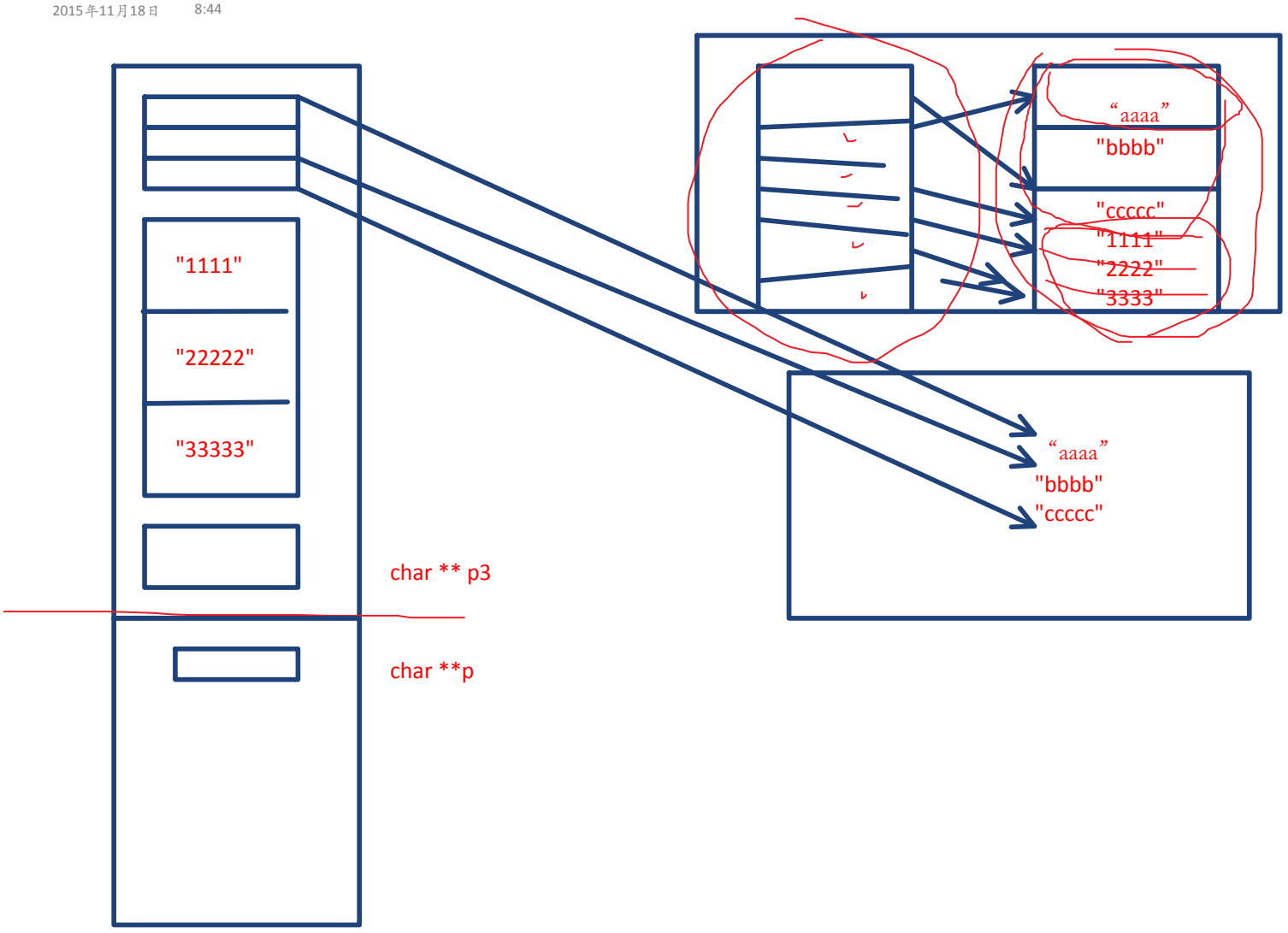
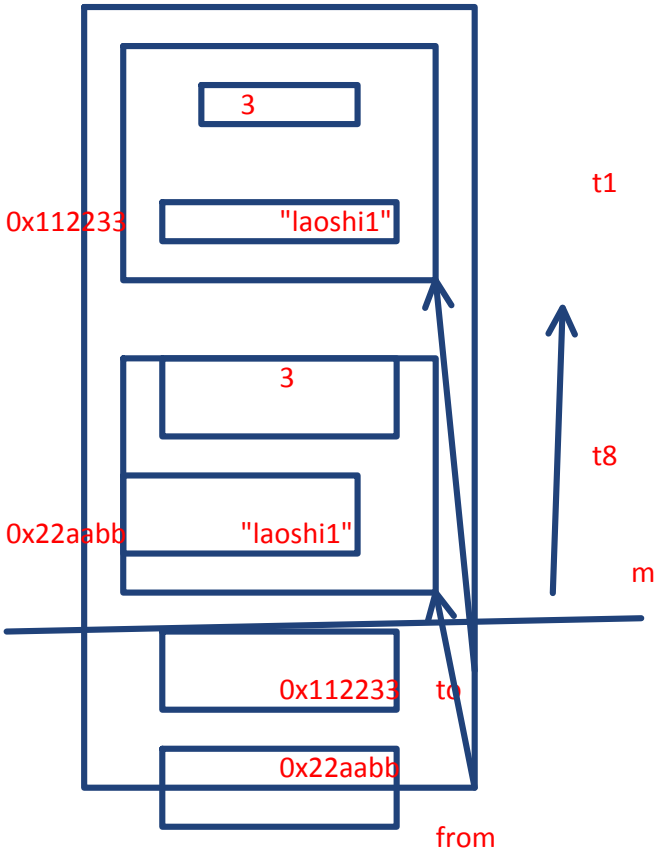
1 作业题

2015年11月18日 8:44



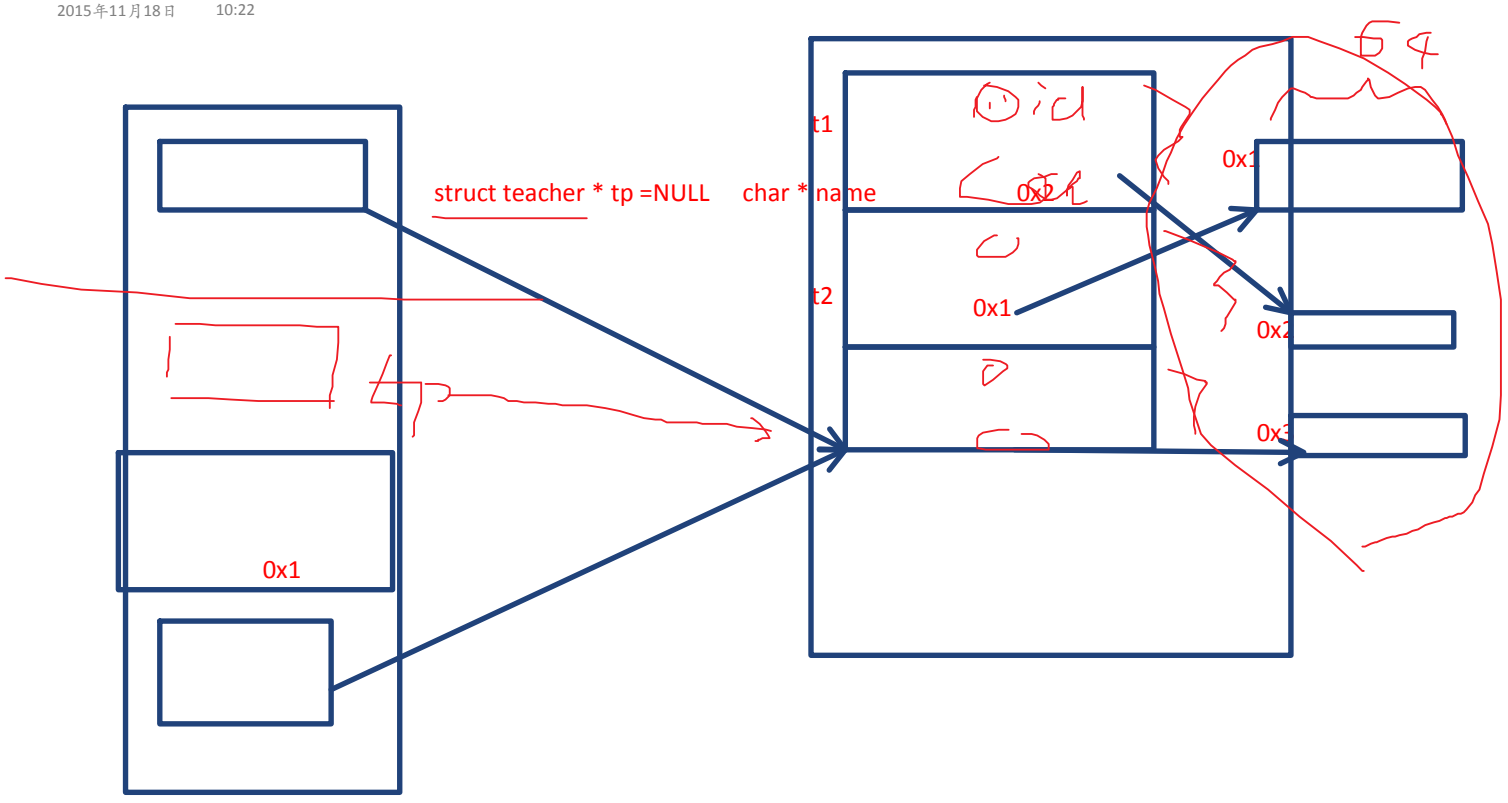
2 结构体赋值

2015年11月18日 10:06



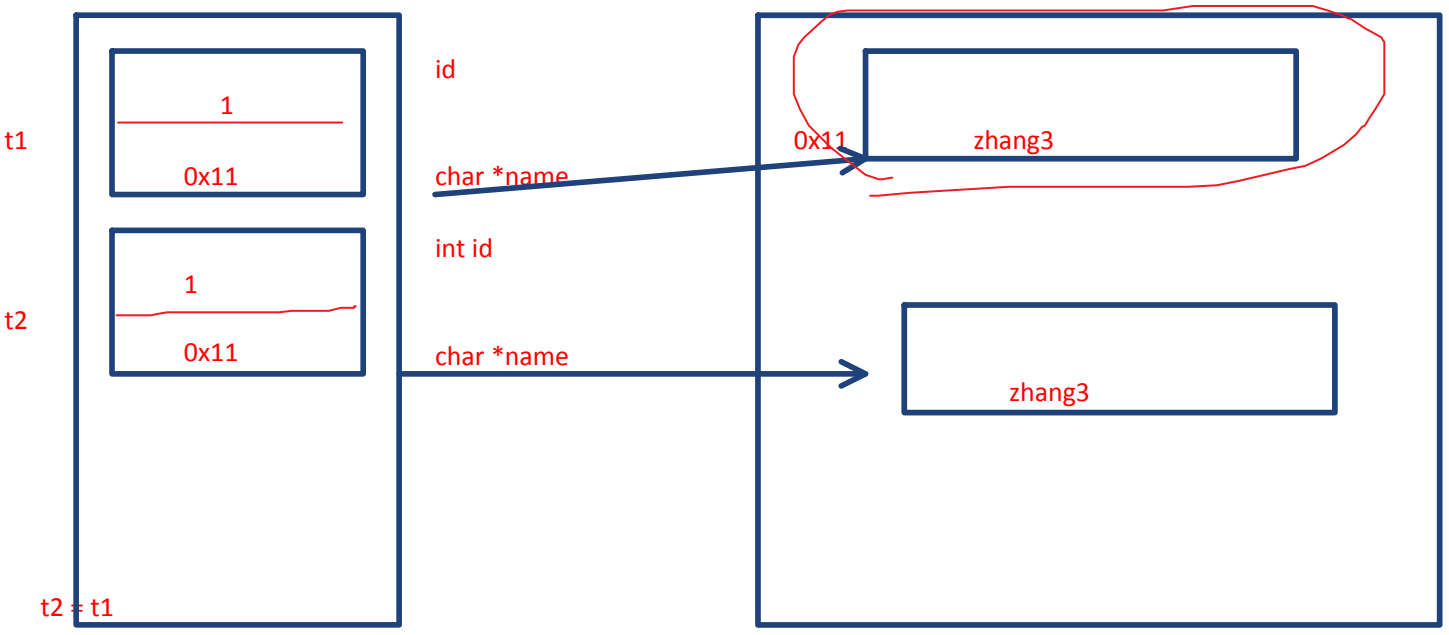
3 结构体作为函数参数

2015年11月18日 10:22



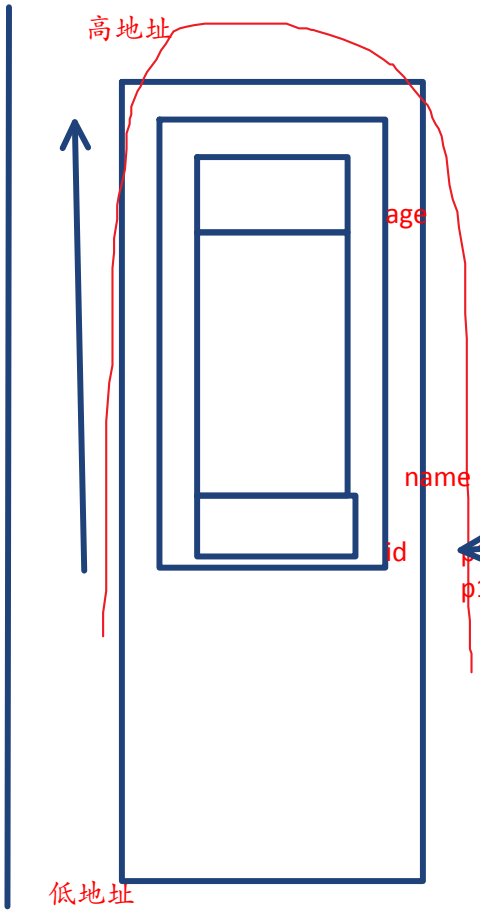
4 结构体的深拷贝和浅拷贝

2015年11月18日 11:31



5 结构体成员的偏移

2015年11月18日 11:45




```
strcut teacher
{
    int id;
    char name[64];
    int age;
};
```

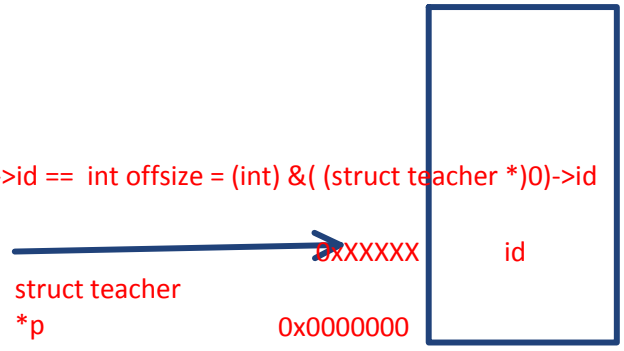
.-> 是不是

```
struct teacher t1;
struct teacher *p1 = &t1

    t1.id  == p1->id;
    t1.name=== p1->name    （p1+sizeof(id)）
    t1.age = == p1->age (p1+sizeof(id) + sizeof(name))
```

p1->id
p1->name

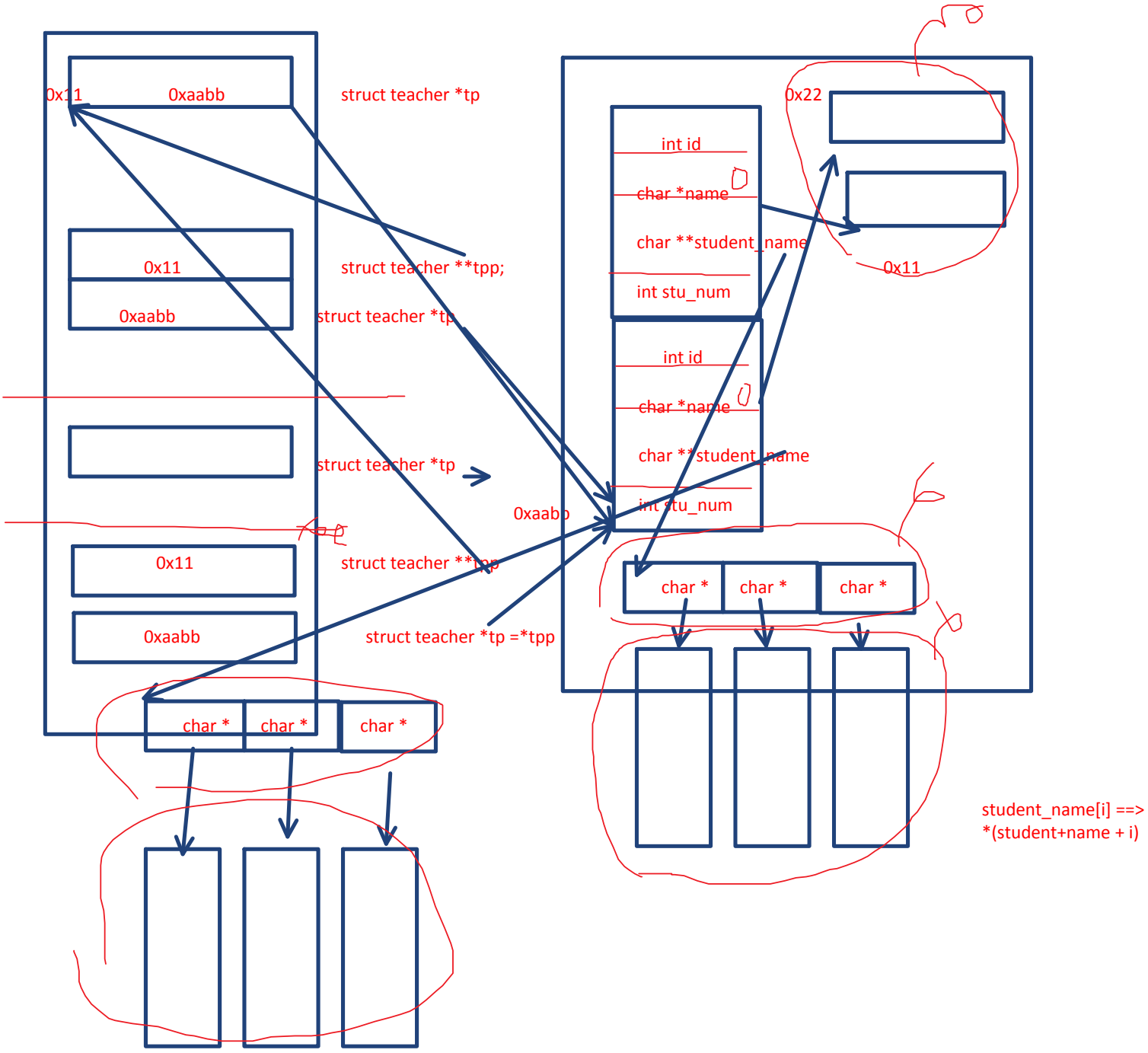
p->id == int offsize = (int) &((struct teacher *)0)->id



struct teacher
*p

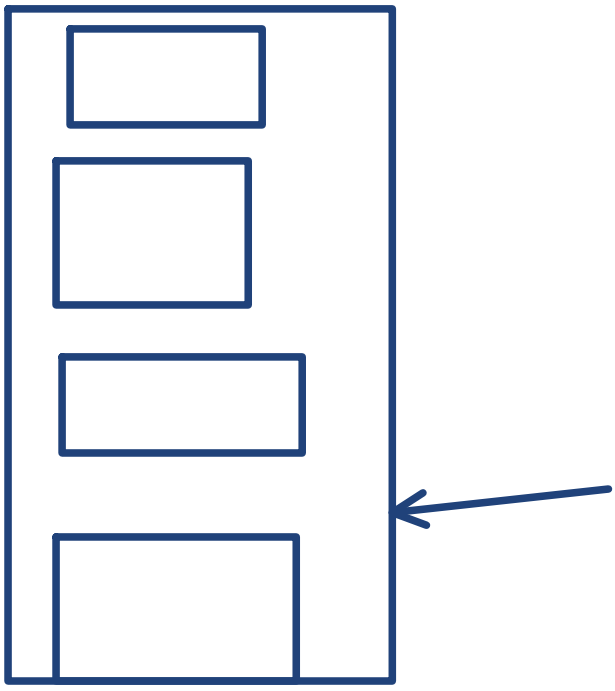
6 结构体嵌套二级指针

2015年11月18日 14:49



7

2015年11月18日 16:00

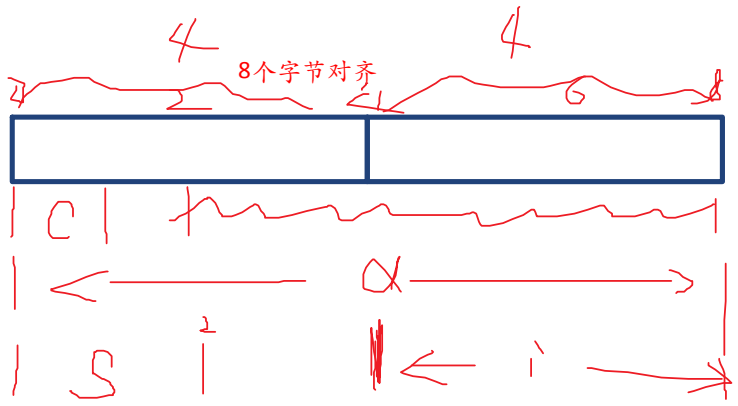


id.
p->

8 结构体字节对齐方式

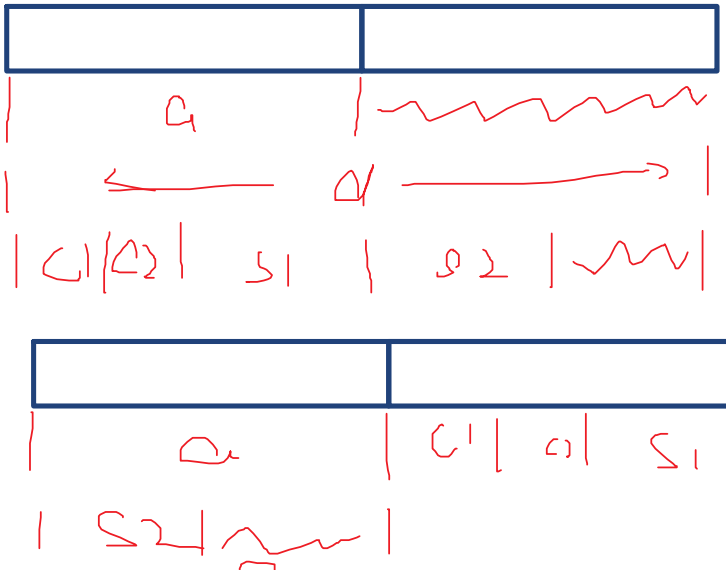
2015年11月18日 16:05

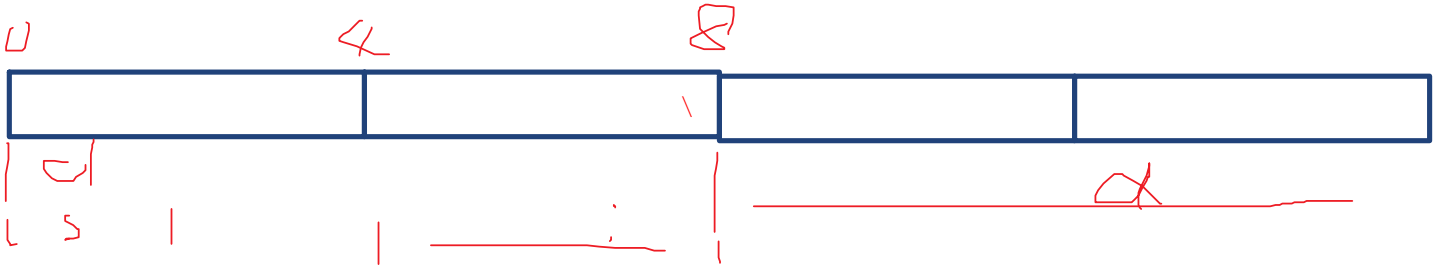
```
struct A
{
    char        c;          //1byte
    double      d;          //8byte
    short       s;          //2byte
    int         i;          //4byte
};
```



1 字节对齐，每行的最大长度是字节对齐的长度
如果一个数据类型放不下，应该另起一行来存放
2. 每个数据类型必须放在能够整除自己大小的地址上
3 最终结构体大小一定是 里面成员最大数据类型大小的 最小公倍数

```
struct B
{
    int a;
    //double d;
    char c1;
    char c2;
    short s1;
    short s2;
};
```

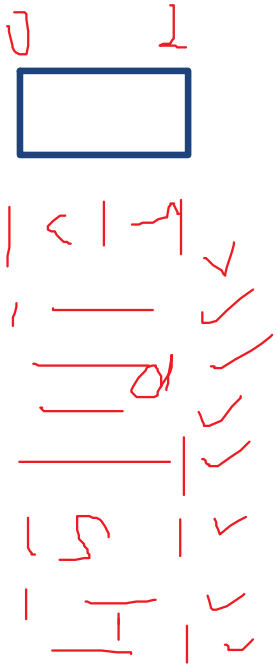


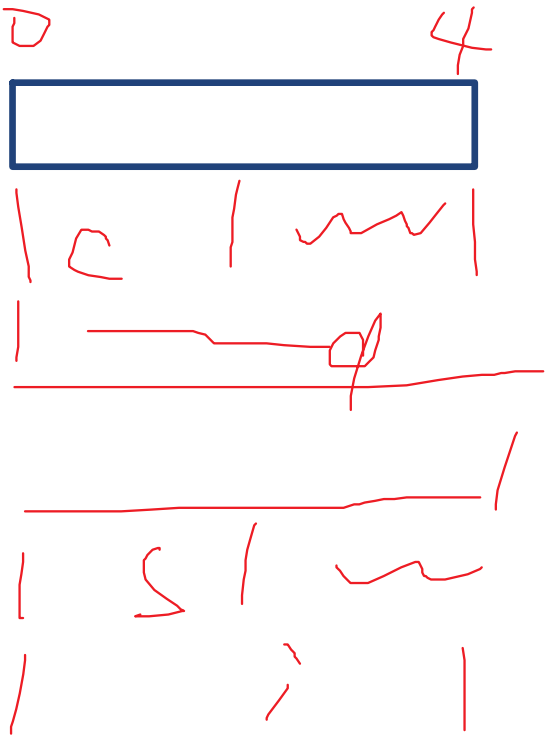


9 结构体字节对齐

2015年11月18日　　16:19

```
struct A
{
    char        c;          //1byte
    double      d;          //8byte
    short       s;          //2byte
    int         i;          //4byte
};
```



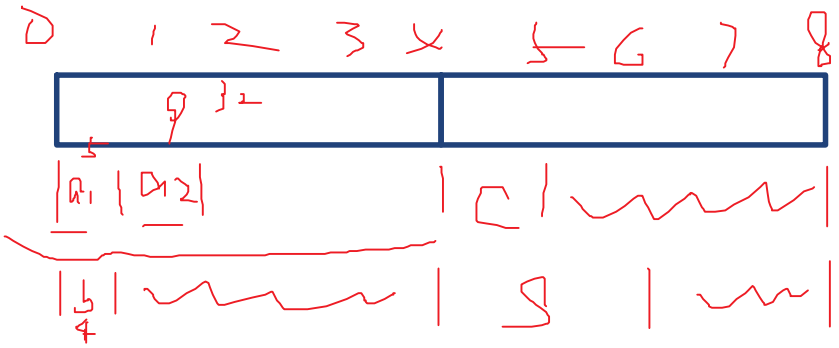


10 不完整字节对齐

2015年11月18日 16:37

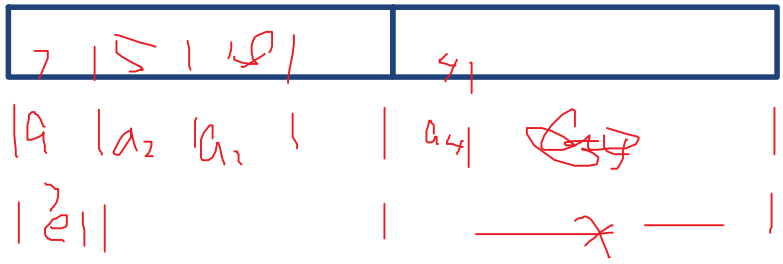
```
struct A
{
    int     a1 : 5;
    int     a2 : 9;
    char    c;
    int     b : 4;
    short   s;
};
```

8字节



1. 不完整类型字节对齐，如果是相同类型相邻的元素，如果之前的空间依然可以存放，在之前的空间直接存放
2. 如果该数据类型不和上一个成员的数据类型一样 需要另开辟空间

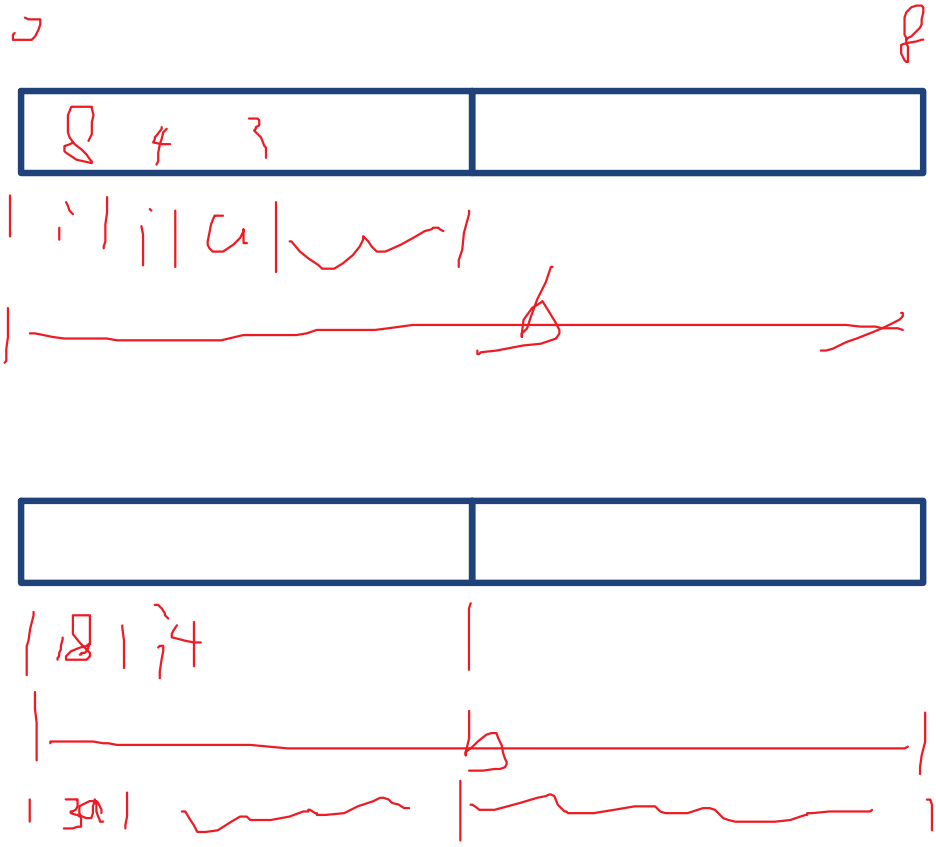
```
struct B
{
    int a : 7;
    int a2 : 5;
    int a3 : 18;
    int a4 : 4;
    char e : 7;
    int x;
};
```



11 字节序练习

2015年11月18日 16:49

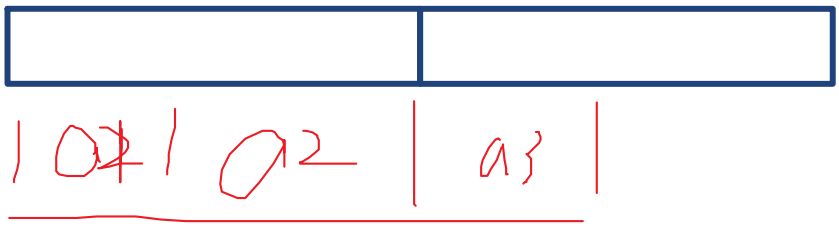
```
struct s1
{
int i:8;
int j:4;
int a:3;
double b;
}
struct s2
{
int i:8;
int j:4;
double b;
int a:3;
}
```



12

2015年11月18日 16:51

```
struct{
short a1;
short a2;
short a3;
}A;
```



```
struct{
long a1;
short a2;
}B;
```

13 位运算

2015年11月18日 16:54

& 与

0000 0011
0001 0101&
0000 0001

& 运算 位数依次对比 如果有0 则为0
全是1 就是1

| 或

0000 0011
0001 0101 |
0001 0111

| 运算 位数依次对比 如果有1 则为1
全是0 就是0

^ 抑或

0000 0011
0001 0101
0001 0110

^ 运算　　如果相同为0，相异为1

14 位移运算

2015年11月18日 17:03

int a = 1; 0000 0000 0000 0000 0000 0000 0000 0001
a <<2 ; 0000 0000 0000 0000 0000 0000 0000 0100

对于左移运算符

不管是有符号类型还是无符号类型
只要是左移，右边就补0

-127 ~ 128

char a; 0000 0000
unsigned char a; 0000 0000 0 ~ 255

对于右移操作符，

int a =1; 0000 0000 0000 0000 0000 0000 0000 0001

如果是有符号的，一般的编译器，如果符号位是1
那么右移就补齐1，如果符号位是0 那么就补齐0

但是有的编译器，会无条件补0

对于右移操作符，不要针对 有符号的数据类型操作

15 掩码

2015年11月18日　　17:19

a			1111 1111 1111 1111 1111 1111 1111 1111

mask　　0000 0000 0000 0000 0000 1111 0000 0000
~mask　　1111 1111 1111 1111 1111 0000 1111 1111
a &(~mask)　　1111 1111 1111 1111 1111 0000 1111 1111

16 子网掩码

2015年11月18日 17:21

192 168 100 14

192.168.100.14 0010 1100 0110 1100 0100 0111 0000 1110
255.255.255.0 1111 1111 1111 1111 1111 1111 0000 0000

x = 0xff 30 ff 30
int value;

```
unsigned int getBit(unsigned int x, int pos, int n);
```

getBit(x, 8, 4)-->f
getBit(x, 16,8)-->30

192.168.100.26 100M 192.168.100.0
192.168.100.14 300M 192.168.100.0

1010110101010101 0101 0110 10101010
0000000000000000 1111 1111 00000000 &

0000000000000000 0101 0110 00000000 a&mask
(a&mask)>>8
00000000000000000000000000000000 0101 0110

a = 12345678;
mask = 0000ff00;

统计100网段的所有流量总和