1.两个操作数都是指向有限定符或无限定符的相容类型的指针，左边指针所指向的类型必须具有右边指针所指向类型的全部限定符。

char \*cp;

const char \*cpp;

则 cpp = cp 可以进行赋值，但反过来就不能进行赋值。

2.const char \* 表示指向一个具有const限定符的char类型的指针。

关键字const并不能把变量变成常量，只表示这个符号不能被赋值，即是只读的。但它并不能防止通过程序内部（甚至外部）的方法来修改真个值。

const最有用的之处是用来限定函数形参，这样函数就不会修改实参指针所指向的数据。

3.当执行算术运算时，操作数的类型如果不同，就会发生转换。数据类型一般朝着浮点数精度更高、长度更长的方向转换，整形数如果转换为signed不会丢失信息，就会转换为signed，否则转换为unsigned。

尽量不要在代码中使用无符号类型，以免增加不必要的复杂性。尤其是不能因为无符号不存在负值就用它来表示数量，这样在涉及混合类型的复杂细节时，不必担心边界情况（如-1被翻译为非常大的正数）。

4.在定义C函数时，缺省情况下函数的名字是全局可见的，加与不加extern关键字效果是一样的，对于链接到它所在的目标文件的任何东西都是可见的。如果要限制对函数的访问，就必须加static关键字。

C语言中对信息可见性的选择很有限，一个符号要么全局可见，要么对其他文件都不可见。

软件对象在大多数情况下应该缺省的采用有限可见性，当需要全局可见时应采用

5.int \*ap[] ap是元素为int指针的数组，即int \*(ap[])

int \*fp() fp是函数，返回int\*

c = getchar() != EOF 表示 c = (getchar() != EOF)

msb << 5 + lsb 表示 msb << (5 + lsb)

i = 1, 2 表示 (i = 1), 2 逗号运算符在所有运算符中优先级最低。

6.gets是过时的函数，只是用于和最初版本保持兼容。C语言的官方手册中，强烈建议用fgets()取代gets()，该函数会对读入的字符数设置一个限制，不会超出缓冲区范围。应该把gets(line)替换成

if(fgets(line, sizeof(line), stdin) == NULL)

exit(1);

7.C语言声明的优先级规则：

1）先从名字开始读取，然后按照优先级顺序依次读取

2）优先级从高到低依次是： 声明中被括号括起来的部分；

后缀操作符：括号()表示是一个函数，方括号[]表示一个数组

前缀操作符：\*号表示“指向...的指针”

3）如果const和（或）volatile关键字的后面紧跟类型说明符（如int，long等），则作用域类型说明符。在其他情况下，关键字作用于它左边紧邻的指针\*号。

例：char \* const \*(\*next)(); 表示next是一个指针，指向一个函数，该函数返回另一个指针，该指针指向一个类型为char的常量指针。

8.typedef关键字并不创建一个变量，而是宣称“这个名字是指定类型的同义词”。不要在一个typedef中放入几个声明器，也不要嵌到声明的中间部分。

不要为了方便起见对结构使用typedef，惟一的好处只是能使你不必书写“struct”关键字。

struct veg{int weight, price\_per\_lb;}veg; /\* 声明结构标签veg和变量veg \*/

typedef struct fruit{int weight, price\_per\_lb;}fruit; /\* 声明结构标签fruit和结构类型fruit \*/

typedef应该用在：1）数组、结构、指针以及函数的组合类型；

2）可移植类型。

9.typedef和#define的区别：

1）可以用其他类型说明符对宏类型名就行扩展，但对typedef所定义的类型名却不能这样做。

#define peach int

unsigned peach i；/\* 没问题 \*/

typedef int banana;

unsigned banana i; /\* 错误！ 非法 \*/

2) 在连续几个变量的声明中，用typedef定义的类型能保证声明中所有变量均为同一种类型，而用#define定义的类型则无法保证。

#define int\_ptr int \*

int\_ptr chalk, cheese;

经过宏扩展后，变为：

int \* chalk, cheese;

这使得chalk是一个指向int的指针，而cheese则是一个int。

10.strcmp函数用于比较两个字符串，当两个字符串相等时函数的返回值为0。令人费解的字符串比较：

if(!strcmp(s, "volatile")) return QUALIFIER;

可以用更好的方法——宏定义来使用：

#define STRCMP(a, R, b) (strcmp(a, b) R 0)

if(STRCMP(s, ==, "volatile"))

11.区分定义和声明：

1）声明相当于普通的声明：它所说明的并非自身，而是描述其他地方创建的对象；

2）定义相当于特殊的声明：它为对象分配内存。

12.指针和数组：

对于数组，每个符号的地址在编译时可知。如果编译器需要一个地址（可能还需要加上偏移量）来执行某种操作，就可以直接进行操作，并不需要增加指令首先取得具体的地址。所以extern char a[] 和 extern char a[100]是等价的，都只声明了a是一个数组，即一个内存地址。

对于指针，必须首先在运行时取得它的当前值，然后才能对它进行解除引用操作，进而进行查找。extern char \*p 告诉编译器p是一个指针，指向的对象是一个字符。而为了取得这个字符必须得到地址p的内容。

什么时候数组和指针式相同的：

1）表达式中数组名被编译器当做一个指向该数组第一个元素的指针；

2）下标总是与指针的偏移量相同；

3）函数参数声明中的数组名和指针。

13.编译器通常分割称几个更小的程序：预处理器（preprocessor）、语法和语义检查器（syntactic and semantic checker）、代码生成器（code generator）、汇编程序（assembler）、优化器（optimizer）、链接器（linker）。

静态链接：函数库的一份拷贝是可执行文件的物理组成部分；

动态链接：可执行文件只包含了文件名，让载入器在运行时能够寻找程序所需要的函数库。

1）动态链接可执行文件比相同的静态链接可执行文件的体积小（节省磁盘空间和虚拟内存）；

2）所有动态链接到某个特定函数库的可执行文件在运行时共享该函数库的一个单独拷贝；

3）动态链接使得函数库的版本升级更加容易，而不必因新版本函数库或操作系统的发布重新链接。

14.函数库链接：

1）动态库文件的扩展名是“.so”，而静态库文件的扩展名是“.a”；

2）通过 -l thread 选项，告诉编译器链接到libthread.so；

3）编译器期望在确定的目录找到库（编译器选项-L(-R) pathname、特殊位置/usr/lib）；

4）观察头文件，确认所使用的函数库（头文件的名字通常不与所对应的函数库名相似）

5）与提取动态库中的符号相比，静态库中的符号提取的方法限制更严

函数库选项应置于何处：始终将-l函数库选项放在编译命令行的最右边

15.Interpositioning：通过编写与库函数同名的函数来取代该库函数的行为（要格外小心）

准则：不要让程序中的任何符号成为全局的，除非有意把它们作为程序的接口之一

16.编程语言理论的经典对立之一就是代码和数据的区别，也可以认为是编译时和运行时的分界线。编译器的绝大部分工作和翻译代码有关；必要的数据存储管理的绝大部分都在运行时进行。

在UNIX中，段表示一个二进制文件相关的内容块，载入器取文件中每个段的映像并直接将它们放入内存中。本质上来说，在正在执行的程序中是一块内存区域。每个区域都有特定的目的。

文本段：包含程序的指令；

数据段：包含经过初始化的全局和静态变量，以及它们的值；

BSS段：保存没有初始化的变量，大小从可执行文件中得到，然后链接器得到对应大小的内存块，跟在数据段之后；

堆栈段：保存局部变量、临时数据、传递到函数中的参数等；

堆空间：用于动态分配的内存。

17.堆栈段向低地址扩展，空间有限。用途：

1）为函数内部声明的局部变量（也称自动变量）提供存储空间；

2）进行函数调用时，存储与函数相关的维护性信息（过程活动记录）；

3）用作暂时存储区，把部分计算结果压到堆栈中。

18.堆的常见问题：

1）释放或者改写仍在使用的内存（称为“内存损坏”）；

2）未释放不再使用到的内存（称为“内存泄漏”）

避免内存泄漏：每次调用malloc分配内存时，注意在以后要调用相应的free来释放它。

如何检测内存泄漏：使用top命令观察swap交换空间大小。

19.虚拟内存：用磁盘来扩充内存，程序实际需要使用的虚拟内存区段的内容被载入物理内存中；当物理内存中的数据有一段时间未被使用就被转移到硬盘中。

所有进程共享机器的物理内存，进程运行时数据在磁盘和内存之间来回移动。内存管理硬件负责把虚拟地址翻译为物理地址，并让一个进程始终运行于系统的真正内存中。对于不马上运行的进程，操作系统也可以暂时取回所有分配给它的物理内存资源，将所有信息备份到磁盘上，即被“换出”。

虚拟内存通过“页”的形式组织，即操作系统在磁盘和内存之间移动或保护的单位，一般为几K字节。

20.当硬件告诉操作系统一个有问题的内存引用时，会出现两种错误：总线错误（bus error）和段错误（segmentation fault）。操作系统通过向出错的进程发送一个信号与之交流。

1）总线错误：几乎是由于未对齐的读或写引起的。对齐（alignment）就是数据项只能存储在地址是数据项大小的整数倍的内存位置上，编译器通过自动分配和填充数据（在内存中）来进行对齐。一个会引起总线错误的程序如：

union {

char a[10];

int i;

}u;

int \*p = (int \*)&(u.a[1]);

\*p = 17;

2）段错误：内存管理单元的异常所致，通常包括以下几个直接原因：解除引用一个包含非法值的指针；接触引用一个空指针；在未得到正确权限时进行访问；用完了堆栈或者堆空间。如：

int \*p = NULL;

\*p = 17;

21.库函数调用和系统调用区别：函数库调用时语言或应用程序的一部分，系统调用时操作系统的一部分。

1）所有ANSI C编译器版本的C函数库都相同，而各个操作系统的系统调用是不同的；

2）函数库调用与用户程序相联系，系统调用调用的是系统内核的服务；

3）库函数调用在用户地址空间执行，系统调用在内核地址空间执行；

4）库函数调用属于过程调用，开销小，系统调用需要切换到内核上下文环境然后切换回来，开销较大

22.文件描述符：开放文件的每个进程表的一个偏移量，用于UNIX系统调用中，用于标识文件；

文件指针：保存了一个FILE结构的地址，FILE结构用于表示开放的I/O流，用于ANSI C标准I/O库调用中，用于标识文件。

23.面向对象编程：

1）封装：把相关的类型、数据和函数组合在一起；

2）继承：从一个类派生另外一个类，使前者所有的特征在后者自动可用，复用已经定义的操作；

3）多态：支持相关的对象具有不同的成员函数（原型相同），并允许对象与适当的成员函数进行运行时绑定。