(2条消息) C中宏定义#和##_LtMamba的博客-CSDN博客_c宏定义##

在宏定义中：

1. #的用法：

#后只能跟参数，是把跟在后面的参数转换成一个字符串的作用。

例如：

```
    > #define  FOO(arg)   my##arg
则
    > FOO(abc)
相当于   myabc
```

例如：

```
    > #define STRCPY(dst, src)   strcpy(dst, #src)
则
    > STRCPY(buff, abc)
相当于   strcpy(buff, "abc")
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11

2. ##的用法

①. 用法一

如果##后的参数本身也是一个宏的话，##会阻止这个宏的展开，也就是只替换一次。

```
#define STRCPY(a, b)    strcpy(a ## _p, #b)

int main()

{
    char var1_p[20];

    char var2_p[30];



    STRCPY(STRCPY(var1,var2),var2);



}
```

- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27

②. 用法二：

“##”是一种分隔连接方式，它的作用是先分隔，然后进行强制连接。

另外一些分隔标志是，包括操作符，比如 +, -, *, /, [,], …，所以尽管下例的宏定义没有空格，但是依然表达有意义的定义：

```
#define add(a, b)  a + b
```

- 1

上面宏定义中“+”分隔符强制连接的作用是去掉和前面的字符串之间的空格，而把两者连接起来。
而##也拥有和上面的“+”类似的作用。

举列 – 试比较下述几个宏定义的区别：

```
#define A1(name, type)  type name_##type##_type 或

#define A2(name, type)  type name##_##type##_type

A1(a1, int);

A2(a1, int);
```

解释:

1）在第一个宏定义中，"name"和第一个"_"之间，以及第2个"_"和第二个"type"之间没有被分隔，所以预处理器会把name_##type##_type解释成3段：

“name_”、“type”、以及“_type”，这中间只有“type”是在宏前面出现过

的，所以它可以被宏替换。

2）而在第二个宏定义中，“name”和第一个“_”之间也被分隔了，所以

预处理器会把name##_##type##_type解释成4段：“name”、“_”、“type”

以及“_type”，这其间，就有两个可以被宏替换了。

3）A1和A2的定义也可以如下：

```
#define A1(name, type)  type name_  ##type ##_type
```

<##前面随意加上一些空格>

```
#define A2(name, type)  type name ##_ ##type ##_type
```

结果是## 会把前面的空格去掉完成强连接，得到和上面结果相同的宏定义

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16

- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31