在日常的编程中，有时候需要在结构体中存放一个长度动态的字符串，一般的做法，是在结构体中定义一个指针成员，这个指针成员指向该字符串所在的动态内存空间，例如：

```
typedef struct test
{
int a;
double b;
char *p;
};
```

p指向字符串。这种方法造成字符串与结构体是分离的，不利于操作。如果把字符串跟结构体直接连在一起，不是更好吗？于是，可以把代码修改为这样：

```
char a[] = "hello world";
test *stpTest = (test *)malloc(sizeof(test) + strlen( a ) + 1 );
strcpy(stpTest + 1, a );
```

这样一来，(char*)(stpTest + 1)就是字符串"hello world"的地址了。这时候p成了多余的东西，可以去掉。但是，又产生了另外一个问题：老是使用(char*)((stpTest + 1)不方便。如果能够找出一种方法，既能直接引用该字符串，又不占用结构体的空间，就完美了，符合这种条件的代码结构应该是一个非对象的符号地址，在结构体的尾部放置一个0长度的数组是一个绝妙的解决方案。不过，C/C++标准规定不能定义长度为0的数组，因此，有些编译器就把0长度的数组成员作为自己的非标准扩展。

在讲述柔性数组成员之前，首先要介绍一下不完整类型(incomplete type)。不完整类型是这样一种类型，它缺乏足够的信息例如长度去描述一个完整的对象，

它的出现反映了C程序员对精炼代码的极致追求，这种代码结构产生于对动态结构体的需求。

鉴于这种代码结构所产生的重要作用，C99甚至把它收入了标准中。C99使用不完整类型实现柔性数组成员，在C99 中，结构中的最后一个元素允许是未知大小的数组，这就叫做柔性数组(flexible array)成员(也叫伸缩性数组成员)，但结构中的柔性数组成员前面必须至少一个其他成员。柔性数组成员允许结构中包含一个大小可变的数组。柔性数组成员只作为一个符号地址存在，而且必须是结构体的最后一个成员，sizeof 返回的这种结构大小不包括柔性数组的内存。柔性数组成员不仅可以用于字符数组，还可以是元素为其它类型的数组。包含柔性数组成员的结构用malloc ()函数进行内存的动态分配，并且分配的内存应该大于结构的大小，以适应柔性数组的预期大小。柔性数组的使用请看下面的例子：

```
typedef struct test
{
int a;
double b;
char c[0];
};
```

有些编译器会报错无法编译可以改成：

```
typedef struct test
{
int a;
double b;
char c[];
```

```
6       };
```

通过如下表达式给结构体分配内存：

```
1       test *stpTest = (test *)malloc(sizeof(test)+100*sizeof(char));
```

c就是一个柔性数组成员，如果把stpTest指向的动态分配内存看作一个整体，c就是一个长度可以动态变化的结构体成员，柔性一词来源于此。c的长度为0，因此它不占用test的空间，同时stpTest->c就是“hello world”的首地址，不需要再使用(char *)(stpTest + 1)这么丑陋的代码了。那个0个元素的数组没有占用空间，而后我们可以进行变长操作了。这样我们为结构体指针c分配了一块内存。用stpTest->c[n]就能简单地访问可变长元素。

当然，上面既然用malloc 函数分配了内存，肯定就需要用free 函数来释放内存：

```
1       free(stpTest);
```

应当尽量使用标准形式，在非C99的场合，可以使用指针方法。需要说明的是：C89不支持这种东西，C99把它作为一种特例加入了标准。但是，C99所支持的是incomplete type，而不是zero array，形同int a[0];这种形式是非法的，C99 支持的形式是形同int a[];只不过有些编译器把int a[0];作为非标准扩展来支持，而且在C99 发布之前已经有了这种非标准扩展了，C99 发布之后，有些编译器把两者合而为一了。

说明：

```
struct test
{
    int i;
    int c;
    char a[];
};
```

重点： a 不占内存，c 的变量名就是内存地址， sizeof（struct test） 为 8，编译器内存对齐时 a 不占内存，后续动态分配的时候 程序手动分配