基础4 C++类对象布局

```
class Point{
    Point(float xval);
    float x() const;
    static int PointCount;
    virtual ~Point();
    virtual void test();
    float _x;
    static int _count;
};
```

用虚表存放

非虚函数(存在代码区)　虚函数(存放在代码区)

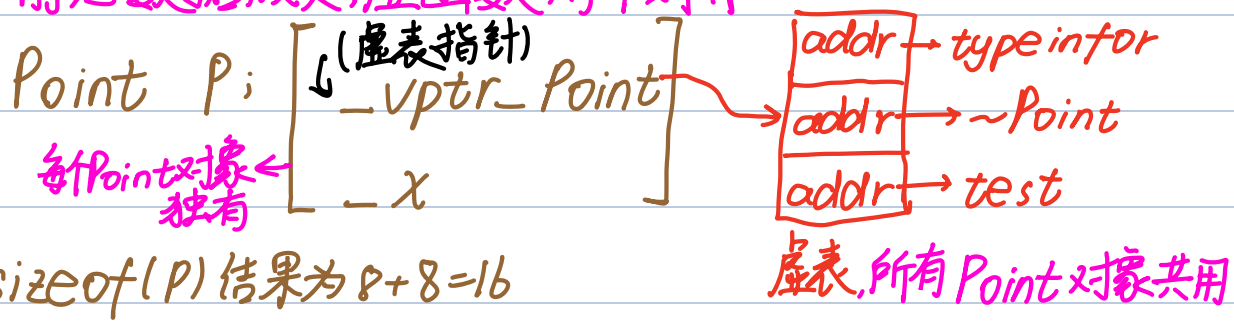
但会在对象中维护一T指针 函数相关

每个对象维护一T

所有对象维护同一T(全局)

①影响C++对象大小的3个因素

非静态数据成员；虚函数；字节对齐 ← 一旦有在，对象多8字节



sizeof(P)结果为8+8=16

②字节对齐：类中变量占有X个字节，那么该T变量应该放在X的整数倍的位置上 以便CPU高效访问内存

注：①long数据类型在win64上占有4字节，在linux 64上占8字节

②基本类型的对齐值=自身大小(char 1字节，int 4字节，double 8字节)

③成员偏移量(相对于起始地址)必须为是自身对齐值的整数倍

④结构体类最后会向内部最大对象对齐

⑤结构体总大小必须为自身对齐值的整数倍

由于内存对齐，类内属性的声明顺序就要注意

相同类型尽量放一起，摆放核心为小在前，大在后

```
struct BadAlign{
  char a;
  double b;
  int c;
  short d;
}
```

a	x	x	x	x	x	x	x
b	b	b	b	b	b	b	b
c	c	c	c	d	d	x	x

(1(a)+7(偏))+8(b)+(4(c)+2(d)+2(x))=24

存在继承时的内存对齐：

```
struct A{ int a;
          short b;}

struct B:public A{short c;}
```

sizeof(B)为 8(A)+2(short)+2(对齐)=12

原因：如果sizeof(B)为8，那么类B强转为类A时会出现问题

A:

a	a	a	a
b	b	x	x

B:

a	a	a	a
b	b	x	x
c	c	x	x

并非

a	a	a	a
b	b	c	c

③初探神秘的虚表

编译器为含虚函数的类生成的函数指针数组

而虚表指针为每个类对象特有的指向虚表的指针

怎么得到虚表(使用类型强转)　A a;

本身为void*，所以要使用强转

```
auto vptr = (int_64*)(&a);
```
得到虚表指针

```
auto virtable = (int_64*)*vptr;
```
解引用指针再强转得到虚表首地址

```
auto virfun1 = (Fun*)*virtable;
```
再次解引用强转得到第一个函数

```
virfun1();
```

豆包给出的获取指针与虚表

```
void** vptr = (void**)&a;
void** virtable = (void**)*vptr;
```

DS给出的获取指针与虚表

```
void **vptr = *(void***)&a;
void **virtable1 = vptr[i];
```