K	类型
1	char
2	short
4	int,float
8	long,double,char*

确保每种数据类型都是按照指定方式来组织和分配，即每种类型的对象都满足它的对齐限制，就可保证实施对齐。编译器在汇编代码中放入命令，指明全局数据所需的对齐。例如，3.6.8 节开始的跳转表的汇编代码声明在第 2 行包含下面这样的命令：

```
.align 8
```

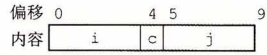
这就保证了它后面的数据(在此，是跳转表的开始)的起始地址是 8 的倍数。因为每个表项长 8 个字节，后面的元素都会遵守 8 字节对齐的限制。

对于包含结构的代码，编译器可能需要在字段的分配中插入间隙，以保证每个结构元素都满足它的对齐要求。而结构本身对它的起始地址也有一些对齐要求。

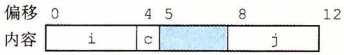
比如说，考虑下面的结构声明：

```
struct S1 {
    int  i;
    char c;
    int  j;
};
```

假设编译器用最小的 9 字节分配，画出图来是这样的：



它是不可能满足字段 i(偏移为 0)和 j(偏移为 5)的 4 字节对齐要求的。取而代之地，编译器在字段 c 和 j 之间插入一个 3 字节的间隙(在此用蓝色阴影表示)：



结果，j 的偏移量为 8，而整个结构的大小为 12 字节。此外，编译器必须保证任何 struct S1 * 类型的指针 p 都满足 4 字节对齐。用我们前面的符号，设指针 p 的值为 x_p。那么，x_p必须是 4 的倍数。这就保证了 p-> i(地址 x_p)和 p-> j(地址 x_p+8)都满足它们的 4 字节对齐要求。

另外，编译器结构的末尾可能需要一些填充，这样结构数组中的每个元素都会满足它的对齐要求。例如，考虑下面这个结构声明：

```
struct S2 {
    int  i;
    int  j;
    char c;
};
```

如果我们将这个结构打包成 9 个字节，只要保证结构的起始地址满足 4 字节对齐要求，我们仍然能够保证满足字段 i 和 j 的对齐要求。不过，考虑下面的声明：

```
struct S2 d[4];
```