知识点1【typedef 给已有的类型取个别名】（了解）0-1

案例：给int取个别名INT32

```
//typedef使用步骤：
//1、先用已有的类型 定义一个变量
//2、用别名 替换 变量名
//3、在整个表达式前 添加typedef

```

```c
//注意:不能创造新的类型
typedef int INT32;
void test01()
{
 INT32 num = 10;
 printf("num = %d\n", num);//10
}
```

案例：给数组取个别名

案例：给指针取别名

```c
typedef int *P_TYPE;
void test03()
{
 int num = 10;
 P_TYPE p = &num;//P_TYPE p == int *p
 printf("*p = %d\n", *p);//10
}
```

案例：给函数指针 取别名

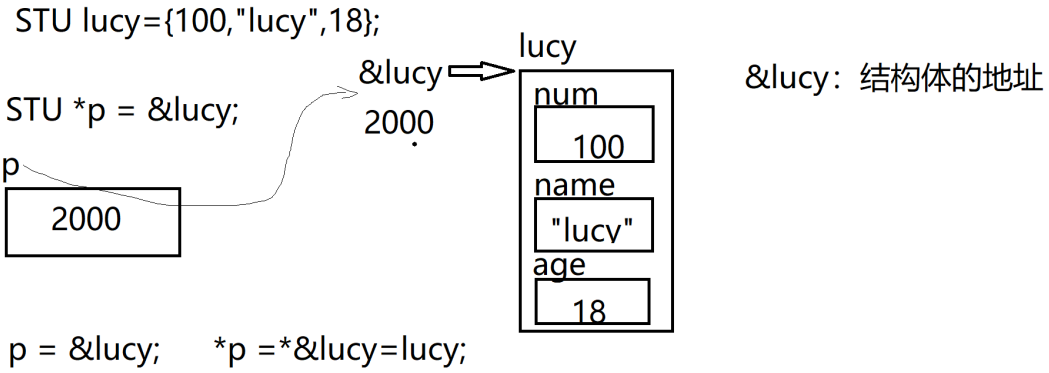
```c
int my_add(int x,int y)
{
 return x+y;
}
//FUN_P 是一个函数指针类型 该函数 必须有两个int形参 以及一个int返回值
typedef int (*FUN_P)(int x,int y);
void test04()
{
 FUN_P p = my_add;

 printf("%d\n", p(100,200));//300
}
```

案例：给结构体类型取个别名

```c
typedef struct stu
{
 int num;
 char name[32];
 int age;
}STU;//此时的STU不是结构体变量 而是结构体类型

```

```c
struct stu2
{
 int num;
 char name[32];
 int age;
}STU2;//变量名

void test05()
{
 STU lucy={100,"lucy",18};//struct stu lucy

}
```

知识点2【结构体指针】0-2



p = &lucy;　　*p =*&lucy=lucy;

*p == lucy

lucy.num	lucy.name	lucy.age	如果.或->的左边是普通结构体变量 就用.
(*p).num	(*p).name	(*p).age	如果.或->左边是地址 就用->
p->num	p->name	p->age	

案例：结构体指针

```c
#include<stdio.h>

typedef struct
{
 int num;
 char name[16];
 int age;
}STU;
//STU 是结构体类型

void test01()
```

```
{
 STU lucy={100,"lucy",18};

 STU *p = &lucy;


 printf("num = %d, name=%s, age=%d\n",lucy.num, lucy.name, lucy.age);
 printf("num = %d, name=%s, age=%d\n",(*p).num, (*p).name, (*p).age);
 printf("num = %d, name=%s, age=%d\n",p->num, p->name, p->age);

 printf("num = %d\n", (&lucy)->num );

 return;
}
```

案例：从堆区给结构体申请一个空间

```
void test02()
{
 STU *p = NULL;
 //从堆区申请结构体空间
 p = (STU *)calloc(1,sizeof(STU));
 if(NULL == p)
 {
 perror("calloc");
 return;
  }

 //获取键盘输入
 printf("请输入一个学生的信息num name age\n");
 scanf("%d %s %d", &p->num, p->name, &p->age);

 //遍历
 printf("num = %d, name=%s, age=%d\n",p->num, p->name, p->age);

 //释放空间
 if(p != NULL)
 {
 free(p);
 p=NULL;
 }
```

```
25
26   return;
27  }
```

运行结果：

```
请输入一个学生的信息num name age
100 bob 18
num = 100, name=bob, age=18
Press any key to continue
```

知识点3【结构体指针作为函数的参数】

```
1  void mySetSTUData(STU *p)//p=&lucy
2  {
3   printf("请输入一个学生的信息num name age\n");
4   scanf("%d %s %d", &p->num, p->name, &p->age);
5
6   return;
7  }
8  void myPrintSTUData(const STU *p)//p =&lucy *p只读
9  {
10   //const STU *p 不允许用户借助 p修改 p所指向空间的内容
11  // p->num = 10000;
12   printf("sizeof(p)=%d\n", sizeof(p));
13   printf("num = %d, name=%s, age=%d\n",p->num, p->name, p->age);
14  }
15  void test03()
16  {
17   STU lucy;
18   memset(&lucy,0,sizeof(lucy));
19
20   //定义一个函数 给lucy的成员获取键盘输入
21   mySetSTUData(&lucy);
22
23   //定义一个函数 打印lucy的成员信息
24   myPrintSTUData(&lucy);
25  }
```

运行结果：

```
请输入一个学生的信息num name age
100 bob 18
num = 100, name=bob, age=18
Press any key to continue
```

案例：从堆区申请申请一个结构体数组 分函数 实现

```
STU * get_array_addr(int n)
{
 return (STU *)calloc(n,sizeof(STU));
}

//arr代表的是空间首元素地址
void my_input_stu_array(STU *arr, int n)
{
 int i=0;

  for(i=0;i<n;i++)
  {
  printf("请输入第%d个学生的信息\n",i+1);
  //scanf("%d %s %d", &arr[i].num, arr[i].name, &arr[i].age);
  scanf("%d %s %d", &(arr+i)->num , (arr+i)->name, &(arr+i)->age);
  }
}

void my_print_stu_array(const STU *arr, int n)
{
  int i=0;
  for(i=0;i<n;i++)
  {
  printf("num=%d, name=%s, age=%d\n", \
  (arr+i)->num, (arr+i)->name,(arr+i)->age);
  }

  return;
}

void test04()
{
```

```c
    int n = 0;
    STU *arr=NULL;

    printf("请输入学生的个数:");
    scanf("%d", &n);

    //根据学生的个数 从堆区 申请空间
    arr = get_array_addr(n);
    if(arr == NULL)
    {
    perror("get_array_addr");
    return;
    }

    //从键盘 给结构体数组arr输入数据
    my_input_stu_array(arr, n);

    //遍历结构体数组
    my_print_stu_array(arr, n);

    //释放空间
    if(arr != NULL)
    {
    free(arr);
    arr=NULL;
    }
}
```

运行结果：

```
请输入学生的个数:3
请输入第1个学生的信息
1 lucy 18
请输入第2个学生的信息
2 bob 19
请输入第3个学生的信息
3 tom 20
num=1, name=lucy, age=18
num=2, name=bob, age=19
num=3, name=tom, age=20
Press any key to continue
```

知识点4【结构体的内存对齐】1-1

```
struct data
{
    char c;//1B
    int i;//4B
};
```

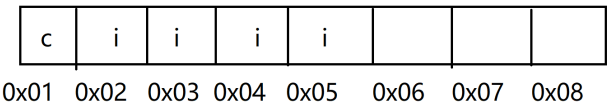
32位CPU 一次性读取4字节

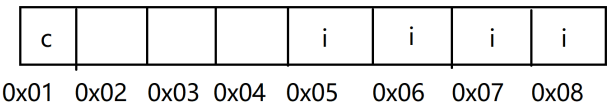
c:需要一个周期 提取0x01~0x04 只要0x01

i:需要两个周期:

第1周期：提取0x01~0x04 只要0x02~0x04

第2周期：提取0x05~0x08 只要0x05





c:需要一个周期 提取0x01~0x04 只要0x01

i:需要一个周期 提取0x05~0x08 只要0x05~0x08

对齐规则：

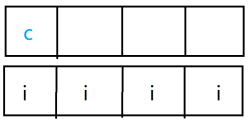
```
struct data
{
    char c;//1B
    int i;//4B
};
```

步骤：

1、确定分配单位：每一行应该分配的字节数 由结构体中最大的基本类型 长度确定

2、确定成员的起始位置的偏移量=成员的基本类型整数（0~n）倍。

3、收尾工作：结构体总大小==分配单位的整数倍。




```
1  struct data
2  {
3   char c;//1B
4   int i;//4B
5  };
6  void test05()
7  {
8   struct data d;
9   //结构体的大小 >= 成员大小之和
10   printf("%d\n",sizeof(struct data));//8
11
12   printf("&d.c = %u\n",&d.c );
13   printf("&d.i = %u\n",&d.i );
14  }
```

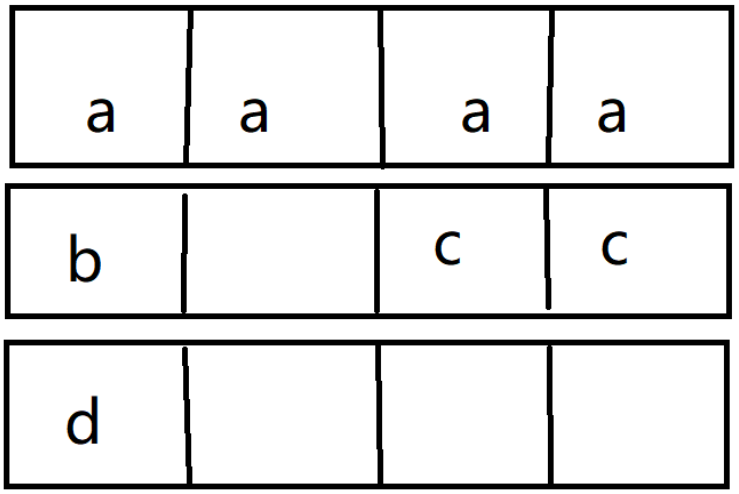
运行结果：

```
8
&d.c = 1703636
&d.i = 1703640
Press any key to continue
```

案例：

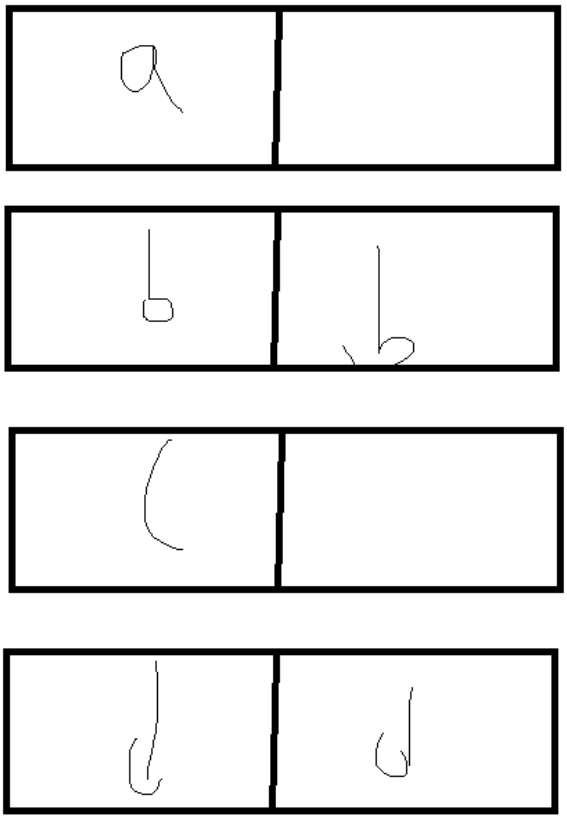
```
1  typedef struct
2  {
3   int a;
4   char b;
5   short c;
6   char d;
7  }DATA;
8
```

```
void test06()
{
 DATA d;
 printf("%d\n", sizeof(DATA));

 printf("%u\n", &d.a);
 printf("%u\n", &d.b);
 printf("%u\n", &d.c);
 printf("%u\n", &d.d);

}
```



案例：

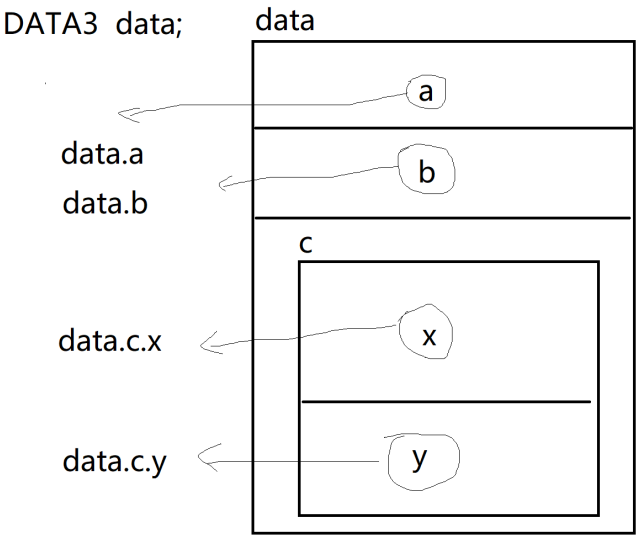
```
typedef struct
{
 char a;
 short b;
 char c;
 short d;
}DATA;
```



结构体体嵌套结构体1-2

```
typedef struct
{
    int x;
    int y;
}DATA2;
```

```
typedef struct
{
    int a;
    int b;
    DATA2 c;
}DATA3;
```

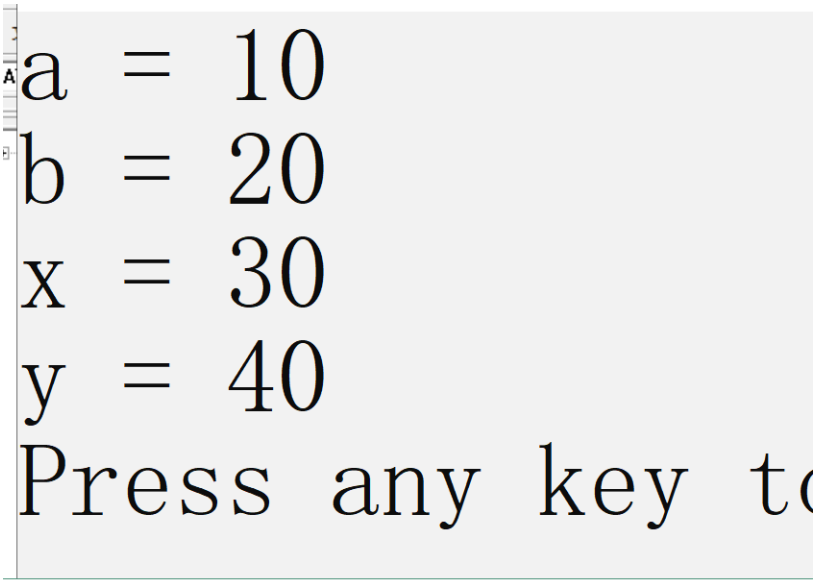


结构体嵌套结构体 一定要访问到最底层

```
typedef struct
{
 int x;
 int y;
}DATA2;
```

```
6
7  typedef struct
8  {
9   int a;
10   int b;
11   DATA2 c;//结构体变量c 作为了DATA3的成员 叫结构体嵌套结构体
12  }DATA3;
13
14  void test07()
15  {
16   //DATA3 data={10,20,30,40};
17   DATA3 data={10,20,{30,40}};//推荐
18
19   printf("a = %d\n", data.a);
20   printf("b = %d\n", data.b);
21   printf("x = %d\n", data.c.x);//访问到最底层
22   printf("y = %d\n", data.c.y);
23  }
```

运行结果：



结构体嵌套结构体的内存对齐

```
typedef struct
{
   short  d
   char  e;
}DATA2;
```

```
typedef struct
{
   short a;
   int b;
   DATA2 c;
   char f;
}DATA;
```

1、确定分配单位：每一行应该分配的字节数

由所有结构体中最大的基本类型长度决定

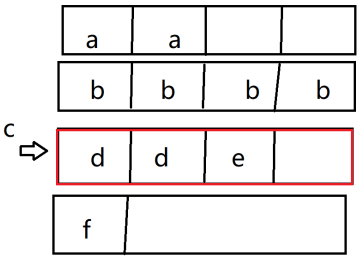
2、确定成员的偏移量 = 自身类型的整数（0~n）倍

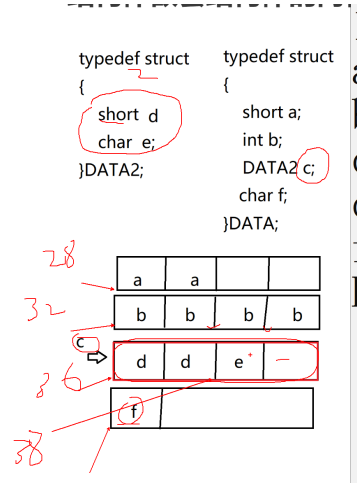
结构体成员 偏移量 = 被嵌套的结构体中最大的基本类型整数倍。

结构体成员 中的 成员偏移量 相对与 被嵌套的结构体

3、收尾工作：结构体的总大小 == 分配单位的整数倍

结构体成员 的总大小 == 被嵌套的结构体里面最大基本类型整数倍






```
16
a:1703628
b:1703632
c中d: 1703636
c中e: 1703638
f:1703640
Press any key to continue
```

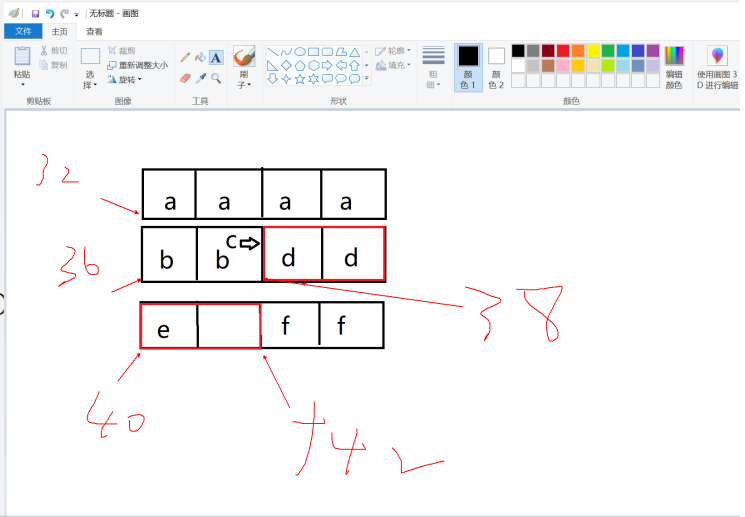
```
typedef struct
{
 short d;
 char e;
}DATA2;

typedef struct
{
 short a;
  int b;
  DATA2 c;
  char f;
}DATA;

void test08()
{
```

```
 DATA data;
 printf("%d\n",sizeof(DATA));

 printf("a:%u\n", &data.a);
 printf("b:%u\n", &data.b);
 printf("c中d: %u\n",&data.c.d);
 printf("c中e: %u\n",&data.c.e);
 printf("f:%u\n",&data.f);
}
```

案例:

```
typedef struct
{
 short d;
 char e;
}B;
typedef struct
{
 int a;
 short b;
 B c;
 short f;
}A;
```

```
12
a:1703632
b:1703636
c中d: 1703638
c中e: 1703640
f:1703642
Press any key to
```

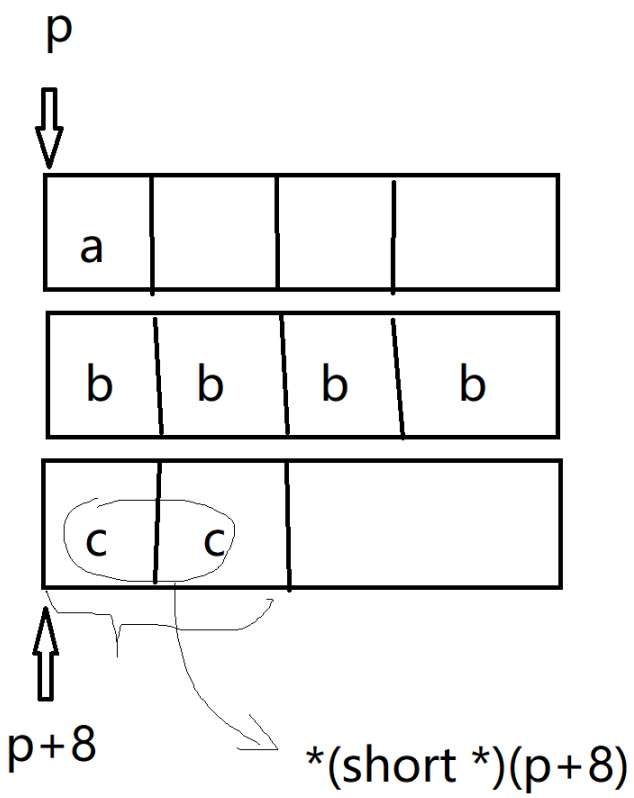


案例: （了解）

```
typedef struct
{
 char a;
 int b;
```

```
 short c;
}DATA;
void test10()
{
 DATA data={'a',100, 20};
  char *p = &data;

  printf("c = %hd\n", data.c);
  //需求 借助p访问20
  printf("c = %hd\n", *(short *)(p+8));
}
```

原理图：



运行结果：

```
c = 20
c = 20
Press any key t
```