# 指针比较

char str1[] = "abc";

char str2[] = "abc";

const char str3[] = "abc";

const char str4[] = "abc";

char \*str5 = "abc";

char \*str6 = "abc";

const char \*str7 = "abc";

const char \*str8 = "abc";

str1 == str2 //0

str3 == str4 //0

str5 == str6 //1

str6 == str7 //1

str7 == str8 //1

像str1到str4这些字符数组，局部变量在**栈空间分配**，虽然数据一样，但是各自都有自己的存储位置，所以他们比较的结果是0，即不相同；str5到str8这几个指针也是在**栈空间分配**，都指向"abc"，这个"abc"是存储在**数据区**的，也就是说它们指向的是同一个位置，所以输出是1。

# const（只读）

const要求其所修饰的对象为常量，不可对其修改和二次赋值操作（不能作为左值出现）。

const int a;

int const a;

上面两句是等价的，都表示一个常整型数，意思是

const int a;

a = 0; //error

const int a = 0;

a = 1; //error

上面的都是错误的，因为a是只读的，所以只能定义的时候赋值而且之后不可以改变。

char \* const p1;

char const \*p2;

const char \*p3;

const char \* const p4;

const位于\*的右侧，则const是修饰指针本身的，即指针本事是常量不可以修改；const位于\*的左侧，则const是用来修饰指针所指向的变量，即指针指向的变量是不可以修改的。

p1是指针常量，p1本身不可以被修改，而p1指向的内容可以被修改；

p2和p3是常量指针，本身可以被修改，而指向的数据不可以被修改；

p4是本身和指向的内容都不可以被修改。

所以在实际的使用中，char \* const p1在定义的时候就要赋值，否则先定义在赋值，就会出错。

char a = 'a';

char \* const p1;

p1 = &a;

上面就是错的，正确写法应该是

char a = 'a';

char \* const p1 = &a;

(\*p1)++; //ok

同样，char const \*p2可以不在定义的时候赋值，但是不可以通过(\*p2)修改指向的值；

char const \*p2;

p2 = &a;

(\*p2)++; *//error*

# static

static的使用场景包括三种：

1. 修饰函数体内的变量（局部变量）；
2. 修饰函数体外的变量（全局变量）；
3. 修饰函数。

对于情况1，static延长了局部变量的生命周期，static的局部变量，并不会随着函数的执行结束而被销毁，当它所在的函数被再次执行时，该静态局部变量会保留上次执行结束时的值。下面的代码执行会输出0,1。

void **test**(){

static int i = 0;

**printf**("%d\n", i);

i++;

}

int **main**(){

**test**();

**test**();

}

对于情况2和3，static是对它修饰的对象进行了**作用域限定**，static修饰的函数以及函数外的变量，都是**只能在当前的代码文件中被访问，其它的文件不能直接访问**。当多个模块中有重名的对象出现时，不妨把它们用static进行修饰。

static修饰全局变量：

1. 该变量只初始化一次，防止其他文件单元中被引用；
2. 该变量只在定义该变量的源文件内有效，在同一源程序其他源文件中不能使用。

static修饰函数：

1. static修饰的函数的作用域只在本文件；
2. static函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝。

# volatile

volatile的作用是告知编译器，它修饰的变量随时都可能被改变，因此，编译后的程序每次在使用该变量的值时，**都会从变量的地址中读取数据，而不是从寄存器中获取**。

# extern

extern可置于变量或者函数前，以表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或函数时，在其它模块中寻找其定义。另外，extern也可用来进行链接指定。

## assert

assert()是一个调试程序时经常使用的宏，在程序运行时它计算括号内的表达式，如果表达式为FALSE (0), 程序将报告错误，并终止执行。如果表达式不为0，则继续执行后面的语句，它的作用是终止程序以免导致严重后果，同时也便于查找错误。

# 问题

## char \*p = "abc"和char p[] = "abc"的区别

char p[] = "abc"，数组要么在静态存储区（全局数组），要么在栈上（局部变量），p是数组名，有一块内存，p的值是数组的起始地址。

"abc"被编译器处理后，是在全局静态区的一块内存空间，不可被改写，char\* p指向了它；p[]是得到了它的拷贝。

比如char \*p = "abc";这条语句被编译器解释后，"abc"在全局静态区，而p指针指向这个区域，所以肯定无法使用p来修改"abc"。而char p[] = "abc"，是在栈上的一个char数组，所以通过p可以来修改数组里的变量。

## strcpy和memcpy

1. strcpy只能复制字符串，而memcpy可以复制任意内容，例如字符数组、整型、结构体、类等;
2. strcpy不需要指定长度，它遇到被复制字符的串结束符'\0'就结束，**所以容易溢出**。memcpy则是根据其第3个参数决定复制的长度；

# 特殊函数

## sprintf

http://www.runoob.com/cprogramming/c-function-sprintf.html

发送格式化输出到 **str** 所指向的字符串

int **sprintf**(char \*str, const char \*format, ...)

format是格式化的参数，比如

"%c", "%d", "%x", "%o", "%f"分别用来格式化字符、有符号十进制数、无符号16进制数、有符号8进制、浮点，还有其他，就是没有二进制

char str[10];

**sprintf**(str, "c : %x", 100);

# 参考