1. 数组名代表地址,对于数组名a用不用&都一样,但是因为\*与&是逆运算，互相抵消,故\*&a不等于\*a而等于a(可以认为是有叠加保护)
2. (\*pa)++是可以的,直接对指向的值进行修改,因为指向的值都是变量(常量没有内存地址,故&只能结合左值)
3. 只可以用指针取字符串常量的地址，该类被认为是左值，而其他的字面值常量（例如1,2,3），则被认为是右值，不允许取地址。至于const常量，只能说是一个const修饰的只读变量，和字面值常量不同。
4. 数组名可看成一个指针，其中存储着数组的首地址,但是其又不完全等于指针

关系有 1.数组名是指针型常量,故不能作为左值.

2.指针变量pa也可以用下标法指向元素,pa[i]

3.void exchangeSort(double a[ ], int n)

形式参数用double a[]与double \*a都可以(都是指针的意思)

1. 从int转化为int\*必须用强制类型转换,不能隐式类型转换
2. 为什么不需要把数组作为返回值传出函数

因为传进去的是数组的地址（称为传址），此后的操作直接在这个地址上进行，修改的是数据内容本身，而不是数据的副本，实际上并不存在传入传出数据的动作

以前的变量做参数，传进去的是数值（称为传值），而不是变量的地址，无论如何操作，都修改不了实参变量的值(包括全局参数)，因此需要用返回值传出数据并赋值(这实际上是一种保护,防止模块化函数运行时把原参数改了,故用一中间变量来代替)

1. 数组溢出的字节数不多的话可能还有意义,太多就不行了
2. 关于数组溢出的数据读写问题:a[10]

a[11]存在哪里了？——在RAM中。

会引入错误吗？——如果地址虽然越界但仍在RAM中，不会引起大问题，最多是把你不想改写的数据改写了(如果数组a,b连在一起,就会把b[0]改写)。

每次都会这样吗？——一般都是(但有时也不会出现错误)，但是不排除哪次访问到了不得了的地方导致严重的错误。

1. 用函数交换变量值这种用函数来更改本源的操作只有用指针这一根本方法更改地址内容.
2. 字符串其实隐含一个指针，存放了字符串（首个字符）的起始地址

数组名同样隐含一个指针，存放了数组（首个元素）的起始地址

它们（中的地址）都可以被赋给其它指针，但都不能被赋值

可以认为字符串是一个有内容没名字的字符数组

1. 赋初值的过程比较复杂,就认为是右边的附到左边就行了,不要管过程

Ps: char \*str2=&”world”;是不行的

str2=&”world”也不行,似乎字符串与&不相容

1. 指针必须指向一个允许修改内容的存储空间，才能修改其中的内容，而字符串"world!"中的内容是不允许修改的
2. char \*str1="Hello"的str1[5]被填入了’\0’,而\0是空字符的意思,故始终不显示.并且虽然指针是字符型(1字节),但是%s会把汉字(2字节)也输出
3. 字符完全相同的字符串的地址相同,但只要有一个字符不同,地址都不同.
4. 字符串是碰到\0就停止,所以”hel\0lo”就只会输出hel.
5. int main() {

char src[15]="Hello,World!";

strncpy(d2, src+strlen(src)-6, 6);printf(“%s\n", d2);

很巧妙,利用指针的移动(char\*一次移动1字节),只输出字符串后6位字符

1. \*const与const\*的区别在于到底是谁是常量,\*const是指针为常量(地址不能改变),const\*是指向的内容为常量
2. 现在明白为什么scanf要加&，printf可以直接用变量名了吗？因为scanf要改变变量a的值，而printf只需要知道a的值,所以把a作为参数的时候，scanf要传入地址，printf只需要传入值！
3. 数组与指针的区别在于,数组是额外分配空间并附上常量指针,而指针只能在别人的空间上移动
4. &s就是从某一地址开始,持续输出字符,直至遇到\0
5. Sizeof与strlen的区别在于sizeof会算上\0,而strlen不会
6. 转义字符在scanf中无用
7. &不能对常量取地址
8. 字母与数字字符都只占一个字节,对应1个ascii码,而汉字等其他语言占两个字节,需要两个ascii码连用才能输出,否则是乱码
9. 重叠空间问题:double b=12.3;char a[5];gets(a); 如果字符串输入溢出,则会占用a的空间,使a的值发生改变.
10. Sqat函数输出double型
11. 静态变量(包括全局变量)系统会自动赋初值0
12. 局部变量优先(定义屏蔽),同名的话以局部变量为准
13. 复合语句也可以定义独立的局部变量
14. exit(0)是整个程序退出,是return0的升级版
15. Static int可以使变量生存周期扩大至与全局变量相同,但作用域不变.(因此不同函数中static int可以重名).
16. char \*whatday(int month[ ],char day[ ][5],int m,int d);

二维数组作形参，行数无意义，但必须指明列数(这是让系统能分清谁是哪一行,否则系统会按一维计算,所以在这种情况下\*a与a[][5]不等价,因为指针变量只为一维).

1. 对于二维数组month[][5],Month[1]是数组的第一行的地址,而month则为整个数组的首地址,也就是数组第0行的地址,也就是数组第0个元素的地址
2. 空类型 void主要用于定义未定类型的指针 void \*malloc(unsigned int size); malloc未定基类型.
3. 不同基类型的指针也可以互相赋值(就是地址) nt\* o;char\*h; o=h;
4. Int\* 指针也可以读取%s字符串,即&s与指针基类型无关,不管基类型是几个字节,自己就是一个一个字节读.
5. %04d表示输出4位整数,不足位用0补
6. 同一类型结构变量可以互相赋值，因为是变量！不同类型的之间当然不能相互赋值,成员都不一样.
7. 嵌套struct也是有的 struct staff{

char name[16]; int ID; float salary; struct date array}; [\\另一个结构变量](\\\\另一个结构变量)

1. Srand放在循环里与循环外是不同的:  
   循环外初始种子只赋一次time,当计算出第一个随机数后,系统会自动将其作为新的种子进行下一次计算,依次递推.

循环内种子会多次赋初值,但由于程序运行时间跨度短,不同初值几乎相同,导致每个随机数都相同.

1. 结构类型当然可以存放相同类型的变量,这一点与数组相同,区别在于:数组的元素的命名是用数字 a[1] ,而结构则是用文字 a.year ,不同需求用不同方法.
2. Gets函数不能接受’回车’,但可以接受空格与tab;但scanf(“%s”)遇到空格,tab与会车时都不接受且停止.

终止后，对终止字符处理不同。比如输入为"test\nabcd"。执行gets后，\n不会留在缓冲区中，即这时调用getchar得到的字符是'a'。执行scanf后，\n会留在缓冲区，这时调用getchar得到的字符是'\n'.

1. Puts函数会在最后自动输出’\n’,printf不行
2. EOF即end of file,是文件结束标志,可以认为就是宏定义#define EOF -1
3. Exit函数在stdlib.h里(但有时会自动找到)
4. Fscanf不用在%d后加空格、tab或回车以间隔数据(系统会识别),fprintf通常应在%d后加空格、tab或回车以间隔数据
5. scanf得接收到东西,回车对于scanf来说是见一次吐一次,一直没有接收到想接收的东西的话就一直处于等待输入的状态,是一定要接收到想接收的东西,然后回车才停止.
6. 从键盘输入文件名时还是要双斜杠.
7. 从文件中获取数据，显然要事先知道数据的类型、顺序等,否则格式字符不好确定
8. %15d是右对齐到十五个间距,%-15d是左对齐.
9. 文件这块很常见把文件输入输出函数写在表达式里,因为想利用文件输入输出函数的返回值判断是否结束.
10. rb与b只在是否进行字符转换,即回车和换行的转换问题上有区别
11. EOF不是文件中实际存在的内容，也不是只表示读文件到了结尾的状态，它还能表示 I/O 操作中的读、写错误以及其它一些关联操作的错误状态.所以,在文本文件中,错误与结束可以同时检测EOF,而在二进制文件中,最好不用文本输入输出函数,因为EOF会与-1混淆,这就是为什么二进制输入输出函数返回值是数据个数
12. getch()函数是在输入一个字符后自动结束,无需回车,常用于press any keys to continue
13. 从文件读结构数组时只能用fscanf一个一个成员读,若用fread读则会把空格读入(全盘复制)
14. 读二进制文件：无法用任何数字或字符表示文件结束，可以使用feof()函数，返回非0值表示文件读写位置指针移到了文件尾.这是因为fread与fwrite的返回值均为unsigned int,不能用-1,而fscanf与fprintf返回值为int 可以为-1
15. fread与fwrite的原型:

fread(&stu[i],sizeof(struct student),1,fp);

fwrite(&stu[i],sizeof(struct student),1,fp);

Ps:上式红色处,说明fread与fwrite复制数据块时需要地址,而不是变量