sprintf 编辑

字符串格式化命令，主要功能是把格式化的数据写入某个字符串中。sprintf 是个变参函数。

中文名 字符串格式输出 外文名 string print format 外语缩写 sprintf 主要功能 把格式化的数据写入某个字符串中 头文件stdio.h 原 型 int sprintf 返回值 字符串长度

目录

1 函数简介

▪ 功能

▪ 头文件

▪ 原型

▪ 参数列表

▪ 相关函数

2 说明及应用

▪ 例子

▪ 转换字符

▪ 格式化数字字符串

▪ 字符/Ascii码

3 常见问题

函数简介编辑

功能

把格式化的数据写入某个字符串缓冲区。

头文件

stdio.h

原型

int sprintf( char \*buffer, const char \*format, [ argument] … );

参数列表

buffer：char型指针，指向将要写入的字符串的缓冲区。

format：格式化字符串。

[argument]...：可选参数，可以是任何类型的数据。

返回值：字符串长度（strlen）

相关函数

int sprintf\_s(char \*buffer,size\_t sizeOfBuffer,const char \*format, [argument] ... );

int \_sprintf\_s\_l(char \*buffer,size\_t sizeOfBuffer,const char \*format,locale\_t locale ,[argument] ... );

int swprintf\_s(wchar\_t \*buffer,size\_t sizeOfBuffer,const wchar\_t \*format ,[argument]...);

int \_swprintf\_s\_l(wchar\_t \*buffer,size\_t sizeOfBuffer,const wchar\_t \*format,locale\_t locale ,[argument]…);

template <size\_t size>

int sprintf\_s(char (&buffer)[size],const char \*format, [argument] ... ); //仅存在于C++

template <size\_t size>

int swprintf\_s(wchar\_t (&buffer)[size],const wchar\_t \*format ,[argument]...); //仅存在于C++

说明及应用编辑

例子

#include<stdio.h>/\*某个stdio.h\*/

int　main()/\*主函数“整数”类型\*/

{

　char　buffer[50];/\*“字符”类型的数组，下面共有50个元素。\*/

　int　n,a=5,b=3;/\*三个变量都为“整数”类型,intn中间要有空格\*/

n=sprintf(buffer,"%d plus %d is %d",a,b,a+b);/\*赋予数值\*/

printf("[%s]is a string %d chars long\n",buffer,n);/\*“格式输出函数”\*/

return　0;/\*“返回零”

也就是程序正常退出\*/

}

输出结果：

[5 plus 3 is 8] is a string 13 chars long

sprintf格式的规格如下所示。[]中的部分是可选的。

%[指定参数][标识符][宽度][.精度]指示符

若想输出'%'本身时, 请使用'%%'处理。

1. 处理字符方向。负号时表示从后向前处理。

2. 填空字元。 0 的话表示空格填 0；空格是内定值，表示空格就放着。

3. 字符总宽度。为最小宽度。

4. 精确度。指在小数点后的浮点数位数。

转换字符

%% 印出百分比符号，不转换。

%c 整数转成对应的 ASCII 字元。

%d 整数转成十进位。

%f 倍精确度数字转成浮点数。

%o 整数转成八进位。

%s 整数转成字符串。

%x 整数转成小写十六进位。

%X 整数转成大写十六进位。

$money = 123.1

$formatted = sprintf ("%06.2f", $money); // 此时变数 $ formatted 值为 "123.10"

$formatted = sprintf ("%08.2f", $money); // 此时变数 $ formatted 值为 "00123.10"

$formatted = sprintf ("%-08.2f", $money); // 此时变数 $ formatted 值为 "123.1000"

$formatted = sprintf ("%.2f%%", 0.95 \* 100); // 格式化为百分比

%08.2f 解释:

%开始符

0是 "填空字元" 表示,如果长度不足时就用0来填满。

8格式化后总长度

2f小数位长度，即2位

第3行值为"00123.10" 解释:

因为2f是(2位)+小数点符号(1位)+前面123(3位)=6位，总长度为8位,故前面用[填空字元]0表示，即00123.10

第4行值为"123.1000" 解释:

-号为反向操作，然后填空字元0添加在最后面了

在将各种类型的数据构造成字符串时，sprintf 的强大功能很少会让你失望。由于sprintf 跟printf 在用法上几乎一样，只是打印的目的地不同而已，前者打印到字符串中，后者则直接在命令行上输出。这也导致sprintf 比printf 有用得多。

sprintf 是个变参函数，定义如下：

int sprintf( char \*buffer, const char \*format [, argument] ... );

除了前两个参数类型固定外，后面可以接任意多个参数。而它的精华，显然就在第二个参数：

格式化字符串上。

printf 和sprintf 都使用格式化字符串来指定串的格式，在格式串内部使用一些以“%”开头的格式说明符（format specifications）来占据一个位置，在后边的变参列表中提供相应的变量，最终函数就会用相应位置的变量来替代那个说明符，产生一个调用者想要的字符串。

格式化数字字符串

sprintf 最常见的应用之一莫过于把整数打印到字符串中，所以，sprintf 在大多数场合可以替代

itoa。

如：

//把整数123 打印成一个字符串保存在s 中。

sprintf(s, "%d", 123); //产生"123"

可以指定宽度，不足的左边补空格：

sprintf(s, "%4d%4d", 123, 4567); //产生：" 1234567"

当然也可以左对齐：

sprintf(s, "%-4d%4d", 123, 4567); //产生："123 4567"

也可以按照16 进制打印：

sprintf(s, "%8x", 4567); //小写16 进制，宽度占8 个位置，右对齐

sprintf(s, "%-8X", 4568); //大写16 进制，宽度占8 个位置，左对齐

这样，一个整数的16 进制字符串就很容易得到，但我们在打印16 进制内容时，通常想要一种左边补0 的等宽格式，那该怎么做呢？很简单，在表示宽度的数字前面加个0 就可以了。

sprintf(s, "%08X", 4567); //产生："000011D7"

上面以”%d”进行的10 进制打印同样也可以使用这种左边补0 的方式。

这里要注意一个符号扩展的问题：比如，假如我们想打印短整数（short）-1 的内存16 进制表示形式，在Win32 平台上，一个short 型占2 个字节，所以我们自然希望用4 个16 进制数字来打印它：

short si = -1;

sprintf(s, "%04X", si);

产生“FFFFFFFF”，怎么回事？因为sprintf 是个变参函数，除了前面两个参数之外，后面的参数都不是类型安全的，函数更没有办法仅仅通过一个“%X”就能得知当初函数调用前参数压栈时被压进来的到底是个4 字节的整数还是个2 字节的短整数，所以采取了统一4 字节的处理方式，导致参数压栈时做了符号扩展，扩展成了32 位的整数-1，打印时4 个位置不够了，就把32 位整数-1 的8 位16 进制都打印出来了。

如果你想看si 的本来面目，那么就应该让编译器做0 扩展而不是符号扩展（扩展时二进制左边补0 而不是补符号位）：

sprintf(s, "%04X", (unsigned short)si);

就可以了。或者：

unsigned short si = -1;

sprintf(s, "%04X", si);

sprintf 和printf 还可以按8 进制打印整数字符串，使用”%o”。注意8 进制和16 进制都不会打印出负数，都是无符号的，实际上也就是变量的内部编码的直接的16 进制或8 进制表示。

控制浮点数打印格式

浮点数的打印和格式控制是sprintf 的又一大常用功能，浮点数使用格式符”%f”控制，默认保留小数点后6 位数字，比如：

sprintf(s, "%f", 3.1415926); //产生"3.141593"

但有时我们希望自己控制打印的宽度和小数位数，这时就应该使用：”%m.nf”格式，其中m 表示打印的宽度，n 表示小数点后的位数。比如：

sprintf(s, "%10.3f", 3.1415626); //产生：" 3.142"

sprintf(s, "%-10.3f", 3.1415626); //产生："3.142 "

sprintf(s, "%.3f", 3.1415626); //不指定总宽度，产生："3.142"

注意一个问题，你猜

int i = 100;

sprintf(s, "%.2f", i);

会打出什么东东来？“100.00”？对吗？自己试试就知道了，同时也试试下面这个：

sprintf(s, "%.2f", (double)i);

第一个打出来的肯定不是正确结果，原因跟前面提到的一样，参数压栈时调用者并不知道跟i相对应的格式控制符是个”%f”。而函数执行时函数本身则并不知道当年被压入栈里的是个整数，于是可怜的保存整数i 的那4 个字节就被不由分说地强行作为浮点数格式来解释了，整个乱套了。不过，如果有人有兴趣使用手工编码一个浮点数，那么倒可以使用这种方法来检验一下你手工编排的结果是否正确。

字符/Ascii码

我们知道，在C/C++语言中，char 也是一种普通的scalable 类型，除了字长之外，它与short，int，long 这些类型没有本质区别，只不过被大家习惯用来表示字符和字符串而已。（或许当年该把这个类型叫做“byte”，然后现在就可以根据实际情况，使用byte 或short 来把char 通过typedef 定义出来，这样更合适些）于是，使用”%d”或者”%x”打印一个字符，便能得出它的10 进制或16 进制的ASCII 码；反过来，使用”%c”打印一个整数，便可以看到它所对应的ASCII字符。以下程序段把所有可见字符的ASCII 码对照表打印到屏幕上（这里采用printf，注意”#”与”%X”合用时自动为16 进制数增加”0X”前缀）：

for(int i = 32; i < 127; i++) {

printf("[ %c ]: %3d 0x%#04X\n", i, i, i);

}

连接字符串

sprintf 的格式控制串中既然可以插入各种东西，并最终把它们“连成一串”，自然也就能够连接字符串，从而在许多场合可以替代strcat，但sprintf 能够一次连接多个字符串（自然也可以同时在它们中间插入别的内容，总之非常灵活）。比如：

1

2

3

4

5

6

char\*buf[60];

char\*who="I";

char\*whom="CSDN";

sprintf(buf,"%slove%s.",who,whom);

printf("%s",buf);

//输出结果："IloveCSDN."

strcat 只能连接字符串（一段以'\0'结尾的字符数组或叫做字符缓冲，null-terminated-string），但有时我们有两段字符缓冲区，他们并不是以 ’’结尾。比如许多从第三方库函数中返回的字符数组，从硬件或者网络传输中读进来的字符流，它们未必每一段字符序列后面都有个相应的’’来结尾。如果直接连接，不管是sprintf 还是strcat 肯定会导致非法内存操作，而strncat 也至少要求第一个参数是个null-terminated-string，那该怎么办呢？我们自然会想起前面介绍打印整数和浮点数时可以指定宽度，字符串也一样的。比如：

char a1[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G'};

char a2[] = {'H', 'I', 'J', 'K', 'L', 'M', 'N'};

如果：

sprintf(s, "%s%s", a1, a2); //Don't do that!

十有八九要出问题了。是否可以改成：

sprintf(s, "%7s%7s", a1, a2);

也没好到哪儿去，正确的应该是：

sprintf(s, "%.7s%.7s", a1, a2);//产生："ABCDEFGHIJKLMN"

这可以类比打印浮点数的”%m/nf”，在”%m.ns”中，m 表示占用宽度（字符串长度不足时补空格，超出了则按照实际宽度打印），n 才表示从相应的字符串中最多取用的字符数。通常在打印字符串时m 没什么大用，还是点号后面的n 用的多。自然，也可以前后都只取部分字符：

sprintf(s, "%.6s%.5s", a1, a2);//产生："ABCDEFHIJKL"

在许多时候，我们或许还希望这些格式控制符中用以指定长度信息的数字是动态的，而不是静态指定的，因为许多时候，程序要到运行时才会清楚到底需要取字符数组中的几个字符，这种动态的宽度/精度设置功能在sprintf 的实现中也被考虑到了，sprintf 采用”\*”来占用一个本来需要一个指定宽度或精度的常数数字的位置，同样，而实际的宽度或精度就可以和其它被打印的变量一样被提供出来，于是，上面的例子可以变成：

sprintf(s, "%.\*s%.\*s", 7, a1, 7, a2);

或者：

sprintf(s, "%.\*s%.\*s", sizeof(a1), a1, sizeof(a2), a2);

实际上，前面介绍的打印字符、整数、浮点数等都可以动态指定那些常量值，比如：

1

2

3

sprintf(s,"%-\*d",4,'A');//产生"65"

sprintf(s,"%#0\*X",8,128);//产生"0X000080"，"#"产生0X

sprintf(s,"%\*.\*f",10,2,3.1415926);//产生"3.14"

打印地址信息

有时调试程序时，我们可能想查看某些变量或者成员的地址，由于地址或者指针也不过是个32 位的数，你完全可以使用打印无符号整数的”%u”把他们打印出来：

1

sprintf(s,"%u",&i);

不过通常人们还是喜欢使用16 进制而不是10 进制来显示一个地址：

1

sprintf(s,"%08X",&i);

然而，这些都是间接的方法，对于地址打印，sprintf 提供了专门的”%p”：

1

sprintf(s,"%p",&i);

我觉得它实际上就相当于：

1

sprintf(s,"%0\*x",2\*sizeof(void\*),&i);

利用sprintf 的返回值

较少有人注意printf/sprintf 函数的返回值，但有时它却是有用的，sprintf 返回了本次函数调用最终打印到字符缓冲区中的字符数目。也就是说每当一次sprinf 调用结束以后，你无须再调用一次strlen 便已经知道了结果字符串的长度。如：

1

intlen=sprintf(s,"%d",i);

对于正整数来说，len 便等于整数i 的10 进制位数。

下面的是个完整的例子，产生10 个[0, 100)之间的随机数，并将他们打印到一个字符数组s 中，

以逗号分隔开。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

intmain()

{

srand(time(0));

chars[64];

intoffset=0;

for(inti=0;i<10;i++)

{

offset+=sprintf(s+offset,"%d,",rand()%100);

}

s[offset-1]='\n';//将最后一个逗号换成换行符。

printf(s);

return0;

}

设想当你从数据库中取出一条记录，然后希望把他们的各个字段按照某种规则连接成一个字符串时，就可以使用这种方法，从理论上讲，他应该比不断的strcat 效率高，因为strcat 每次调用都需要先找到最后的那个’’的位置，而在上面给出的例子中，我们每次都利用sprintf 返回值把这个位置直接记下来了。

MSDN中例子：

//crt\_sprintf.c//compilewith:/W3//Thisprogramusessprintftoformatvarious//dataandplacetheminthestringnamedbuffer.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

#include<stdio.h>

intmain(void)

{

charbuffer[200],s[]="computer",c='l';

inti=35,j;

floatfp=1.7320534f;//Formatandprintvariousdata:

j=sprintf(buffer,"String:%s\n",s);//C4996

j+=sprintf(buffer+j,"Character:%c\n",c);//C4996

j+=sprintf(buffer+j,"Integer:%d\n",i);//C4996

j+=sprintf(buffer+j,"Real:%f\n",fp);//C4996

//Note:sprintfisdeprecated;considerusingsprintf\_sinstead

printf("Output:\n%s\ncharactercount=%d\n",buffer,j);

}

Copy

Output:

String:computer

Character:l

Integer:35

Real:1.732053

charactercount=53

常见问题编辑

sprintf 是个变参函数，使用时经常出问题，而且只要出问题通常就是能导致程序崩溃的内存访问错误，但好在由sprintf 误用导致的问题虽然严重，却很容易找出，无非就是那么几种情况，通常用眼睛再把出错的代码多看几眼就看出来了。

缓冲区溢出

第一个参数的长度太短了，没的说，给个大点的地方吧。当然也可能是后面的参数的问题，建议变参对应一定要细心，而打印字符串时，尽量使用”%.ns”的形式指定最大字符数。

忘记了第一个参数

低级得不能再低级问题，用printf 用得太惯了。//偶就常犯。：。（

变参对应出问题

通常是忘记了提供对应某个格式符的变参，导致以后的参数统统错位，检查检查吧。尤其是对应”\*”的那些参数，都提供了吗？不要把一个整数对应一个”%s”，编译器会觉得你欺她太甚了（编译器是obj 和exe 的妈妈，应该是个女的，:P）。

sprintf\_s和snprintf

sprintf\_s()是sprintf()的安全版本，通过指定缓冲区长度来避免sprintf()存在的溢出风险 。在使用VS2008时如果你使用了sprintf函数，那么编译器会发出警告：使用sprintf存在风险，建议使用sprintf\_s。这个安全版本的原型是：

int sprintf\_s(char \*buffer,size\_t sizeOfBuffer,const char \*format [,argument] ... );

不过sprintf\_s()是微软私有的函数，考虑到跨平台移植，最好使用snprintf()。两者的原型基本相同：

int \_snprintf(char \*buffer, size\_t count, const char \*format [,argument] ... );

strftime

spritf 还有个不错的表妹：strftime，专门用于格式化时间字符串的，用法跟她表哥很像，也是一大堆格式控制符，只是毕竟小姑娘家心细，她还要调用者指定缓冲区的最大长度，可能是为了在出现问题时可以推卸责任吧。这里举个例子：

time\_t t = time(0);

//产生"YYYY-MM-DD hh:mm:ss"格式的字符串。

char s[32];

strftime(s, sizeof(s), "%Y-%m-%d %H:%M:%S", localtime(&t));

sprintf 在MFC 中也能找到他的知音：CString::Format，strftime 在MFC 中自然也有她的同道：CTime::Format，这一对由于从面向对象那里得到了赞助，用以写出的代码更觉优雅。