《传智播客 C语言就业班》 第三讲 结构体

野指针：指向一个已删除的对象或未申请访问受限内存区域的指针（指针指向一个内存空间，不能往里面存数据，也不能释放）。

free释放指针所指的内存空间之后，还要注意将指针变量本身重置为NULL，如下。否则会产生野指针。

```
strcpy(p1, "1112222");
printf("p1:%s\n",p1);

if(p1 != NULL)
{
   free(p1);
   P1 = NULL;
}
```

避免野指针的方法：
1）定义指针的时候初始化成NULL 2）释放内存的时候先判断指针是不是NULL 3）free释放指针所指向的内存空间后，把指针重置成NULL。

野指针和指针做函数参数在一起，如下：（**这个程序异常重要！**）

```
char *getMem2(int count)
{
   char *tmp = NULL;
   tmp = (char*)malloc(100*sizeof(char)); //char tmp[100]
   return tmp;
}

//错误写法
int FreeMem2(char *p)
{
   if(p == NULL)
   {
      return -1;
   }
   if(p != NULL)
   {
      free(p);
      p = NULL; //想把实参给改掉，你能修改吗？修改不了实参。。
   }
   return 0;
}

void main()
{
   char *myp = NULL;
   myp = getMem2(100);
   FreeMem2(myp);
}
```

注：
虽然free(p)可以释放掉形参所指向的内存空间，但是P=NULL和实参没有任何关系，从而产生野指针。解决办法：对nyp取地址再传过去，变为二级指针才能修改实参myp的值！
1、2级指针做输入——主调函数分配内存；

1、2级指针做输出——被调函数分配内存；

函数指针做函数参数（指针的另外半壁江山！）

- 正向调用
- 反向调用（回调函数）

……暂略，C++再讲

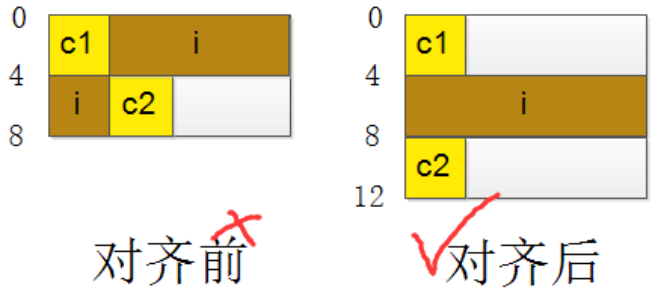
结构体专题

```
//定义了一个数据类型，没有分配内存
struct Teacher
{
	char name[62];
	int age;
}t2, t3;  //可以直接定义变量t2和t3，但一般不用
void main()
{
	struct Teacher t1; //告诉C++编译器分配内存，在临时区
			//捆绑分配，捆绑释放
	struct Teacher *p = (struct Teacher*) malloc(sizeof(struct Teacher));
	free(p);
	printf("%d\n", sizeof(struct Teacher)); //输出为68,说明内存4字节对齐；
			//约定CPU以4字节为单位寻址，速度快
	system("pause");
}
```

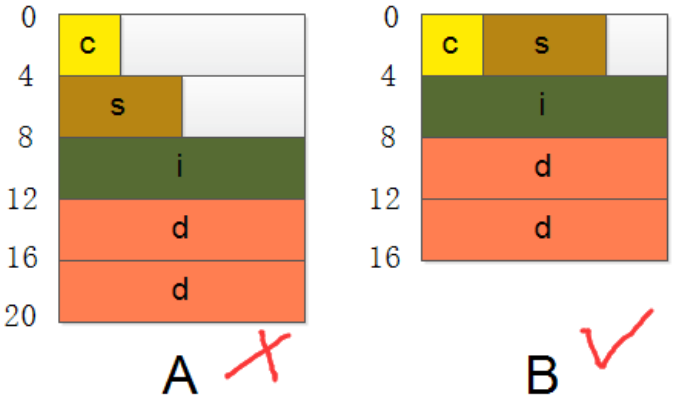
另一种结构体定义方式——typedef重命名；定义后，之前的struct Teacher类型可以省略struct关键字

```
typedef struct _Teacher
{
	char name[64];
	int age;
}Teacher;
```

4字节对齐：https://blog.csdn.net/yilese/article/details/76199869?locationNum=9&fps=1



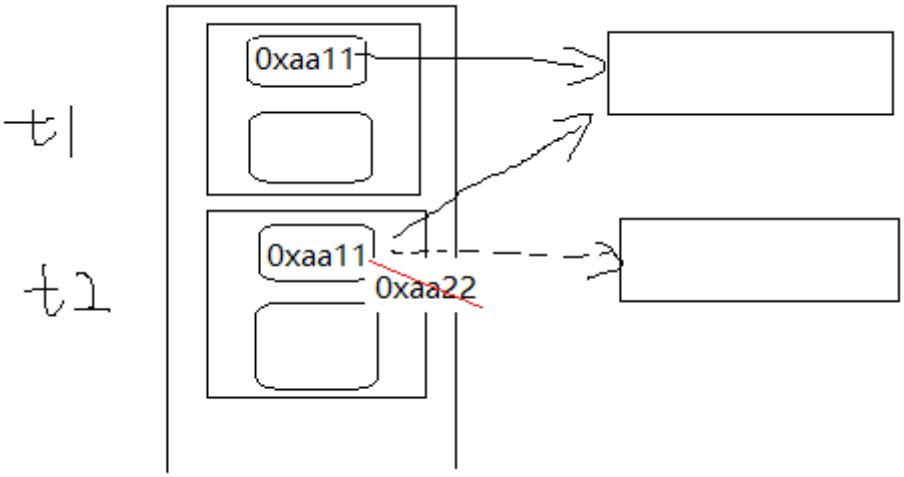
下图，编译器采用B方案：



编译器浅copy操作

```
typedef struct Teacher
{
	char* name;
	int age;
}Teacher;
//编译器给我们提供的copy行为是一个浅copy
//当结构体成员域中含有buf的时候ok
//当结构体成员域中还有指针的时候，C++编译器只会进行指针变量的copy，指针变量所指的内存空间不会再多分配内存
void main()
{
	Teacher t1;
	Teacher t2;
	t1.name = (char*)malloc(100);
	t1.age = 10;
	t2 = t1;
	if (t1.name != NULL)
	{
		free(t1.name);
	}
	if (t2.name != NULL)
	{
		free(t2.name);
	}
	system("pause");
}
```

浅拷贝与深拷贝



如果要深拷贝，可以自写如下函数：

```
int copyObj(Teacher *to, Teacher *from)
{
```

```
    memcpy(to, from, sizeof(Teacher));  //等价于*to = *from
    to->name = (char*)malloc(100);
    strcpy(to->name, from->name);
}
```

- **结构体中的高级话题 —— 成员域偏移量**

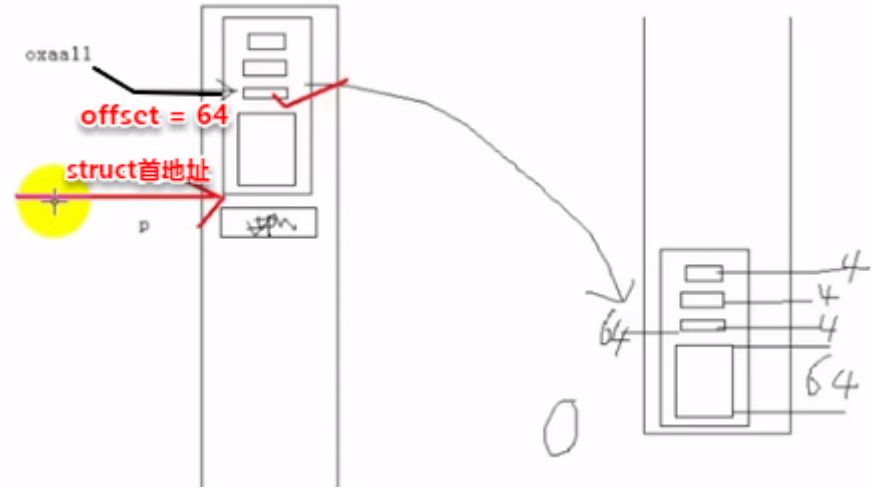
```
typedef struct Teacher
{
    char name[64]; //64
    int age;  //4
    int p;    //4
    char *pname;
}Teacher;
```

```
void main()
{
   Teacher *p = NULL;
   p = p - 1;
   p = p - 2;
   p = p - p;
   &(p->age);  //1 逻辑计算在CPU中，没有读写内存，所以不会宕掉
}
```

如果执行如下代码，输出 i:64

```
i = (int)(&(p->age)); //逻辑计算在CPU中
printf("i:%d\n", i);
```

或者写成int offset = (int)&(((Teacher *)0)->age)，相当于把这个类型映射到地址为0的内存空间中，即按照Teacher这种struct类型来解释这一块内存空间。



根据偏移量64就可以求出结构体的位置。

注意：轻易不要改变struct中数据成员的顺序。就好比如果发布一个头文件和动态库，如果改了头文件就和动态库不匹配了，必须要重新编译，原因就是偏移量不一样了。

注意：.和->的本质是CPU寻址，不操作内存。

```
void main()
{
    Teacher t1;
    Teacher t2;
    Teacher *p = NULL;
    printf("%d\n", sizeof(Teacher));
    p = &t1;
    strcpy(t1.name, "name");

    t1.age = 10;
    p->age = 12;
    p->age; //. ->的本质是寻址，寻每一个成员变量相对于大变量t1的内存偏移，并没有操作内存，所以这样写没有问题。

    t2 = t1;  //编译器做了什么工作

    printf("t2.age:%d\n", t2.age);
}
```