则结果不会以NULL字符结束。如果n<source的长度，只是将source的前n个字符复制到destinin的前n个字符，不自动添加'\0'，也就是结果destinin不包括'\0'，需要再手动添加一个'\0'。如果source的长度小于n个字节，则以NULL填充destinin直到复制完n个字节。

source和destinin所指内存区域不可以重叠且destinin必须有足够的空间来容纳source的字符长度+'\0'。

###### strcmp（string.h）

strcmp函数是string compare(字符串比较)的缩写，用于比较两个字符串并根据比较结果返回整数。基本形式为strcmp(str1,str2)，若str1=str2，则返回零；若str1<str2，则返回负数；若str1>str2，则返回正数；

两个字符串自左向右逐个字符相比（按ASCII值大小相比较），直到出现不同的字符或遇'\0'为止；

char str1[10] = {'a', 'b', 'c', '\0'};

char str2[10] = {'a', 'b', 'c', 'd', 'e', '\0'};

strcmp(str1, str2)的结果是-100，而d的ASCII码十进制是100。

###### select函数

（还可以用来精准定时）

###### sleep函数

sleep(x)将调用该函数的线程挂起x秒，头文件是unistd.h

###### strdup和strndup函数

#include <string.h>

char \*strdup(const char \*s);

char \*strndup(const char \*s, size\_t n);

strdup() 函数将参数 s 指向的字符串复制到另一片空间中（该空间用malloc申请，所以要记得free），并返回指向复制字符串分配的空间。

而strndup只复制前面的n个字符。

###### sprintf函数

int sprintf(char \*string, char \*format [,argument,…];

头文件是stdio.h，作用是将后面的内容格式化到string指向的地址（字符串）；除了string，它的其他部分调用形式和printf相同（比printf多个char \*string）。

###### sscanf函数

###### strlen函数

头文件string.h，计算字符串长度，到'\0'但不包括'\0'（长度/个数=索引 + 1）；

###### access

#include<unistd.h>

int access(const char\* pathname, int mode);

pathname：

文件绝对路径（包括文件名）

mode：

F\_OK 值为0，判断文件是否存在

X\_OK 值为1，判断对文件是可执行权限

W\_OK 值为2，判断对文件是否有写权限

R\_OK 值为4，判断对文件是否有读权限

注：后三种可以使用或“|”的方式，一起使用，如W\_OK|R\_OK

###### memcmp（string.h）

int memcmp(const void \*str1, const void \*str2, size\_t n));其功能是把存储区 str1 和存储区 str2 的前 n 个字节进行比较。该函数是按字节比较的，位于string.h。

str1-- 指向内存块的指针。

str2-- 指向内存块的指针。

n-- 要被比较的字节数。

如果返回值 < 0，则表示 str1 小于 str2。

如果返回值 > 0，则表示 str2 小于 str1。

如果返回值 = 0，则表示 str1 等于 str2。

###### memcpy（string.h）

memcpy指的是C和C++使用的内存拷贝函数，函数原型为void \*memcpy(void \*destin, void \*source, unsigned n)；函数的功能是从源source所指的内存地址的起始位置开始拷贝n个字节到目标destin所指的内存地址的起始位置中。

返回值是指向dest的指针。

###### C语言关键字

restrict关键字是告诉编译器：“我”保证定义的指针对指向的地址是独享的，不会有其他指针也指向该区域，所有对该区域的修改都必须通过该指针来，编译器基于该承诺便可大胆的做出相关优化（具体是不是独享，编译器不知道也不会检查，编译器只会当它是独占的从而进行相关优化，程序员自己之后也要遵循自己的承诺，否则可能会出现意想不到的错误）；

static关键字是告诉编译器每次操作该变量时都要去原来的地方（内存中）读取出该值（拒绝优化，拒绝从寄存器中读取）；

###### #pragma pack(n)

设置结构体的对齐方式：以n字节对齐（关于字节对齐之前已经说过，默认的对齐方式由架构位数决定，比如32位架构以及64位架构）；改变默认字节对齐方式的用途之一是在进行相关硬件设备驱动开发时，显式告诉编译器在具体内存当中怎么对齐，比如经常要让结构体的成员变量连在一起，这时候就可以使用#pragma pack(1)，之后的结构体定义就按1个字节对齐（连在一起），使用#pragma pack()恢复默认对齐方式；

###### 补充知识：C语言.c文件和.h文件使用说明

在C语言中一般不对外提供全局变量，感觉这样容易乱，但可以提供变量类型，比如结构，枚举这些；全局变量放在.c文件中，不对外开放，C语言的这种对外服务模式是只提供的服务函数和变量类型（以及一些宏），不提供全局变量的访问，下面以链表为例（两种情况都不提供全局变量）：

可以提供链表结点类型以及链表相关操作函数，此时对应的服务模式有两种：

1是.c文件定义了链表类型，然后其他文件调用时需要实例化链表类型变量，并将其作为参数传给链表操作函数；在这种情况下，链表对于外部.c文件是可见的，因为它实例化了链表变量。

2不对外提供链表类型（将链表类型的定义放在.c文件中而不放在给其他文件包含的.h文件），在.c文件中内部维护相关的链表变量，同样也可以满足用户的链表相关操作；在这种情况下，需要定义查看链表的操作，因为链表是 内部定义和维护的，对用户不可见。

此外提一下对外提供全局变量（少用一些，尤其是大工程）：标准做法是在.c文件中声明全局变量V（可同时初始化），然后在.h文件中用extern type V进行外部变量的声明（不可初始化）。

**其他：**不是所有定义声明的函数都要对外开放，不开放的函数只在.c文件声明和定义即可，不需要在.h文件声明（对其他文件隐藏）。

提醒一点，除了外部使用.c文件提供的功能需要include其对应的.h文件外，它自己本身也要include自己的.h文件（这里再结合gcc命令以及C语言程序的编译一起总结）。

#### 临时

结构变量struct的定义以及typedef的用法；

结构体的声明（没用具体的结构变量）：

struct 结构名{

成员列表;

};

结构变量的定义：

struct 结构名 变量名1, 变量名2…；

struct 结构名{

成员列表;

}变量名1, 变量名2…;（声明和定义放一起）

typedef struct tagPoint

{

double x;//这里不可以写 double x = 0，因为只声明了数据类型没定义变量，C语言不允许这样的初始化

double y;

double z;

char str1[10];

char str2[10];

int k;

} Point;

Point a = {1, 2, 3, "ffdfs", {'a', 'b', '\0'};

或者typedef struct tagPoint Point; 此时Point不是结构变量；typedef还可以用在其他结构类型上；

“要精通查找算法，通过查找算法串起其他数据结构，如链表，数组，哈希表，树等等”

如果文本末尾没有换行符，那么vim会在文件末尾自动加上一个换行符（应该可以修改这个默认配置）

C语言程序中在任何出现字符串的地方，都要保证其末尾有'\0'的存在，相关字符串处理函数内存统一不用额外去考虑有没有'\0'的存在，因为已经默认是处理字符串了，而字符串末尾必有'\0'，否则就不叫字符串。

尽量不要随便用char \*指针指向一片空间（用void \*代替），这么做的话就默认这片空间是字符串（后面有'\0'）；尤其是作为输入参数的时候，char \*默认是字符串，后面某个位置必有'\0'。

字符串在初始化的的时候可以用等号，而定义完再赋值则不行：

char str1[10] = "dfsafds";

char \*str2 = "fdsagf";

str2 = "dddf";//错误，必须要手动操作（如strcpy函数）

这种初始化赋值会把'\0'也算上去（个人认为不是赋值，而是把指针指向等号后面的字符串内存区），sizeof(str1)应该是指针大小，而不是数组大小10；

字符数组元素个数算'\0'，但当前作为字符串使用，那么字符串长度不算末尾的'\0'；

不可以给未知字符串赋值，char \*str；str[0] = 'a'是错的，char \*str = "fs"; str[3] = 'a'也是错的（str[2]可以赋值，因为在编译时它是str的一部分），出错的原因是指针定义了，但指向的地址没有定义，所以应该使用字符数组；

switch case语句不能出现变量定义；

涉及到整型变量（包括结构体内的成员变量）尽量使用uint32\_t，int16\_t这类的定义（包含头文件<stdint.h>），这样更明确一点，可以提供兼容性；

sizeof的定义——sizeof的结果等于对象或者类型所占的内存字节数（所以sizeof也可以对该类型的变量使用，返回的是变量占据的内存大小，一般情况下它们的结果是相等的）。

###### C文件

一个C文件include的文件不要只包含其对应的h文件；比如a.c用的b.c提供的接口服务，那么以下做法是不推荐的：

a.h文件中

#include "b.h"

…（a.h自己的部分：a.c对外提供的数据类型和接口服务函数）

而是既然a.c用到了b.c的接口服务或者数据类型，那么a.c文件中就应该明确地：

#include "b.h"

#include "a.h"

其中a.h只包含对外开放的自己定义的数据类型或者对外开放的a.c的接口函数原型声明；

（第二种写法架构更清晰，较好地明确了每个C文件使用的依赖）

但要注意与一种情况区分：包含的h文件是对应的直接使用的接口函数声明的，不要再去include接口函数底层实现用到的接口函数对应的h文件（层级关系）；因为本文件没有用到更底层的接口（用到了也要明确包含，哪怕它的上层已经包含过了），一般情况下也用不到，因为现在所认知的C程序架构是层级的；

举个例子，a.c是b.c的底层调用的主要来源，a.c用到了stdio.h中的printf，于是a.h包含了stdio.h；而b.c包含了a.h，意味着也包含了stdio.h，但是一旦b.c显式使用了printf，那么b.c也应该显式包含stdio.h；而如果b.c全部使用a.c的接口（比如a.c把printf封装成另一个更符合需要的输出接口），则不用包含stdio.h。

以上规则的目的是明确C文件之间的层级依赖；

此外再说一下，不对外提供的数据类型和接口函数就不要写在.h文件中，只写在C文件中即可；

开发C项目时，根据目前的经验，分析完功能要求后，在实现时最好分底层函数和上层函数，上层函数可以同级之间调用，也可以获得下层提供的服务接口，但不要直接操作底层；先写底层，底层函数之间耦合性越低越好，然后再向上封装；

编译链接文件不需要链接其他文件：比如a.c调用到了b.c的函数，然后在生成a.o文件时，不需要链接b.o，因为如果要生成链接文件，那么命令参数是-c，意味着只编译（不链接），只编译就不需要去链接其他文件了，至于a.o里面的函数调用了b.c的函数导致该函数找不到实现，但在没有关系，只需在最终生成可执行文件的时候用-o链接a.o，b.o即可，链接器自己会从依赖中慢慢找（不要纠于“与a.o的函数找不到实现”）。链接文件（.o）只能是“一个C文件对应一个链接文件”，命令gcc -o file1.c file2.c file3.c的执行，中间也是对每一个C文件独立生成自己的.o文件，**链接时只用到C文件自己的内容（当然，预编译会将include的部分直接复制粘贴过来）**，它还不能使用其他C文件的函数，全局变量这些，链接的时候才根据声明的名字，到指定的C文件去找并链接成可执行文件（链接之后得到只能是可执行文件，不会是进一步封装的链接文件，那个叫库文件，生成库文件也不叫链接，库文件更像是把链接文件打个包）；如果C文件想把自己的全局变量和函数供其他C文件使用，应该将这些全局变量的声明以及函数的声明放在对应的头文件中供其他C文件include。

全局变量的使用：一个.c文件要让其他c文件使用自己的全局变量，正确做法是把全局变量的定义放在c文件中，全局变量的声明放在h文件中，然后让其他c文件包含该头文件。如果把全局变量的定义放在头文件中，其他c文件包含，此时各自c文件的编译是没有问题的，但是一旦将它们链接起来就出问题了，因为同名变量在多个c文件中被定义了；而如果是extern声明包含则没有问题，因为变量声明多少次都没有问题（类似于函数的声明）；

全局变量声明方法举例extern int tmp;

链接出错一般是在指定的链接文件中找不到定义；

**函数参数的合法性检查：**

1. 参数数据类型不作检查：比如上面说的字符串char \*（要不然就使用void \*），调用者要保证传入的参数是符合类型；而如果是直接面向用户，则需要作类型检查以及其他更周全的合法性检查；
2. 参数值的范围要作检查（不能是某些值或者不能在某个范围之外）；

#### make最基本使用

edit : main.o kdb.o command.o \

insert.o search.o

gcc -o edit main.o kbd.o command.o insert.o search.o

main.o : main.c defs.h

gcc -c main.c

kbd.o : kdb.c defs.h command.h

gcc -c kbd.c

command.o : command.c defs.h command.h

gcc -c command.c

insert.o : insert.c defs.h buffer.h

gcc -c display.c

search.o : search.c defs.h buffer.h

gcc -c search.c

clean :

rm edit main.o kbd.o command.o insert.o search.o

###### 基本写法和规则：

target … : prerequisites …

command

…

（command写法是新行 + tab键开头）prerequisites中如果有一个以上的文件比target文件要新的话（或者makefile定义的target在当前文件夹中不存在），command所定义的命令就会被执行。第一个target一般是make的最终生成目标：这会带来层级影响，比如insert.c更新了，那么insert.o对应的生成命令会被执行，导致insert.o变新，于是edit对应的链接命令会被执行（**make的核心规则便是这种依赖关系**）。

make clean意思是执行标号clean对应的命令（清除掉所有的生成文件）。

如果发现老是出现链接错误，那么可以查一查是不是原来错误的链接文件一直没有删掉（错误链接文件对应的依赖没有发生改动，导致每次make，错误链接文件也不会重新编译生成，一直留在文件夹中）；

**最后要生成的可执行文件必须放在最前面**；

#### Linux

Linux定义的文本文件（C代码文件也是文本文件），每一行的末尾都有\n换行符（反过来也成立：即文本文件与其他文件的区别就是文本文件有行的概念，而且以\n符划分每一行）。

而vim是一个文本编辑器，所以用vim打开的文件默认是文本文件，当vim保存退出时，如果用户没有手动加换行符，vim会自动添加（建议不要修改vim这个特性，因为一般情况下处理的是文本文件，如果遇到那些需要严格按换行符处理文本文件的程序就麻烦了，因为它是认为Linux文本文件每一行后面都有换行符，所以如果是文本文件，也应该要遵守该规则）；

如果不行手动保存为文本文件，则:set bin noeol+Enter然后再:wq；或者以vim -b xxx打开文件；

###### vim配置

执行 `vim ~/.vimrc`进入配置文件（单用户配置，全局配置去/etc里面）

基本设置

set nocompatible 不使用vi默认键盘布局

set number 显示行号

set autoindent 自动对齐

set smartindent 智能对齐

set showmatch 括号匹配模式

set ruler 显示状态行

set incsearch 搜索时一边输入一边显示效果

set tabstop=4 tab键为4个空格

set shiftwidth=4

set softtabstop=4

set cindent C语言格式对齐

set nobackup 不要备份文件

set clipboard+=unnamed 与windows共享剪贴板

syntax on 语法高亮

set mouse=a 鼠标支持

###### od命令

od -x xxx以16进制查看文件（左边显示的是地址，八进制形式）；

0d -c xxx以文本形式显示（转移字符以\0，\n等方式显示，如果数字对应的转义字符也没有，则以八进制显示该数字本身）；