# C语言编程时，各种类型的变量该如何初始化?

玩转嵌入式 Today

Image在敲代码的时候，我们会给变量一个初始值，以防止因为 编译器 的原因造成变量初始值的不确定性。对于数值类型的变量往往初始化为0，但对于其他类型的变量，如 字符型 、指针型 等变量等该如何初始化呢?

## 数值类变量初始化

整型、浮点型的变量可以在定义的同时进行初始化，一般都初始化为 0 。

```
1  int    inum  = 0;
2  float  fnum = 0.00f;
3  double dnum = 0.00;
```

## 字符型变量初始化

字符型变量也可在定义的同时进行初始化，一般初始化为 '\0' 。

```
1  char ch = '\0';
```

## 字符串初始化

字符串初始化的方法比较多，我这里简单介绍三种，因为字符串本质上是由一个个字符组成的字符数组，所以其初始化的最终目的，就是将字符数组里面的一个个字符都初始化为 '\0' 。

> 方法一:使用空的字符串 "" 。

```
char str[10] = "";
```

```
1
```

> 方法二:使用 memset 。

```
char str[10];
memset(str, 0, sizeof(str));
```

```
1
2
```

> 方法三:写一个循环。

```
char str[10];
for(int i = 0; i < 10; i++)
{
    str[i] = '\0';
}
```

```
1
2
3
4
5
```

这里比较推荐的是第二种初始化方法。也即使用 memset 进行初始化。

> 很多人对 `memset` 这个函数一知半解，只知道它可以初始化很多数据类型的变量，却不知道其原理是什么样的，这里做一下简要的说明： **`memset`** 是按照字节进行填充的。
> 先看下面的一段代码：

```
int num;
memset(&num, 0, sizeof(int));
printf("step1=%d\n", num);
memset(&num, 1, sizeof(int));
printf("step2=%d\n", num);
```

```
1
2
3
4
5
```

> 在讨论之前，我们先看一下运行结果

```
chenyc@DESKTOP-IU8FEL6:~/src$ gcc -o memset memset.c -g
chenyc@DESKTOP-IU8FEL6:~/src$ ./memset
step1 = 0
step2 = 16843009
chenyc@DESKTOP-IU8FEL6:~/src$
```

```
1
2
3
4
```

```
5
```

看到这个运行结果，是不是和你想象中的不一样呢?

`step1 = 0` 相信大家都好理解，可 `step2 = 16843009` 很多人就不能理解了。按照一般的惯性思维，不是应该 `= 1` 才对么?

这就是我要说的，**memset**是按照字节进行填充的。

我们知道， `int` 型是4个字节（每个字节有8位），按二进制表示出来就应该是:

```
00000000 00000000 00000000 00000000
```

```
1
```

按照按字节填充的原则，step1 的结果就是将4个字节全部填充0，所以得到的结果仍然是0:

```
00000000 00000000 00000000 00000000
```

```
1
```

而 step2 则是将每个字节都填充为1 *（注意是每个字节，而不是每个byte位）* ，所以相对应的结果就应该是:

```
00000001 00000001 00000001 00000001
```

```
1
```

> 大家可以自己将上面那个二进制数转换成十进制看看，看看是不是 16843009 。
> 所以严格来说，**memset**函数本身并不具有初始化的功能，而是一个单纯的按字节填充函数，只是人们在使用的过程中，扩展出了初始化的作用。

字符串初始化有一个小窍门，我们知道字符串本质上是字符数组，因此它具有两个特性，

- 字符串在内存里是连续的，
- 字符串遇 '\0' 结束。
  所以我们在初始化的时候，总是愿意给字符串本身长度加1的长度的内存进行初始化。

```
char year[4+1];
memset(year, 0, sizeof(year));
strcpy(year,"2018");
```

## 指针初始化

一般来说，指针都是初始化为 NULL 。

```
int *pnum = NULL;
int num = 0;
pnum = &num;
```

```
2
3
```

指针是个让人又爱又恨的东西，一般的整形、字符串等，初始化之后就可以直接拿来用了，可指针如果初始化为 NULL 后，没有给该指针重新分配内存，则会出现难以预料的错误(最最常见的就是操作空指针引起的段错误)。

在动态内存管理中，由于变量的内存是分配在堆中的，所以一般用 malloc 、 calloc 等函数申请过动态内存，在使用完后需要及时释放，一般释放掉动态内存后要及时将指针置空，这也是很多人容易忽略的。

```
char *p = NULL;
p=(char *)malloc(100);
if(NULL == p)
{
    printf("Memory Allocated at: %x\n",p);
}
else
{
    printf("Not Enough Memory!\n");
}
free(p);
p = NULL;    //这一行给指针置空必不可少，否则很可能后面操作了这个野指针
```

```
1
2
3
4
5
6
```

```
7
8
9
10
11
12
```

很多人经常会犯的一个错误，我们知道，在指针作为实参进行参数传递时，该指针就已经退化成了数组，所以很多人就想到用 `memset` 来对该指针进行初始化：

```
void fun(char *pstr)
{
  memset(pstr, 0, sizeof(pstr));
  ...
}
```

```
1
2
3
4
5
```

这种写法是不正确的。我们姑且不管指针能不能用 `memset` 来进行初始化，指针首先保存的是一个4字节的地址，所以 `sizeof(pstr)` 永远只能 `= 4`，这样的初始化就毫无意义。

## 结构体初始化

结构体的初始化就比较简单了，基本也都是采用 `memset` 的方式。

```
typedef struct student
{
    int id;
    char name[20];
    char sex;
}STU;
STU stu1;
memset((char *)&stu1, 0, sizeof(stu1));
```

```
1
2
3
4
5
6
7
8
```

关于初始化结构体的长度问题，也即 `memset` 的第三个参数，一般来说，传入数据类型和变量名效果是一样的，上例中，下面写法是等价的效果：

```
memset((char *)&stu1, 0, sizeof(STU));
```

```
1
```

但是对于结构体数组的初始化，长度就需要注意一下了，还是以上例来做说明：

```
STU stus[10];
memset((char *)&stus, 0, sizeof(stus)); //正确，数组本身在内存上
```

```
memset((char *)&stus, 0, sizeof(STU));  //错误，只会初始化第一个
memset((char *)&stus, 0, sizeof(STU)*10);  //正确，效果与第一个
```

```
1
2
3
4
```

有些人习惯将 `memset` 的第二个参数写成以下形式：

```
memset((char *)&stu1, 0x00, sizeof(stu1));
```

```
1
```

只要理解了 `memset` 是按字节进行填充的，就知道这样写也是正确的，完全没有问题。

状态机思路在嵌入式开发中的应用

头文件中，#include使用引号“”和尖括号<>有什么区别?

C语言状态机编程思想

7个运算放大电路实例解析，什么是运算放大器？很难但是很实用！

自己编写MODBUS协议代码所踩过的坑

C语言源代码展示：常用转换函数实现原理

文本或代码中 \n 和 \r 的区别

喜欢此内容的人还喜欢

C语言函数不能返回数组，却可以返回结构体?

技术让梦想更伟大

【编程之美】用C语言实现状态机(实用)

技术让梦想更伟大

详细剖析 extern "C"

C语言与CPP编程