<https://blog.csdn.net/yangle4695/article/details/52166007>

**strlen**

strlen所作的仅仅是一个计数器的工作，它从内存的某个位置（可以是字符串开头，中间某个位置，甚至是某个不确定的内存区域）开始扫描，直到碰到第一个字符串结束符'\0'为止，然后返回计数器值(长度不包含'\0')。

**sizeof**  
C语言中判断数据类型或者表达式长度符；不是一个函数，字节数的计算在程序编译时进行，而不是在程序执行的过程中才计算出来!  
  
  
程序举例编辑  
举例1：（在Visual C++6.0中运行通过）  
#include<string.h>  
#include<stdio.h>  
int main(void)  
{  
char\*s="GoldenGlobalView";  
printf("%s has %d chars",s,strlen(s));  
getchar();  
return 0;  
}  
**区别**  
strlen(char\*）函数求的是字符串的实际长度，它求得方法是从开始到遇到第一个'\0'，如果你只定义没有给它赋初值，这个结果是不定的，它会从aa首地址一直找下去，直到遇到'\0'停止。  
char aa[10];cout<<strlen(aa)<<endl; //结果是不定的  
char aa[10]={'\0'}; cout<<strlen(aa)<<endl; //结果为0  
char aa[10]="jun"; cout<<strlen(aa)<<endl; //结果为3（此处不完全正确，结果可能是不定的，如果编译器将其他的非零字符串放在aa[10]的内存地址后，strlen结果还是不定的，MSP430在IAR编译测试过）（但是在vc6中结果为3，编译器会在“处自动添上\0。）  
char aa[5]="hello";cout<<strlen(aa)<<endl; //结果5（此处不完全正确，理由如上，因为实际内存中，hello后面不一定是空（即\0））（还应视编译器而定）  
而sizeof（）返回的是变量声明后所占的内存数，不是实际长度，此外sizeof不是函数，仅仅是一个取字节运算符，strlen是函数。  
sizeof(aa) 返回10  
int a[10]; sizeof(a) 返回40 （根据语言int型 c 是四个字节 c++是四个 java 是两个）  
⒈sizeof操作符的结果类型是size\_t，它在头文件中typedef为unsigned　int类型。  
该类型保证能容纳实现所建立的最大对象的字节大小。  
⒉sizeof是取字节运算符（关键字），strlen是函数。  
⒊sizeof可以用类型做参数，strlen只能用char\*做参数，且必须是以''\0''结尾的。  
sizeof还可以用函数做参数，比如：  
short f();  
printf("%d\n",sizeof(f()));  
输出的结果是sizeof(short），即2。  
⒋数组做sizeof的参数不退化，传递给strlen就退化为指针了。  
⒌大部分编译程序 在编译的时候就把sizeof计算过了是类型或是变量的长度。这就是sizeof(x）可以用来定义数组维数的原因  
char str[20]="0123456789";  
long a=strlen(str); //a=10;  
int b=sizeof(str); //而b=20;  
6.strlen的结果要在运行的时候才能计算出来，是用来计算字符串的长度，不是类型占内存的大小。  
7.sizeof后如果是类型必须加括弧，如果是变量名可以不加括弧。这是因为sizeof是个操作符不是个函数。  
⒏当适用了于一个结构类型时或变量， sizeof 返回实际的大小，  
当适用一静态地空间数组， sizeof 归还全部数组的尺寸。  
sizeof 操作符不能返回动态地被分派了的数组或外部的数组的尺寸  
⒐数组作为参数传给函数时传的是指针而不是数组，传递的是数组的首地址，  
如：  
fun(char [8])  
fun(char [])  
都等价于 fun(char \*)  
在C++里参数传递数组永远都是传递指向数组首元素的指针，编译器不知道数组的大小  
如果想在函数内知道数组的大小， 需要这样做：  
进入函数后用memcpy拷贝出来，长度由另一个形参传进去  
fun(unsiged char \*p1,int len){ unsigned char\* buf = new unsigned char[len+1] memcpy(buf,p1,len);}  
我们能常在用到 sizeof 和 strlen 的时候，通常是计算字符串数组的长度  
看了上面的详细解释，发现两者的使用还是有区别的，从这个例子可以看得很清楚：  
char str[20]="0123456789";  
int a=strlen(str); //a=10; >>>> strlen 计算字符串的长度，以结束符 0x00 为字符串结束。  
int b=sizeof(str); //而b=20; >>>> sizeof 计算的则是分配的数组 str[20] 所占的内存空间的大小，不受里面存储的内容改变。  
上面是对静态数组处理的结果，如果是对指针，结果就不一样了  
char\* ss = "0123456789";  
sizeof(ss) 结果 4>>>>ss是指向字符串常量的字符指针，sizeof 获得的是一个指针的值所占的空间，应该是长整型的，所以是4  
sizeof(\*ss) 结果 1>>>> \*ss是第一个字符 其实就是获得了字符串的第一位'0' 所占的内存空间，是char类型的，占了 1 位  
strlen(ss)= 10 >>>> 如果要获得这个字符串的长度，则一定要使用 strlen  
sizeof返回对象所占用的字节大小. //正确  
strlen返回字符个数. //正确  
在使用strlen时，有一个很特别的情况，就是数组名到指针蜕变，  
char Array[3] = {'0'};  
sizeof(Array) == 3;  
char \*p = Array;  
strlen(p) == 1;//sizeof(p）结果为4  
在传递一个数组名到一个函数中时，它会完全退化为一个指针  
----------------------------------------------------------  
看完以上你是否很清楚sizeof和strlen的区别了呢？还不明白的话，我们看下面几个例子：  
第一个例子  
char\* ss = "0123456789";  
sizeof( ss) 结果 4 ===》ss是指向字符串常量的字符指针  
sizeof(\*ss) 结果 1 ===》\*ss是第一个字符  
大部分编译程序 在编译的时候就把sizeof计算过了 是类型或是变量的长度  
这就是sizeof(x）可以用来定义数组维数的原因  
char str[20]="0123456789";  
int a=strlen(str); //a=10;  
int b=sizeof(str); //而b=20;  
char ss[] = "0123456789";  
sizeof(ss) 结果 11 ===》ss是数组，计算到\0位置，因此是10+1  
sizeof(\*ss) 结果 1 ===》\*ss 是第一个字符  
char ss[100] = "0123456789";  
sizeof(ss) 结果是100 ===》ss表示在内存中的大小 100×1  
strlen(ss) 结果是10 ===》strlen是个函数，内部实现是用一个循环计算到\0之前为止  
int ss[100] = "0123456789";  
sizeof(ss) 结果 400 ===》ss表示在内存中的大小 100×4  
strlen(ss) 错误 ===》strlen的参数只能是char\* 且必须是以'\0'结尾的  
char q[]="abc";  
char p[]="a\n";  
sizeof(q),sizeof(p),strlen(q),strlen(p);  
结果是 4 3 3 2  
第二个例子  
class X{ int i; int j; char k;};X x;  
cout<<sizeof(X)<<endl; 结果 12 ===》内存补齐  
cout<<sizeof(x)<<endl; 结果 12 同上  
第三个例子  
char szPath[MAX\_PATH]  
如果在函数内这样定义，那么sizeof(szPath）将会是MAX\_PATH，但是将szPath作为虚参声明时（void fun(char szPath[MAX\_PATH]）），sizeof(szPath）却会是4（指针大小）  
还有一位网友的说明也很好：  
其实理解 sizeof 只需要抓住一个要点：栈  
程序存储分布有三个区域：栈、静态和动态。能够从代码直接操作的对象，包括任何类型的变量、指针，都是在栈上的；动态和静态存储区是靠栈上的指针来间接操作的。sizeof 操作符，计算的是对象在栈上的投影体积；记住这个就很多东西都很清楚了。  
char const \* static\_string = "Hello";  
sizeof(static\_string) 是 sizeof 一个指针，所以在 32bit system 是 4  
char stack\_string[] = "Hello";  
sizeof(stack\_string) 是 sizeof 一个数组，所以是 6 \* sizeof(char)  
char \* string = new char[6];  
strncpy(string,"Hello",6");  
sizeof(string) 是 sizeof 一个指针，所以还是 4。和第一个不同的是，这个指针指向了动态存储区而不是静态存储区。  
不管指针指向的内容在什么地方，sizeof 得到的都是指针的栈大小  
C++ 中对引用的处理比较特殊；sizeof 一个引用得到的结果是 sizeof 一个被引用的对象的大小；所以  
struct O{ int a,b,c,d,e,f,g,h;};int main(){ O & r = \*new O; cout << sizeof(O) << endl; // 32 cout << sizeof r << endl; // 也是 32 system("PAUSE");}  
r 引用的是整个的 O 对象而不是指向 O 的指针，所以 sizeof r 的结果和 sizeof O 完全相同。  
自定义函数实现strlen（）函数的功能  
下面几种实现strlen函数的源代码大家参考  
例1  
  
#include<stdio.h>  
#include<assert.h>  
typedefunsignedintu\_int;  
u\_intMystrlen(constchar\*str){  
u\_inti;  
assert(str!=NULL);  
for(i=0;str[i]!='\0';i++);  
returni;  
}  
例2  
  
intstrlen(constchar\*str){  
assert(str!=NULL);  
intlen=0;  
while((\*str++)!='\0')  
len++;  
returnlen;  
}  
例3  
  
intstrlen(constchar\*str){  
assert(str);  
constchar\*p=str;  
while(\*p++!=NULL);  
returnp-str-1;  
}  
例4  
  
intstrlen(constchar\*str){  
assert(str);  
if(\*str==NULL)  
return0;  
else  
return（1+strlen(++str));  
}  
例5  
  
/\*\*\*strlen-Findthelengthofastring\*@s:Thestringtobesized\*/  
size\_tstrlen(constchar\*s){  
constchar\*sc;  
for(sc=s;\*sc!='\0';++sc)/\*nothing\*/;  
returnsc-s;  
}  
以上各种实现的方式都是大同小异的，有的用的是变量，有的用的是指针。  
其中，最后一个用的是递归的方式。其实，在实现库函数的时候，是规定不可以  
调用其他的库函数的，这里只是给大家一个方法，可以不用变量就可以实现strlen。