C语言中函数返回字符串的四种方法 - 苍月代表我 - 博客园

苍月代表我关注 - 1 粉丝 - 169 +加关注

在讨论着四种方法之前，首先要对函数有一个简单的认识，无论是在形实结合时，还是在return语句返回时，都有一个拷贝的过程。你传进来的参数是个值，自然函数在工作之前要把这个值拷贝一份供自己使用，你传进来的是个地址，函数也就会拷贝该地址供自己使用。同样return返回时，如果返回一个值，函数会将该值拷贝一份以提供给主调函数使用，返回的是一个指针（也就是地址），自然拷贝的就是一个地址，供主调函数使用。

先给出一个错误的例子：

```
#include <stdio.h>
#include <string.h>

char * retstring();
int main()
{
    char * name2;
    name2 = retstring();
    printf("%s\n",name2);
    return 0;
}

char * retstring()
{
    char name[10];
    strcpy(name,"汉青");
    return name;
}
```

编译一下代码，会发现提示一个警告，大概意思就是说返回了一个局部变量的地址。这个程序的输出结果是不确定的，因为我们都知道，局部变量的生存期是就在块内部，这里也就是在函数retstring()的内部，在main函数中，name的内存空间已经被回收。

所以不能返回一个自动变量的字符串。。。

下面给出四种返回字符串的方法：

1、 将字符串指针作为函数参数传入，并返回该指针。

2、 使用malloc函数动态分配内存，注意在主调函数中释放。

3、 返回一个静态局部变量。

4、 使用全局变量。

下面是详细解释：

方法一：将字符串指针作为函数参数传入，并返回该指针。

典型的strcpy()函数应该就是采用的这种方法，第一个参数为指向目的字符串的指针,返回值也为这个指针。

```
char* strcpy(char* des,const char* source)

{

 char* r=des;

  assert((des != NULL) && (source != NULL));

  while((*r++ = *source++)!='\0');

  return des;

}
```

方法二：使用**malloc**函数动态分配，但是一定要注意在主调函数中将其释放，应为**malloc**动态分配的内存位于堆区，而堆区的内存是要程序员自己释放的。

一个例子如下：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char * retstring();
int main()
{
    char * name2;
    name2 = retstring();
    printf("%s\n",name2);
    //记住一定要用free释放，否则会造成内存泄露
    free(name2);
    return 0;
}
```

```
char * retstring()
{
    char * name;
    name = (char *)malloc(10);
    strcpy(name,"张汉青");
    return name;
}
```

方法三：返回一个静态局部变量。

一个例子如下：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char * retstring();
int main()
{
    char * name2;
    name2 = retstring();
    printf("%s\n",name2);
    return 0;
}

char * retstring()
{
    static char name[10];
    strcpy(name,"张汉青");
    return name;
}
```

这种方法有一个问题： 由于采用了静态局部变量（位于静态区，程序结束时由系统进行释放），这就导致，如果多次调用这个函数，下一次调用会将上一次调用的结果覆盖掉。

C语言中的库函数，tmpnam()函数、getenv()函数等应该都是采用的这种方法，这也就是为什么，使用这样的函数的时候应该立即将返回结果拷贝一份的原因。

方法四： 使用全局变量。

一个例子如下：

```
char   g_s[100];
char*   fun()
{
```

```
        strcpy(g_s,   "abc ");
        return   g_s;
}
```