

- 一、前言
- 二、小试牛刀
  - 1. 灵活的数组成员
  - 2. 不定参数的宏定义
- 三、为自己打气

## 一、前言

这几天在把一个嵌入式项目的代码，移植到另一个平台，发现很多地方用的都是 C89 标准。

1999 年，C 语言的标准化委员会发布了 C99 标准，引入了许多特性，包括可变长度的数组、灵活的数组成员（用在结构体）、对 IEEE754 浮点数的改进、指定成员的初始化器、内联函数、支持不定参数个数的宏定义，在数据类型上还增加了 long long int 以及复数类型。

于是最近找了一本比较新的 C 语言书籍翻了一下，发现很多比较偏僻的语法，很少被使用到，包括 C99 标准中的一些内容，所以我想把这部分内容整理一下，也是让自己对这一门古老的语言重新梳理一下。

## 二、小试牛刀

### 1. 灵活的数组成员

先不解释概念，我们先来看一个代码示例：

```
// 一个结构体，成员变量 data 是指针
typedef struct _Data1_ {
    int num;
    char *data;
} Data1;

void demo6_not_good()
{
    // 打印结构体的内存大小
    int size = sizeof(Data1);
    printf("size = %d \n", size);

    // 分配一个结构体指针
    Data1 *ams = (Data1 *)malloc(size);
    ams->num = 1;

    // 为结构体中的 data 指针分配空间
    ams->data = (char *)malloc(1024);
    strcpy(ams->data, "hello");
    printf("ams->data = %s \n", ams->data);

    // 打印结构体指针、成员变量的地址
    printf("ams = 0x%x \n", ams);
    printf("ams->num = 0x%x \n", &ams->num);
    printf("ams->data = 0x%x \n", ams->data);

    // 释放空间
    free(ams->data);
    free(ams);
}
```

在我的电脑上，打印结果如下：

```
size = 8
ams->data = hello
ams = 0x9b03410
ams->num   = 0x9b03410
ams->data = 0x9b03420
```

可以看到：该结构体一共有 8 个字节(int 型占 4 个字节，指针型占 4 个字节)。

结构体中的 data 成员是一个指针变量，需要单独为它申请一块空间才可以使用。而且在结构体使用之后，需要先释放 data，然后释放结构体指针 ams，顺序不能错。这样使用起来，是不是有点麻烦？

于是，C99 标准就定义了一个语法：flexible array member(柔性数组)，直接上代码(下面的代码如果编译时遇到警告，请检查下编译器对这个语法的支持)：

```
// 一个结构体，成员变量是未指明大小的数组
typedef struct _Data2_ {
    int num;
    char data[];
} Data2;

void demo6_good()
{
    // 打印结构体的大小
    int size = sizeof(Data2);
    printf("size = %d \n", size);

    // 为结构体指针分配空间
    Data2 *ams = (Data2 *)malloc(size + 1024);

    strcpy(ams->data, "hello");
    printf("ams->data = %s \n", ams->data);

    // 打印结构体指针、成员变量的地址
    printf("ams = 0x%x \n", ams);
    printf("ams->num   = 0x%x \n", &ams->num);
    printf("ams->data = 0x%x \n", ams->data);

    // 释放空间
    free(ams);
}
```

打印结果如下：

```
size = 4
ams->data = hello
ams = 0x9b03410
ams->num   = 0x9b03410
ams->data  = 0x9b03414
```

与第一个例子中有下面几个不同点：

1. 结构体的大小变成了 4；
2. 为结构体指针分配空间时，除了结构体本身的大小外，还申请了 data 需要的空间大小；
3. 不需要为 data 单独分配空间了；
4. 释放空间时，直接释放结构体指针即可；

是不是用起来简单多了？！这就是柔性数组的好处。

从语法上来说，柔性数组就是指结构体中最后一个元素个数未知的数组，也可以理解为长度为 0，那么就可以让这个结构体称为可变长的。

前面说过，数组名就代表一个地址，是一个不变的地址常量。在结构体中，数组名仅仅是一个符号而已，只代表一个偏移量，不会占用具体的空间。

另外，柔性数组可以是任意类型。这里示例大家多多体会，在很多通讯类的处理场景中，常常见到这种用法。

## 2. 不定参数的宏定义

宏定义参数个数可以是不确定的，就像调用 printf 打印函数一样，在定义的时候，可以使用三个点(...)来表示可变参数，也可以在三个点的前面加上可变参数的名称。

如果使用三个点(...)来接收可变参数，那么在使用的时候就需要使用 VA\_ARGS 来表示可变参数，如下：

```
#define debug1(...)    printf(__VA_ARGS__)

debug1("this is debug1: %d \n", 1);
```

如果在三个点(...)的前面加上了一个参数名，那么在使用时就一定要使用这个参数名，而不能使用 VA\_ARGS 来表示可变参数，如下：

```
#define debug2(args...) printf(args)

debug2("this is debug2: %d \n", 2);
```

但是，如果可变参数个数为零时，处理可能就会出问题！

看一下这个宏：

```
#define debug3(format, ...)    printf(format, __VA_ARGS__)

debug3("this is debug4: %d \n", 4);
```

编译、执行都没有问题。但是如果这样来使用宏：

```
debug3("hello \n");
```

编译的时候，会出现错误：error: expected expression before ‘)’ token。为什么呢？

看一下宏扩展之后的代码（\_\_VA\_ARGS\_\_为空）：

```
printf("hello \n",);
```

看出问题了吧？在格式化字符串的后面多了一个逗号！为了解决问题，预处理器给我们提供了一个方法：通过##符号把这个多余的逗号给自动删掉。

于是宏定义改成下面这样就没有问题了。

```
#define debug3(format, ...)    printf(format, ##__VA_ARGS__)
```

类似的，如果自己定义了可变参数的名字，也在前面加上##，如下：

```
#define debug4(format, args...) printf(format, ##args)
```

### 三、为自己打气

这篇文章仅仅是开篇，后续我会继续搜集资料，最终目标是想把 C 语言中的语法和使用，汇总成一个小册子，希望自己能坚持下去！

---

好文章，要转发；越分享，越幸运！

星标公众号，能更快找到我！

---



---

如果您的网站想转载这篇文章，只要保留作者、文章来源和上图二维码即可。

如果您的公众号想转载这篇文章，请私信或留言，我给您开长白。

---

## 推荐阅读

### 【C 语言】

[C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻](#)

[原来gdb的底层调试原理这么简单](#)

[一步步分析-如何用C实现面向对象编程](#)

[提高代码逼格的利器：宏定义-从入门到放弃](#)

[利用C语言中的setjmp和longjmp，来实现异常捕获和协程](#)

### 【应用程序设计】

[都说软件架构要分层、分模块，具体应该怎么做\(一\)](#)

[都说软件架构要分层、分模块，具体应该怎么做\(二\)](#)

[物联网网关开发：基于MQTT消息总线的设计过程\(上\)](#)

[物联网网关开发：基于MQTT消息总线的设计过程\(下\)](#)

[我最喜欢的进程之间通信方式-消息总线](#)

### 【操作系统】

[为什么航天器、导弹喜欢用单片机，而不是嵌入式系统？](#)

### 【物联网】

[关于加密、证书的那些事](#)

[深入LUA脚本语言，让你彻底明白调试原理](#)

### 【胡说八道】

[以我失败的职业经历：给初入职场的技术人员几个小建议](#)