typedef void(\*Fun)(void) 🡪 一个函数指针。这个指针指向一个函数，这个函数的返回值是void，参数列表是void

parent b;

1. &b 🡪 b这个object的地址，这个地址是object b的首地址。
2. (long\*)(&b) 🡪 还是b这个object的地址，只是显示地转化为long\*的指针。因为在64位的机器上long占用8个字节，正好和一个指针占用的大小相等，所以可以转换。
3. \*(long\*)(&b) 🡪 b的首地址所存的内容。一般地，b是个C++的class，那么它的首地址存放的是vtable（虚函数表）的地址，所以这个表达式的值，实际上也是一个地址，这个地址是该object的虚函数表的地址（首地址）。
4. (long\*)\*(long\*)(&b) 🡪 还是b的首地址所存的内容。因为和2中同样的原因，所以这个地址可以转换成long\*这样的指针
5. \*((long\*)\*(long\*)(&b)) 🡪 因为(long\*)\*(long\*)(&b)是虚函数表的首地址，所以对其取值操作，就是取该首地址所存的内容。虚函数表的首地址所存的内容，当然是第一个虚函数的地址了，所以，这个表达式实际上，返回的也是一个函数的地址。
6. Fun pf = (Fun) \*((long\*)\*(long\*)(&b))，Fun如前所示，是一个函数指针，所以这个表达式就是把\*((long\*)\*(long\*)(&b))显示（强制）转换成函数指针，并赋值给同样为函数指针类型的pf，那么，实际上p就指向了object b这个C++ class的第一个虚函数。
7. pf() 🡪根据函数指针调用规则，pf()就是调用pf这个函数指针所指向的函数，实际上就是调用object b的第一个虚函数。
8. (long\*)\*(long\*)(&b) + 1 🡪 一个指针的步进是和该指针所指向的数据类型有关的，在64位机器上long是占用8bytes，那么一个指向long类型的指针（就是long\*）的步进就是8bytes。  
   虚函数表里存的都是指针（即每个占8bytes，亦即64bits），而且在内存上是连续的。而在64位的机器上，指向任意类型的指针占用的内存大小都是64位。那么，既然(long\*)\*(long\*)(&b)是一个long\*的指针，其步进为8bytes(就是64bits)，那么(long\*)\*(long\*)(&b) + 1就是虚函数表里存放第二个指针的地址。以此类推，(long\*)\*(long\*)(&b) + 2是虚函数表里存放第三个指针的地址，等等。  
   所以\*((long\*)\*(long\*)(&b) + 1)就是取虚函数表里第二个函数的指针，即这个表达式（\*((long\*)\*(long\*)(&b) + 1)）是一个指向第二个虚函数的函数指针；第三个虚函数，第四个虚函数… …以此类推。