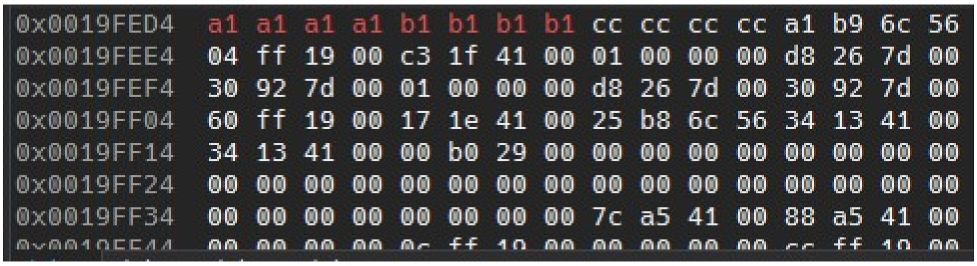
单重继承

无虚函数

```
class A {
public:
    int _a = 0xa1a1a1a1;
};

class B : public A {
public:
    int _b = 0xb1b1b1b1;
};
```

父类数据在前，子类数据在后



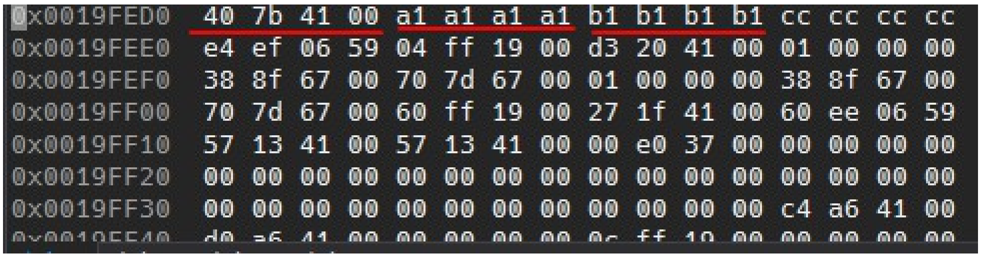
```
0x0019FED4  a1 a1 a1 a1 b1 b1 b1 b1 cc cc cc cc a1 b9 6c 56
0x0019FEE4  04 ff 19 00 c3 1f 41 00 01 00 00 00 d8 26 7d 00
0x0019FEF4  30 92 7d 00 01 00 00 00 d8 26 7d 00 30 92 7d 00
0x0019FF04  60 ff 19 00 17 1e 41 00 25 b8 6c 56 34 13 41 00
0x0019FF14  34 13 41 00 00 b0 29 00 00 00 00 00 00 00 00 00
0x0019FF24  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF34  00 00 00 00 00 00 00 00 7c a5 41 00 88 a5 41 00
0x0019FF44  00 00 00 00 0c ff 19 00 00 00 00 00 cc ff 19 00
```

有虚函数

父类有虚函数

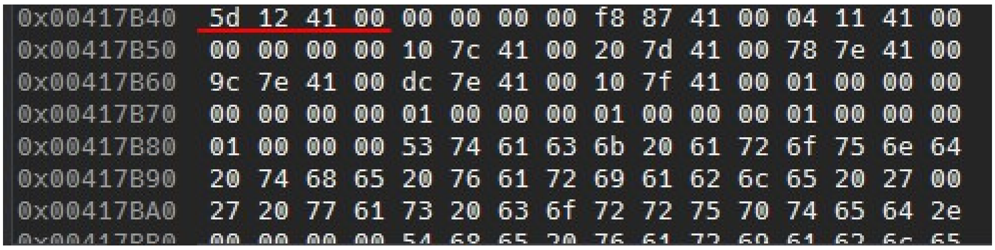
```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : public A {
public:
    int _b = 0xb1b1b1b1;
};
```

同样父类数据在前，子类数据在后



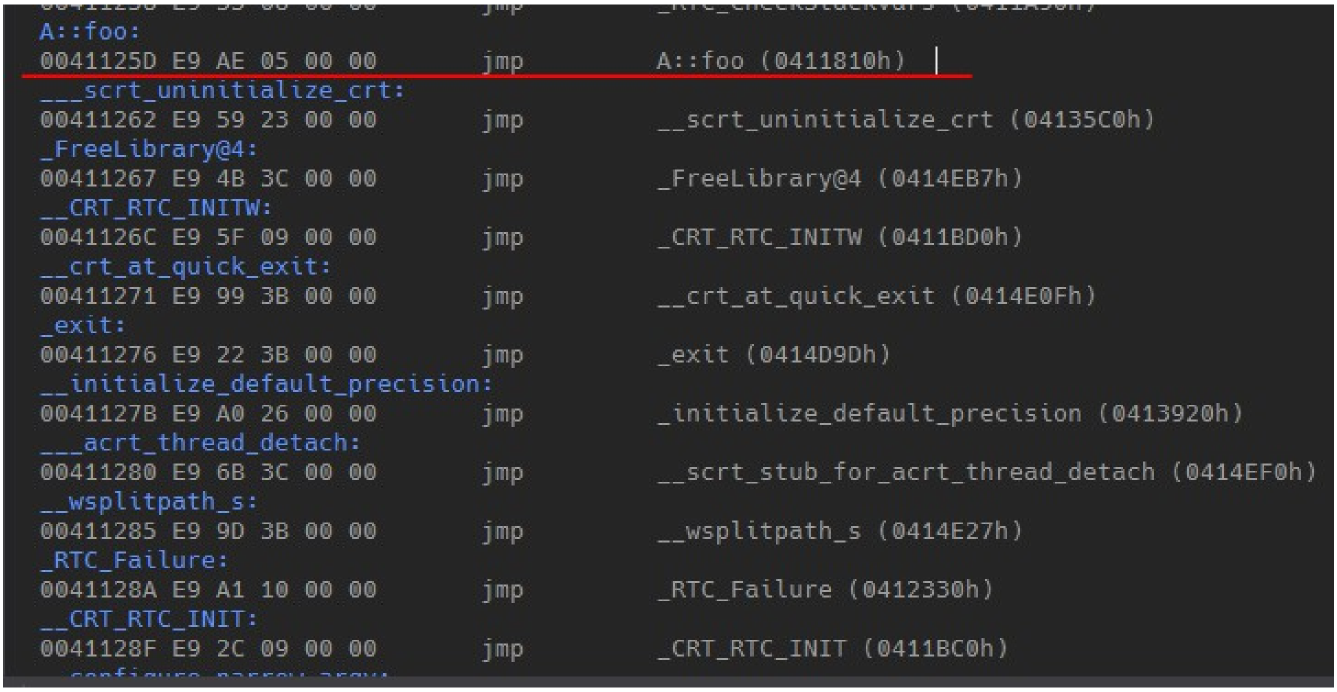
```
0x0019FED0  40 7b 41 00 a1 a1 a1 a1 b1 b1 b1 b1 cc cc cc cc
0x0019FEE0  e4 ef 06 59 04 ff 19 00 d3 20 41 00 01 00 00 00
0x0019FEF0  38 8f 67 00 70 7d 67 00 01 00 00 00 38 8f 67 00
0x0019FF00  70 7d 67 00 60 ff 19 00 27 1f 41 00 60 ee 06 59
0x0019FF10  57 13 41 00 57 13 41 00 00 e0 37 00 00 00 00 00
0x0019FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF30  00 00 00 00 00 00 00 00 00 00 00 00 c4 a6 41 00
0x0019FF40  d0 a6 41 00 00 00 00 00 0c ff 19 00 00 00 00 00
```

对象的首地址是父类的虚表指针



```
0x00417B40  5d 12 41 00 00 00 00 00 f8 87 41 00 04 11 41 00
0x00417B50  00 00 00 00 10 7c 41 00 20 7d 41 00 78 7e 41 00
0x00417B60  9c 7e 41 00 dc 7e 41 00 10 7f 41 00 01 00 00 00
0x00417B70  00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0x00417B80  01 00 00 00 53 74 61 63 6b 20 61 72 6f 75 6e 64
0x00417B90  20 74 68 65 20 76 61 72 69 61 62 6c 65 20 27 00
0x00417BA0  27 20 77 61 73 20 63 6f 72 72 75 70 74 65 64 2e
0x00417BB0  00 00 00 00 54 68 65 20 76 61 72 69 61 62 6c 65
```

指向父类的虚表



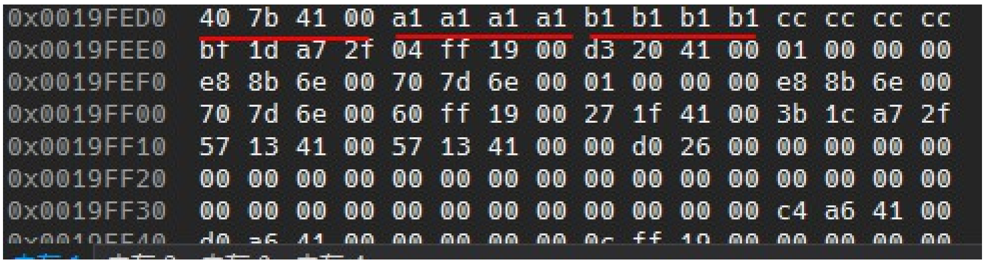
```
A::foo:
0041125D E9 AE 05 00 00       jmp         A::foo (0411810h)
___scrt_uninitialize_crt:
00411262 E9 59 23 00 00       jmp         __scrt_uninitialize_crt (04135C0h)
_FreeLibrary@4:
00411267 E9 4B 3C 00 00       jmp         _FreeLibrary@4 (0414EB7h)
__CRT_RTC_INITW:
0041126C E9 5F 09 00 00       jmp         _CRT_RTC_INITW (0411BD0h)
__crt_at_quick_exit:
00411271 E9 99 3B 00 00       jmp         __crt_at_quick_exit (0414E0Fh)
_exit:
00411276 E9 22 3B 00 00       jmp         _exit (0414D9Dh)
__initialize_default_precision:
0041127B E9 A0 26 00 00       jmp         _initialize_default_precision (0413920h)
___acrt_thread_detach:
00411280 E9 6B 3C 00 00       jmp         __scrt_stub_for_acrt_thread_detach (0414EF0h)
__wsplitpath_s:
00411285 E9 9D 3B 00 00       jmp         __wsplitpath_s (0414E27h)
_RTC_Failure:
0041128A E9 A1 10 00 00       jmp         _RTC_Failure (0412330h)
__CRT_RTC_INIT:
0041128F E9 2C 09 00 00       jmp         _CRT_RTC_INIT (0411BC0h)
__configure_narrow_argv:
```

子类重写父类虚函数

```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : public A {
```

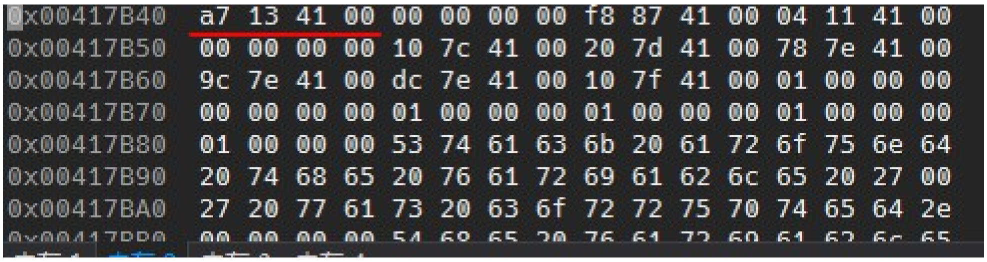
```
public:
    virtual int foo()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};
```

此时同样是父类数据在前，子类在后



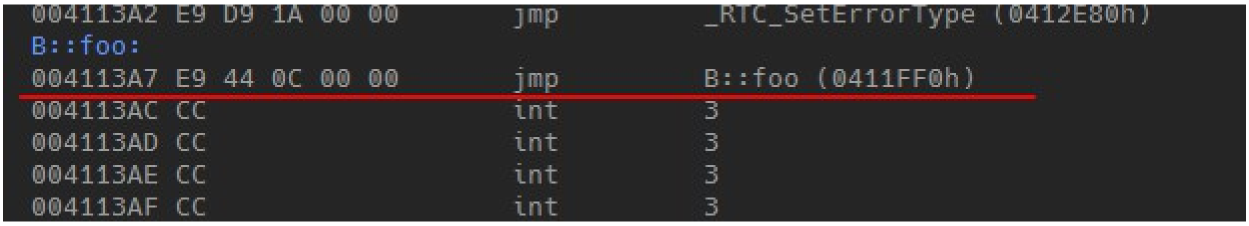
```
0x0019FED0  40 7b 41 00 a1 a1 a1 a1 b1 b1 b1 b1 cc cc cc cc
0x0019FEE0  bf 1d a7 2f 04 ff 19 00 d3 20 41 00 01 00 00 00
0x0019FEF0  e8 8b 6e 00 70 7d 6e 00 01 00 00 00 e8 8b 6e 00
0x0019FF00  70 7d 6e 00 60 ff 19 00 27 1f 41 00 3b 1c a7 2f
0x0019FF10  57 13 41 00 57 13 41 00 00 d0 26 00 00 00 00 00
0x0019FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF30  00 00 00 00 00 00 00 00 00 00 00 00 c4 a6 41 00
0x0019FF40  d0 a6 41 00 00 00 00 00 0c ff 19 00 00 00 00 00
```

对象的首地址是子类的虚表指针



```
0x00417B40  a7 13 41 00 00 00 00 00 f8 87 41 00 04 11 41 00
0x00417B50  00 00 00 00 10 7c 41 00 20 7d 41 00 78 7e 41 00
0x00417B60  9c 7e 41 00 dc 7e 41 00 10 7f 41 00 01 00 00 00
0x00417B70  00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0x00417B80  01 00 00 00 53 74 61 63 6b 20 61 72 6f 75 6e 64
0x00417B90  20 74 68 65 20 76 61 72 69 61 62 6c 65 20 27 00
0x00417BA0  27 20 77 61 73 20 63 6f 72 72 75 70 74 65 64 2e
0x00417BB0  00 00 00 00 54 68 65 20 76 61 72 69 61 62 6c 65
```

指向子类的虚表



```
004113A2 E9 D9 1A 00 00       jmp         _RTC_SetErrorType (0412E80h)
B::foo:
004113A7 E9 44 0C 00 00       jmp         B::foo (0411FF0h)
004113AC CC                   int         3
004113AD CC                   int         3
004113AE CC                   int         3
004113AF CC                   int         3
```

注

以下情况内存分布也一样，不在赘述

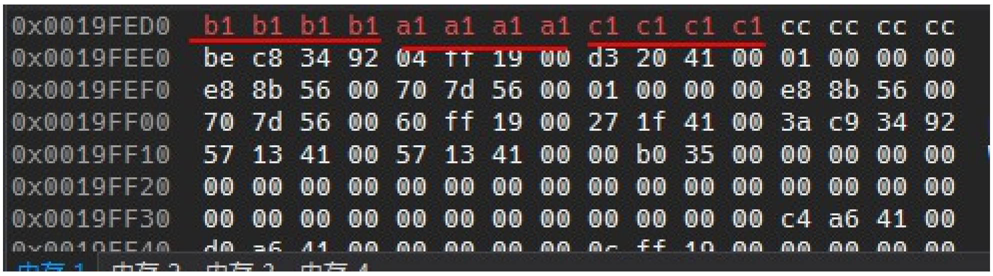
- 子类新增虚函数
- 父类没有虚函数而子类有

多重继承

无虚函数

```
class A {
public:
    int _a = 0xa1a1a1a1;
};

class B {
public:
    int _b = 0xb1b1b1b1;
};

class C : public B, public A {
public:
    int _c = 0xc1c1c1c1;
};
```

按继承的次序，顺位存放父类的数据，最后在存放子类的数据



```
0x0019FED0  b1 b1 b1 b1 a1 a1 a1 a1 c1 c1 c1 c1 cc cc cc cc
0x0019FEE0  be c8 34 92 04 ff 19 00 d3 20 41 00 01 00 00 00
0x0019FEF0  e8 8b 56 00 70 7d 56 00 01 00 00 00 e8 8b 56 00
0x0019FF00  70 7d 56 00 60 ff 19 00 27 1f 41 00 3a c9 34 92
0x0019FF10  57 13 41 00 57 13 41 00 00 b0 35 00 00 00 00 00
0x0019FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF30  00 00 00 00 00 00 00 00 00 00 00 00 c4 a6 41 00
0x0019FF40  d0 a6 41 00 00 00 00 00 0c ff 19 00 00 00 00 00
```

有虚函数

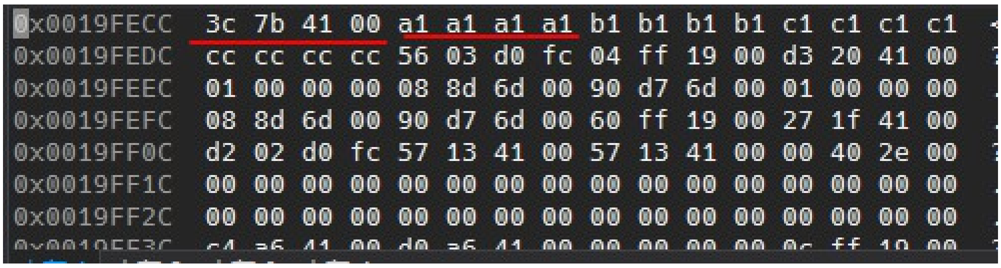
父类有虚函数

1. 父类A有虚函数

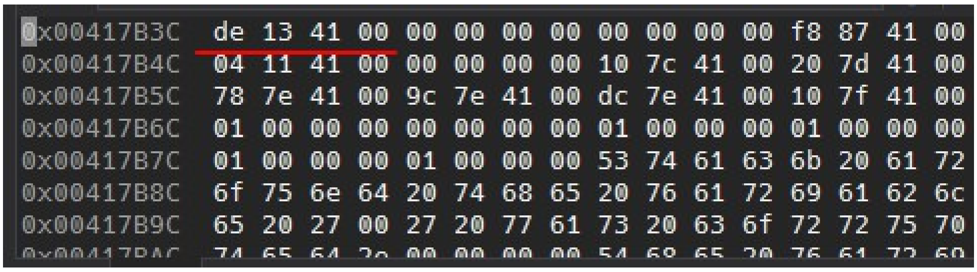
```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B {
public:
    int _b = 0xb1b1b1b1;
};
```

```
class C : public B, public A {
public:
    int _c = 0xc1c1c1c1;
};
```

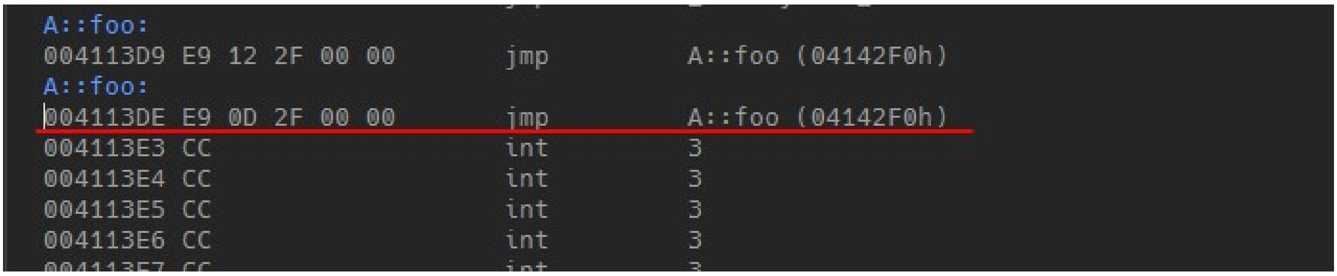
此时，内存分布为：父类A的虚表指针、父类A的数据、父类B的数据和子类C的数据



```
0x0019FECC  3c 7b 41 00 a1 a1 a1 a1 b1 b1 b1 b1 c1 c1 c1 c1
0x0019FEDC  cc cc cc cc 56 03 d0 fc 04 ff 19 00 d3 20 41 00
0x0019FEEC  01 00 00 00 08 8d 6d 00 90 d7 6d 00 01 00 00 00
0x0019FEFC  08 8d 6d 00 90 d7 6d 00 60 ff 19 00 27 1f 41 00
0x0019FF0C  d2 02 d0 fc 57 13 41 00 57 13 41 00 00 40 2e 00
0x0019FF1C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF2C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



```
0x00417B3C  de 13 41 00 00 00 00 00 00 00 00 00 f8 87 41 00
0x00417B4C  04 11 41 00 00 00 00 00 10 7c 41 00 20 7d 41 00
0x00417B5C  78 7e 41 00 9c 7e 41 00 dc 7e 41 00 10 7f 41 00
0x00417B6C  01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0x00417B7C  01 00 00 00 01 00 00 00 53 74 61 63 6b 20 61 72
0x00417B8C  6f 75 6e 64 20 74 68 65 20 76 61 72 69 61 62 6c
0x00417B9C  65 20 27 00 27 20 77 61 73 20 63 6f 72 72 75 70
```

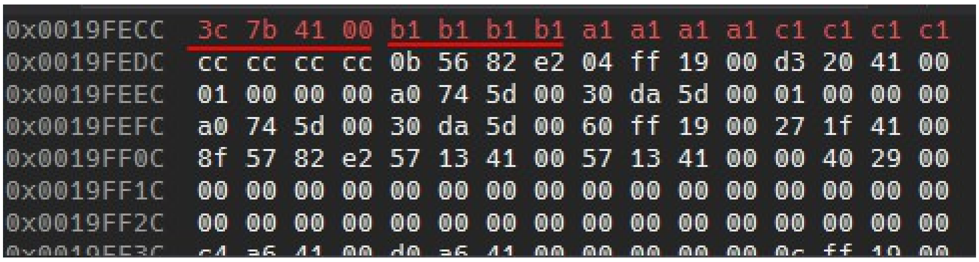


```
A::foo:
004113D9 E9 12 2F 00 00       jmp         A::foo (04142F0h)
A::foo:
004113DE E9 0D 2F 00 00       jmp         A::foo (04142F0h)
004113E3 CC                   int         3
004113E4 CC                   int         3
004113E5 CC                   int         3
004113E6 CC                   int         3
```

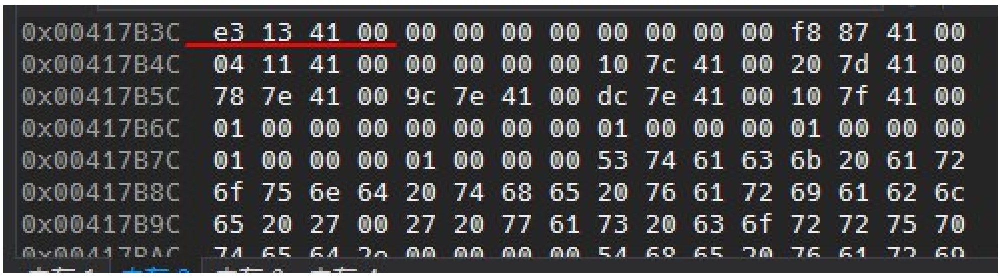
2. 父类B有虚函数

```
class A {
public:
    int _a = 0xa1a1a1a1;
};

class B {
public:
    virtual int bar()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};


class C : public B, public A {
public:
    int _c = 0xc1c1c1c1;
};
```

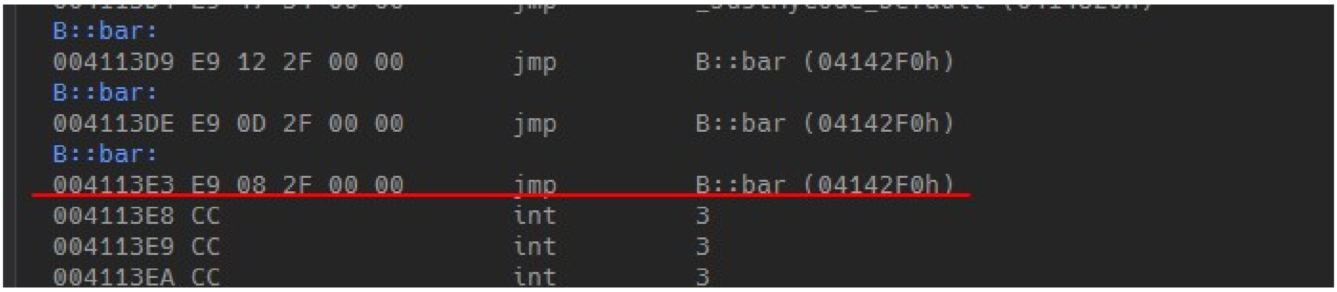
此时，内存分布为：父类C的虚表指针、父类C的数据、父类A的数据和子类C的数据



```
0x0019FECC  3c 7b 41 00 b1 b1 b1 b1 a1 a1 a1 a1 c1 c1 c1 c1
0x0019FEDC  cc cc cc cc 0b 56 82 e2 04 ff 19 00 d3 20 41 00
0x0019FEEC  01 00 00 00 a0 74 5d 00 30 da 5d 00 01 00 00 00
0x0019FEFC  a0 74 5d 00 30 da 5d 00 60 ff 19 00 27 1f 41 00
0x0019FF0C  8f 57 82 e2 57 13 41 00 57 13 41 00 00 40 29 00
0x0019FF1C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF2C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF3C  c4 a6 41 00 d0 a6 41 00 00 00 00 00 0c ff 19 00
```



```
0x00417B3C  e3 13 41 00 00 00 00 00 00 00 00 00 f8 87 41 00
0x00417B4C  04 11 41 00 00 00 00 00 10 7c 41 00 20 7d 41 00
0x00417B5C  78 7e 41 00 9c 7e 41 00 dc 7e 41 00 10 7f 41 00
0x00417B6C  01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0x00417B7C  01 00 00 00 01 00 00 00 53 74 61 63 6b 20 61 72
0x00417B8C  6f 75 6e 64 20 74 68 65 20 76 61 72 69 61 62 6c
0x00417B9C  65 20 27 00 27 20 77 61 73 20 63 6f 72 72 75 70
0x00417BAC  74 65 64 2e 00 00 00 00 54 68 65 20 76 61 72 69
```

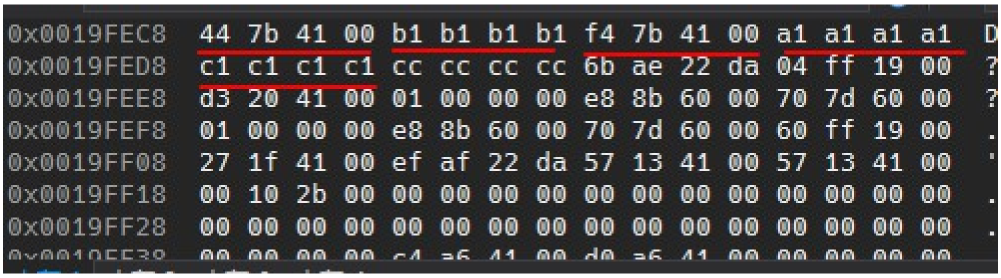


```
B::bar:
004113D9 E9 12 2F 00 00       jmp          B::bar (04142F0h)
B::bar:
004113DE E9 0D 2F 00 00       jmp          B::bar (04142F0h)
B::bar:
004113E3 E9 08 2F 00 00       jmp          B::bar (04142F0h)
004113E8 CC                   int          3
004113E9 CC                   int          3
004113EA CC                   int          3
```

3. 两个父类都有虚函数

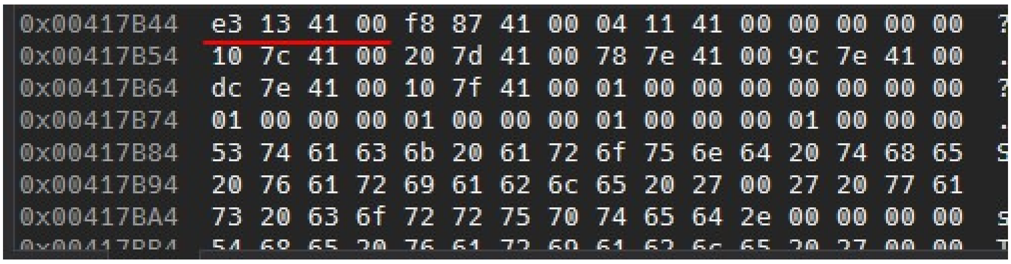
```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B {
public:
    virtual int bar()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};


class C : public B, public A {
public:
    int _c = 0xc1c1c1c1;
};
```

此时内存布局为：父类B虚表指针 、父类B数据、父类A虚表指针、父类A数据和子类C数据

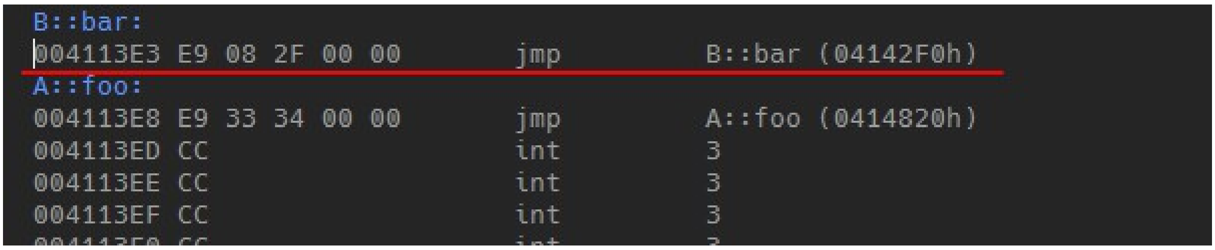


```
0x0019FEC8  44 7b 41 00 b1 b1 b1 b1 f4 7b 41 00 a1 a1 a1 a1  D
0x0019FED8  c1 c1 c1 c1 cc cc cc cc 6b ae 22 da 04 ff 19 00  ?
0x0019FEE8  d3 20 41 00 01 00 00 00 e8 8b 60 00 70 7d 60 00  ?
0x0019FEF8  01 00 00 00 e8 8b 60 00 70 7d 60 00 60 ff 19 00  .
0x0019FF08  27 1f 41 00 ef af 22 da 57 13 41 00 57 13 41 00  '
0x0019FF18  00 10 2b 00 00 00 00 00 00 00 00 00 00 00 00 00  .
0x0019FF28  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .
0x0019FF38  00 00 00 00 c4 a6 41 00 d0 a6 41 00 00 00 00 00
```

父类B的虚表如下：

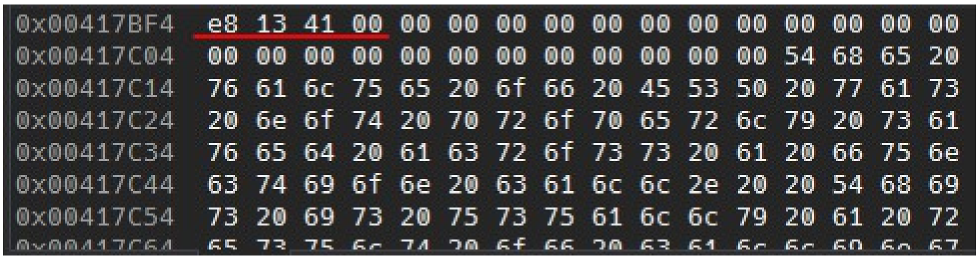


```
0x00417B44  e3 13 41 00 f8 87 41 00 04 11 41 00 00 00 00 00  ?
0x00417B54  10 7c 41 00 20 7d 41 00 78 7e 41 00 9c 7e 41 00  .
0x00417B64  dc 7e 41 00 10 7f 41 00 01 00 00 00 00 00 00 00  ?
0x00417B74  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00  .
0x00417B84  53 74 61 63 6b 20 61 72 6f 75 6e 64 20 74 68 65  S
0x00417B94  20 76 61 72 69 61 62 6c 65 20 27 00 27 20 77 61
0x00417BA4  73 20 63 6f 72 72 75 70 74 65 64 2e 00 00 00 00  s
0x00417BB4  54 68 65 20 76 61 72 69 61 62 6c 65 20 27 00 00  T
```

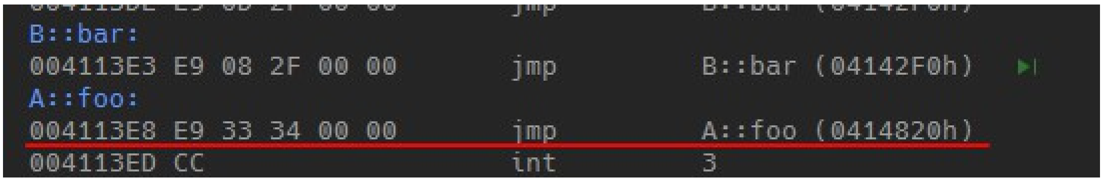


```
B::bar:
004113E3 E9 08 2F 00 00       jmp         B::bar (04142F0h)
A::foo:
004113E8 E9 33 34 00 00       jmp         A::foo (0414820h)
004113ED CC                   int         3
004113EE CC                   int         3
004113EF CC                   int         3
004113F0 CC                   int         3
```

父类A的虚表如下：



```
0x00417BF4  e8 13 41 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00417C04  00 00 00 00 00 00 00 00 00 00 00 00 54 68 65 20
0x00417C14  76 61 6c 75 65 20 6f 66 20 45 53 50 20 77 61 73
0x00417C24  20 6e 6f 74 20 70 72 6f 70 65 72 6c 79 20 73 61
0x00417C34  76 65 64 20 61 63 72 6f 73 73 20 61 20 66 75 6e
0x00417C44  63 74 69 6f 6e 20 63 61 6c 6c 2e 20 20 54 68 69
0x00417C54  73 20 69 73 20 75 73 75 61 6c 6c 79 20 61 20 72
0x00417C64  65 73 75 6c 74 20 6f 66 20 63 61 6c 6c 69 6e 67
```



```
B::bar:
004113E3 E9 08 2F 00 00       jmp         B::bar (04142F0h)
A::foo:
004113E8 E9 33 34 00 00       jmp         A::foo (0414820h)
004113ED CC                   int         3
```

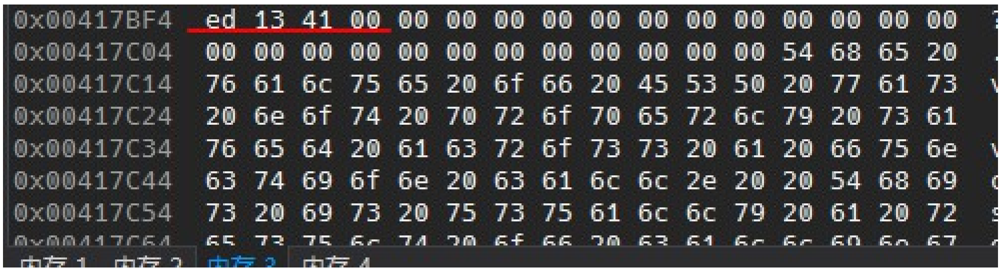
即——父类有虚函数，则会为哪个父类添加一个虚表

子类重写父类的虚函数

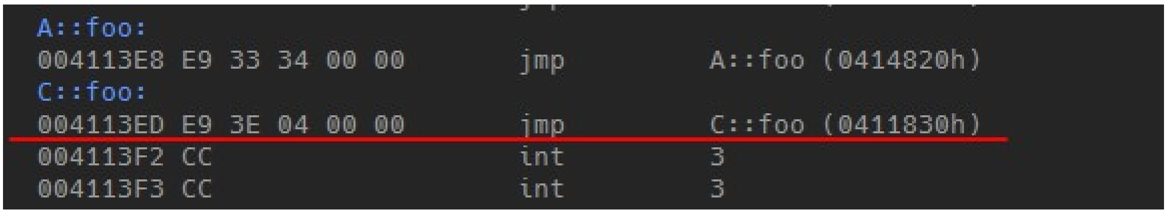
```
class A {
```

```
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B {
public:
    virtual int bar()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};


class C : public B, public A {
public:
    virtual int foo()
    {
        return _c;
    }

    int _c = 0xc1c1c1c1;
};
```

此时内存布局与上例保持一致，但是子类重写的虚函数会覆盖父类虚表中的对应虚函数的位置



```
0x00417BF4  ed 13 41 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00417C04  00 00 00 00 00 00 00 00 00 00 00 00 54 68 65 20
0x00417C14  76 61 6c 75 65 20 6f 66 20 45 53 50 20 77 61 73
0x00417C24  20 6e 6f 74 20 70 72 6f 70 65 72 6c 79 20 73 61
0x00417C34  76 65 64 20 61 63 72 6f 73 73 20 61 20 66 75 6e
0x00417C44  63 74 69 6f 6e 20 63 61 6c 6c 2e 20 20 54 68 69
0x00417C54  73 20 69 73 20 75 73 75 61 6c 6c 79 20 61 20 72
0x00417C64  65 73 75 6c 74 20 6f 66 20 63 61 6c 6c 69 6e 67
```

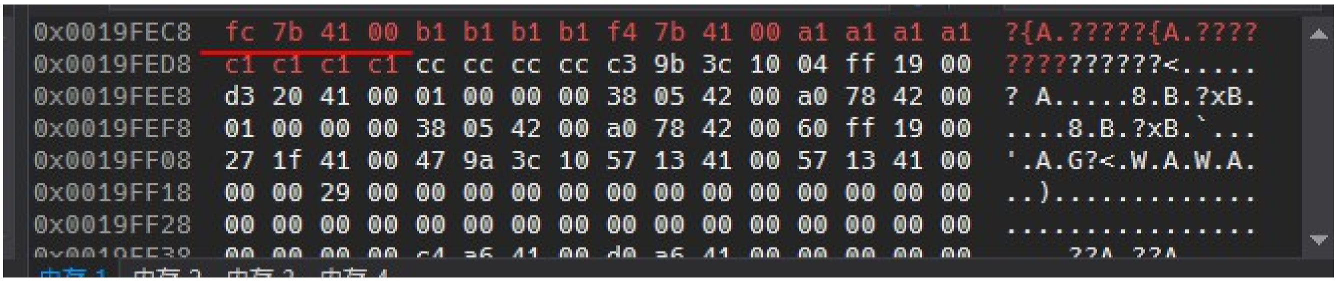


```
A::foo:
004113E8 E9 33 34 00 00        jmp         A::foo (0414820h)
C::foo:
004113ED E9 3E 04 00 00        jmp         C::foo (0411830h)
004113F2 CC                    int         3
004113F3 CC                    int         3
```

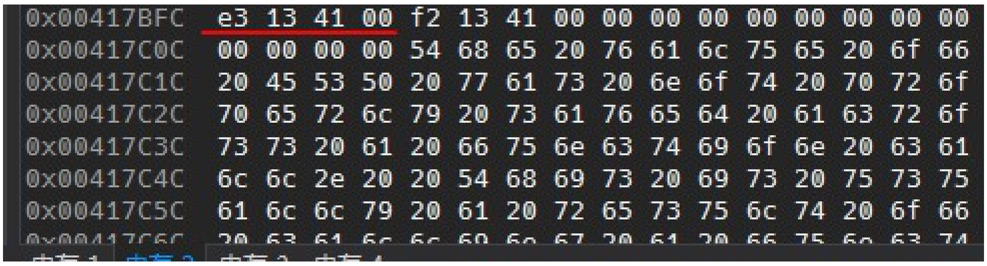
子类新加虚函数

```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B {
public:
    virtual int bar()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};


class C : public B, public A {
public:
    virtual int foo()
    {
        return _c;
    }

    virtual int test()
    {
        return 1;
    }

    int _c = 0xc1c1c1c1;
};
```

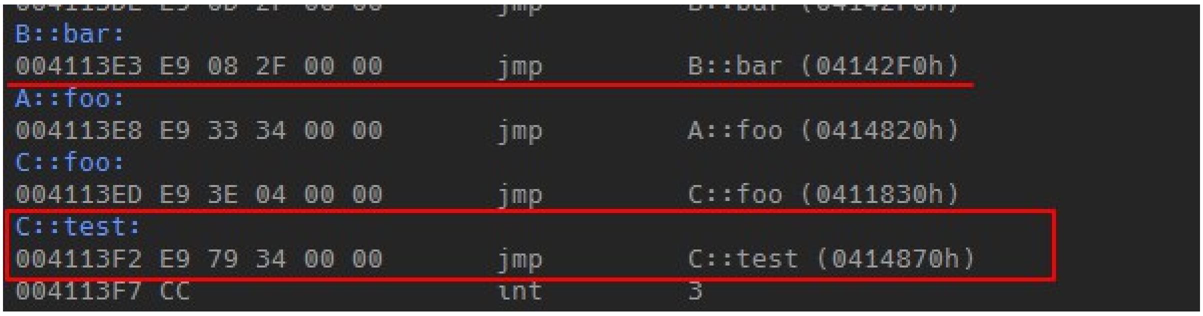
此时内存布局依然同上例保持一致，但子类新加的虚函数会挂靠到第一顺位父类的虚表中



```
0x0019FEC8  fc 7b 41 00 b1 b1 b1 b1 f4 7b 41 00 a1 a1 a1 a1  ?{A.?????{A.????
0x0019FED8  c1 c1 c1 c1 cc cc cc cc c3 9b 3c 10 04 ff 19 00  ??????????<.....
0x0019FEE8  d3 20 41 00 01 00 00 00 38 05 42 00 a0 78 42 00  ? A.....8.B.?xB.
0x0019FEF8  01 00 00 00 38 05 42 00 a0 78 42 00 60 ff 19 00  ....8.B.?xB.`...
0x0019FF08  27 1f 41 00 47 9a 3c 10 57 13 41 00 57 13 41 00  '.A.G?<.W.A.W.A.
0x0019FF18  00 00 29 00 00 00 00 00 00 00 00 00 00 00 00 00  ..).............
0x0019FF28  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0019FF38  00 00 00 00 c4 a6 41 00 d0 a6 41 00 00 00 00 00  ....??A.??A.....
```



```
0x00417BFC  e3 13 41 00 f2 13 41 00 00 00 00 00 00 00 00 00
0x00417C0C  00 00 00 00 54 68 65 20 76 61 6c 75 65 20 6f 66
0x00417C1C  20 45 53 50 20 77 61 73 20 6e 6f 74 20 70 72 6f
0x00417C2C  70 65 72 6c 79 20 73 61 76 65 64 20 61 63 72 6f
0x00417C3C  73 73 20 61 20 66 75 6e 63 74 69 6f 6e 20 63 61
0x00417C4C  6c 6c 2e 20 20 54 68 69 73 20 69 73 20 75 73 75
0x00417C5C  61 6c 6c 79 20 61 20 72 65 73 75 6c 74 20 6f 66
0x00417C6C  20 63 61 6c 6c 69 6e 67 20 61 20 66 75 6e 63 74
```

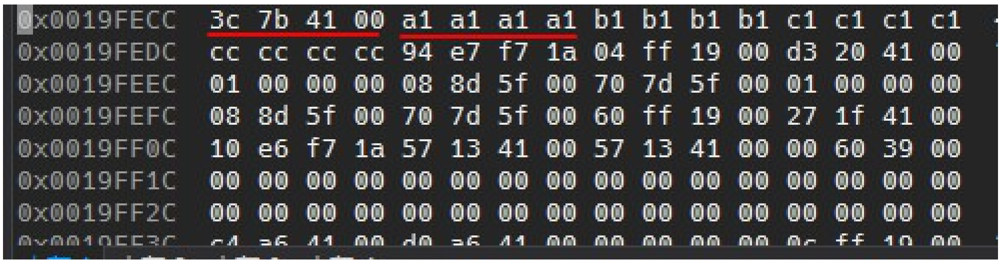


```
B::bar:
004113E3 E9 08 2F 00 00       jmp         B::bar (04142F0h)
A::foo:
004113E8 E9 33 34 00 00       jmp         A::foo (0414820h)
C::foo:
004113ED E9 3E 04 00 00       jmp         C::foo (0411830h)
C::test:
004113F2 E9 79 34 00 00       jmp         C::test (0414870h)
004113F7 CC                   int         3
```

只有一个父类有虚函数

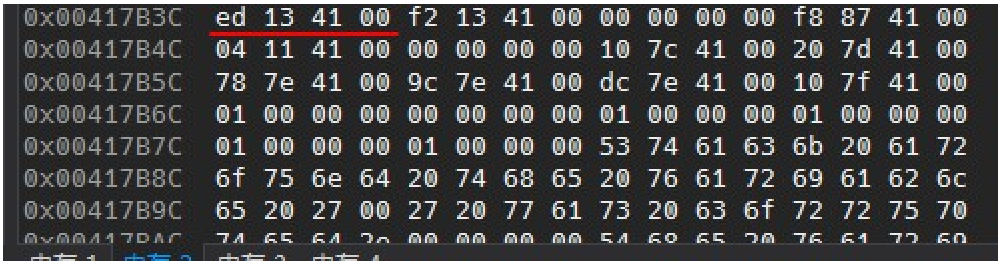
```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B {
public:

    int _b = 0xb1b1b1b1;
};


class C : public B, public A {
public:
    virtual int foo()
    {
        return _c;
    }

    virtual int test()
    {
        return 1;
    }

    int _c = 0xc1c1c1c1;
};
```

综合前例，有虚函数的父类的内存位于最上面，子类的虚函数挂靠到这个父类的虚表中

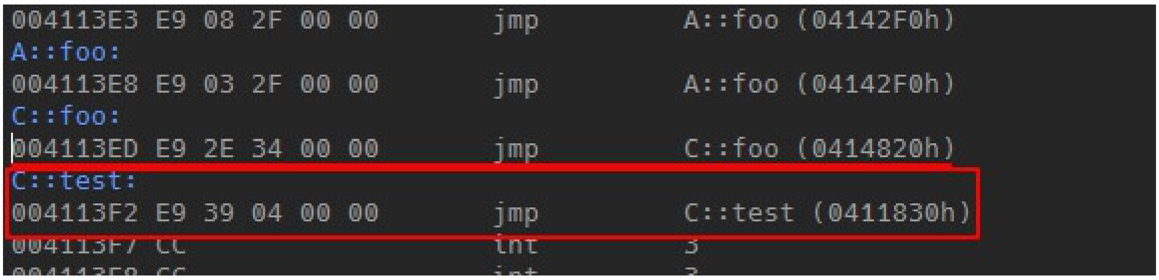


```
0x0019FECC  3c 7b 41 00 a1 a1 a1 a1 b1 b1 b1 b1 c1 c1 c1 c1
0x0019FEDC  cc cc cc cc 94 e7 f7 1a 04 ff 19 00 d3 20 41 00
0x0019FEEC  01 00 00 00 08 8d 5f 00 70 7d 5f 00 01 00 00 00
0x0019FEFC  08 8d 5f 00 70 7d 5f 00 60 ff 19 00 27 1f 41 00
0x0019FF0C  10 e6 f7 1a 57 13 41 00 57 13 41 00 00 60 39 00
0x0019FF1C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF2C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF3C  c4 a6 41 00 d0 a6 41 00 00 00 00 00 0c ff 19 00
```

父类A的虚表如下：



```
0x00417B3C  ed 13 41 00 f2 13 41 00 00 00 00 00 f8 87 41 00
0x00417B4C  04 11 41 00 00 00 00 00 10 7c 41 00 20 7d 41 00
0x00417B5C  78 7e 41 00 9c 7e 41 00 dc 7e 41 00 10 7f 41 00
0x00417B6C  01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0x00417B7C  01 00 00 00 01 00 00 00 53 74 61 63 6b 20 61 72
0x00417B8C  6f 75 6e 64 20 74 68 65 20 76 61 72 69 61 62 6c
0x00417B9C  65 20 27 00 27 20 77 61 73 20 63 6f 72 72 75 70
0x00417BAC  74 65 64 2e 00 00 00 00 54 68 65 20 76 61 72 69
```



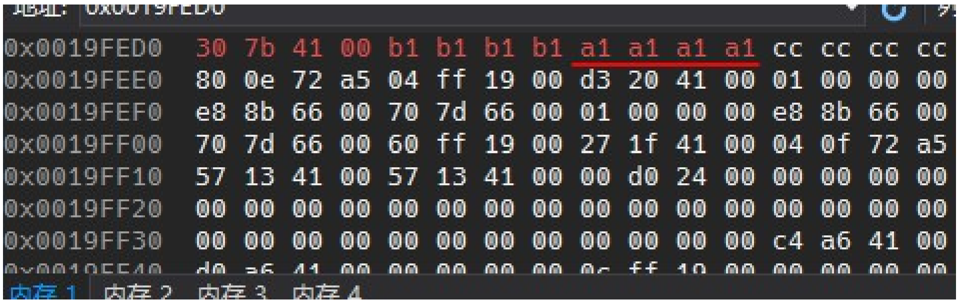
```
004113E3 E9 08 2F 00 00       jmp          A::foo (04142F0h)
A::foo:
004113E8 E9 03 2F 00 00       jmp          A::foo (04142F0h)
C::foo:
004113ED E9 2E 34 00 00       jmp          C::foo (0414820h)
C::test:
004113F2 E9 39 04 00 00       jmp          C::test (0411830h)
004113F7 CC                   int          3
004113F8 CC                   int          3
```

单重虚继承

无虚函数

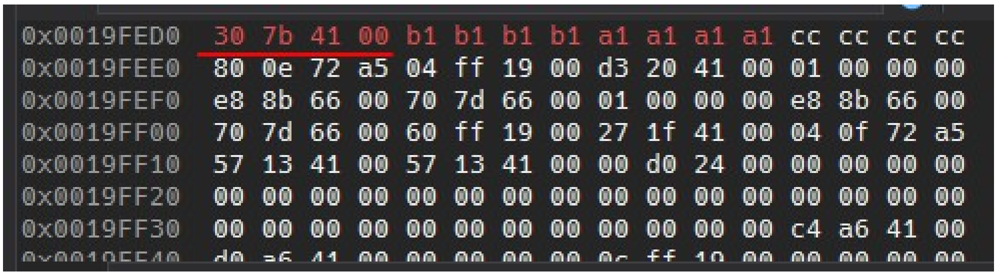
```
class A {
public:
    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    int _b = 0xb1b1b1b1;
};
```

虚基类位于内存的末尾



```
0x0019FED0  30 7b 41 00 b1 b1 b1 b1 a1 a1 a1 a1 cc cc cc cc
0x0019FEE0  80 0e 72 a5 04 ff 19 00 d3 20 41 00 01 00 00 00
0x0019FEF0  e8 8b 66 00 70 7d 66 00 01 00 00 00 e8 8b 66 00
0x0019FF00  70 7d 66 00 60 ff 19 00 27 1f 41 00 04 0f 72 a5
0x0019FF10  57 13 41 00 57 13 41 00 00 d0 24 00 00 00 00 00
0x0019FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF30  00 00 00 00 00 00 00 00 00 00 00 00 c4 a6 41 00
0x0019FF40  d0 a6 41 00 00 00 00 00 0c ff 19 00 00 00 00 00
```

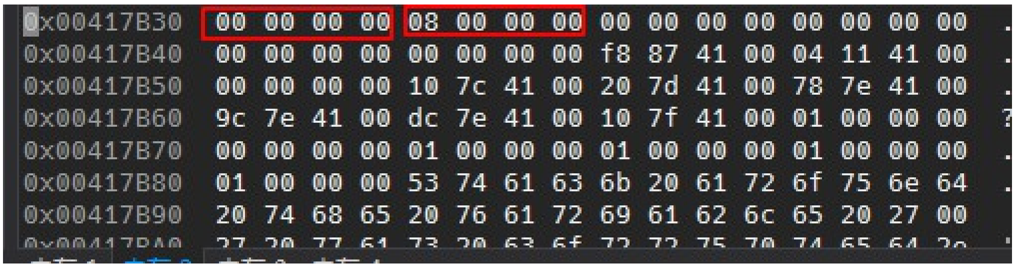
内存首地址存储一个“偏移块指针”指向“偏移块”



```
0x0019FED0  30 7b 41 00 b1 b1 b1 b1 a1 a1 a1 a1 cc cc cc cc
0x0019FEE0  80 0e 72 a5 04 ff 19 00 d3 20 41 00 01 00 00 00
0x0019FEF0  e8 8b 66 00 70 7d 66 00 01 00 00 00 e8 8b 66 00
0x0019FF00  70 7d 66 00 60 ff 19 00 27 1f 41 00 04 0f 72 a5
0x0019FF10  57 13 41 00 57 13 41 00 00 d0 24 00 00 00 00 00
0x0019FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF30  00 00 00 00 00 00 00 00 00 00 00 00 c4 a6 41 00
0x0019FF40  d0 a6 41 00 00 00 00 00 0c ff 19 00 00 00 00 00
```

“偏移块”大小为8字节，其中

- 第一个4字节保存了“偏移块”与本类的首地址的偏移（**猜测**）
- 第二个4字节保存了虚基类到本“偏移块指针”所在处的偏移值



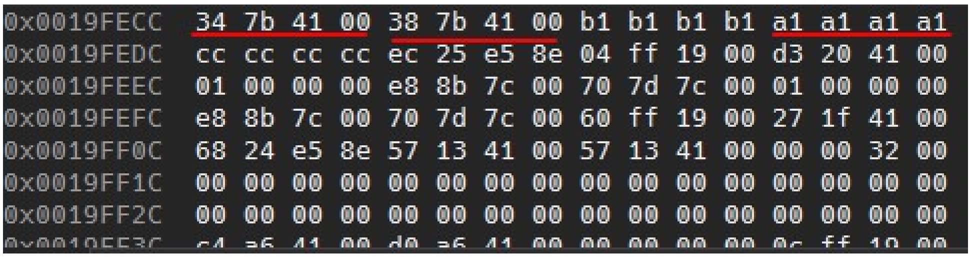
```
0x00417B30  00 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00  .
0x00417B40  00 00 00 00 00 00 00 00 f8 87 41 00 04 11 41 00  .
0x00417B50  00 00 00 00 10 7c 41 00 20 7d 41 00 78 7e 41 00  .
0x00417B60  9c 7e 41 00 dc 7e 41 00 10 7f 41 00 01 00 00 00  ?
0x00417B70  00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00  .
0x00417B80  01 00 00 00 53 74 61 63 6b 20 61 72 6f 75 6e 64  .
0x00417B90  20 74 68 65 20 76 61 72 69 61 62 6c 65 20 27 00
0x00417BA0  27 20 77 61 73 20 63 6f 72 72 75 70 74 65 64 2e  '
```

有虚函数

父类没有虚函数

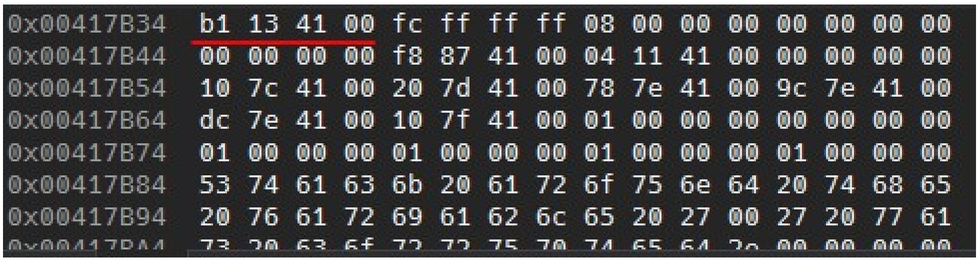
```
![13](E:\作业\一阶段\20190516\img\13.jpg)class A {
public:
    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    virtual int foo()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};
```

依然虚基类在内存末尾，最开头依次是子类虚表指针与“偏移块指针”，父类没有虚表

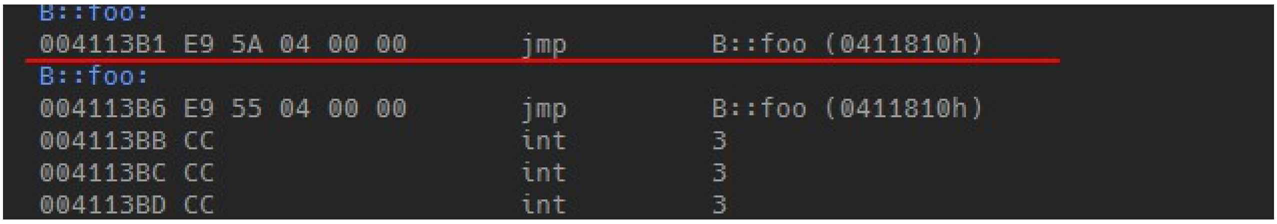


```
0x0019FECC  34 7b 41 00 38 7b 41 00 b1 b1 b1 b1 a1 a1 a1 a1
0x0019FEDC  cc cc cc cc ec 25 e5 8e 04 ff 19 00 d3 20 41 00
0x0019FEEC  01 00 00 00 e8 8b 7c 00 70 7d 7c 00 01 00 00 00
0x0019FEFC  e8 8b 7c 00 70 7d 7c 00 60 ff 19 00 27 1f 41 00
0x0019FF0C  68 24 e5 8e 57 13 41 00 57 13 41 00 00 00 32 00
0x0019FF1C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF2C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF3C  c4 a6 41 00 d0 a6 41 00 00 00 00 00 0c ff 19 00
```

子类虚表

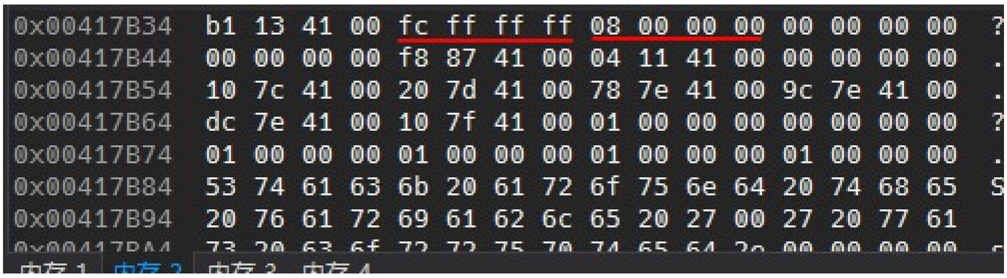


```
0x00417B34  b1 13 41 00 fc ff ff ff 08 00 00 00 00 00 00 00
0x00417B44  00 00 00 00 f8 87 41 00 04 11 41 00 00 00 00 00
0x00417B54  10 7c 41 00 20 7d 41 00 78 7e 41 00 9c 7e 41 00
0x00417B64  dc 7e 41 00 10 7f 41 00 01 00 00 00 00 00 00 00
0x00417B74  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0x00417B84  53 74 61 63 6b 20 61 72 6f 75 6e 64 20 74 68 65
0x00417B94  20 76 61 72 69 61 62 6c 65 20 27 00 27 20 77 61
0x00417BA4  73 20 63 6f 72 72 75 70 74 65 64 2e 00 00 00 00
```



```
B::foo:
004113B1 E9 5A 04 00 00       jmp         B::foo (0411810h)
B::foo:
004113B6 E9 55 04 00 00       jmp         B::foo (0411810h)
004113BB CC                   int         3
004113BC CC                   int         3
004113BD CC                   int         3
```

"偏移块"



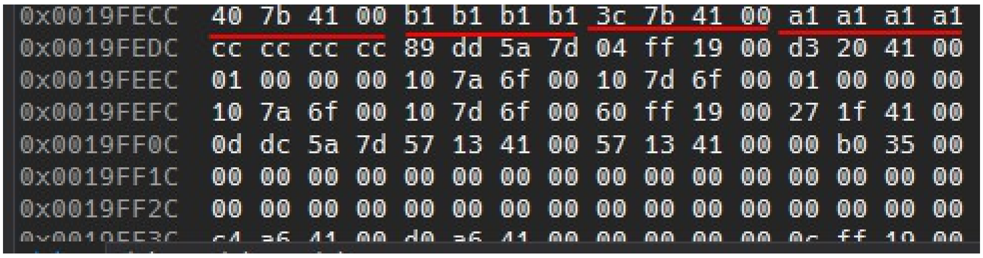
```
0x00417B34  b1 13 41 00 fc ff ff ff 08 00 00 00 00 00 00 00  ?
0x00417B44  00 00 00 00 f8 87 41 00 04 11 41 00 00 00 00 00  .
0x00417B54  10 7c 41 00 20 7d 41 00 78 7e 41 00 9c 7e 41 00  .
0x00417B64  dc 7e 41 00 10 7f 41 00 01 00 00 00 00 00 00 00  ?
0x00417B74  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00  .
0x00417B84  53 74 61 63 6b 20 61 72 6f 75 6e 64 20 74 68 65  S
0x00417B94  20 76 61 72 69 61 62 6c 65 20 27 00 27 20 77 61
0x00417BA4  73 20 63 6f 72 72 75 70 74 65 64 2e 00 00 00 00  s
```

其中`0xfffffffc`是本类对于“内存块指针”所在处的偏移量（为-4），`0x00000008`为虚基类与本“偏移块指针”所在处的偏移量（为8）

子类重写父类虚函数

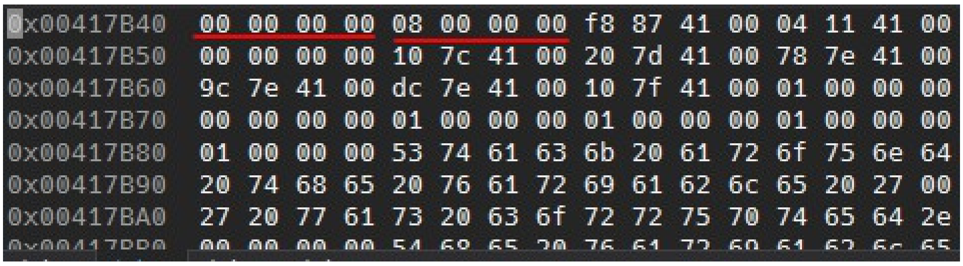
```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    virtual int foo()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};
```

依然虚基类在内存在末尾

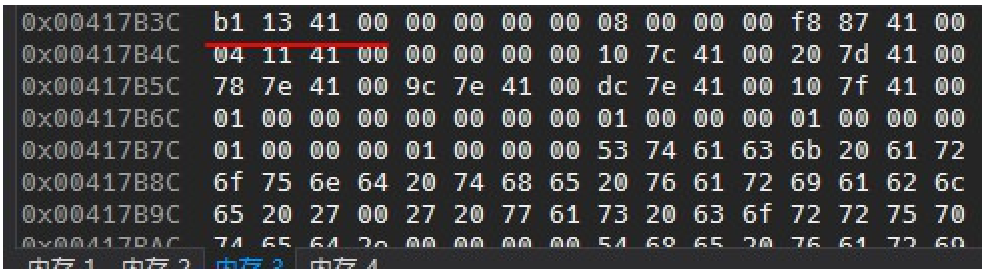


```
0x0019FECC  40 7b 41 00 b1 b1 b1 b1 3c 7b 41 00 a1 a1 a1 a1
0x0019FEDC  cc cc cc cc 89 dd 5a 7d 04 ff 19 00 d3 20 41 00
0x0019FEEC  01 00 00 00 10 7a 6f 00 10 7d 6f 00 01 00 00 00
0x0019FEFC  10 7a 6f 00 10 7d 6f 00 60 ff 19 00 27 1f 41 00
0x0019FF0C  0d dc 5a 7d 57 13 41 00 57 13 41 00 00 b0 35 00
0x0019FF1C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF2C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF3C  c4 a6 41 00 d0 a6 41 00 00 00 00 00 0c ff 19 00
```

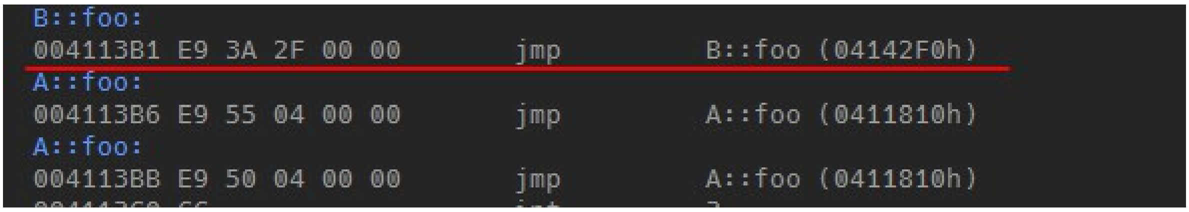
然后对象内存开始依次是“偏移块指针”、子类数据和虚基类虚表指针（子类虚函数附加到虚基类虚表中）



```
0x00417B40  00 00 00 00 08 00 00 00 f8 87 41 00 04 11 41 00
0x00417B50  00 00 00 00 10 7c 41 00 20 7d 41 00 78 7e 41 00
0x00417B60  9c 7e 41 00 dc 7e 41 00 10 7f 41 00 01 00 00 00
0x00417B70  00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0x00417B80  01 00 00 00 53 74 61 63 6b 20 61 72 6f 75 6e 64
0x00417B90  20 74 68 65 20 76 61 72 69 61 62 6c 65 20 27 00
0x00417BA0  27 20 77 61 73 20 63 6f 72 72 75 70 74 65 64 2e
0x00417BB0  00 00 00 00 54 68 65 20 76 61 72 69 61 62 6c 65
```



```
0x00417B3C  b1 13 41 00 00 00 00 00 08 00 00 00 f8 87 41 00
0x00417B4C  04 11 41 00 00 00 00 00 10 7c 41 00 20 7d 41 00
0x00417B5C  78 7e 41 00 9c 7e 41 00 dc 7e 41 00 10 7f 41 00
0x00417B6C  01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0x00417B7C  01 00 00 00 01 00 00 00 53 74 61 63 6b 20 61 72
0x00417B8C  6f 75 6e 64 20 74 68 65 20 76 61 72 69 61 62 6c
0x00417B9C  65 20 27 00 27 20 77 61 73 20 63 6f 72 72 75 70
0x00417BAC  74 65 64 2e 00 00 00 00 54 68 65 20 76 61 72 69
内存 1  内存 2  内存 3  内存 4
```

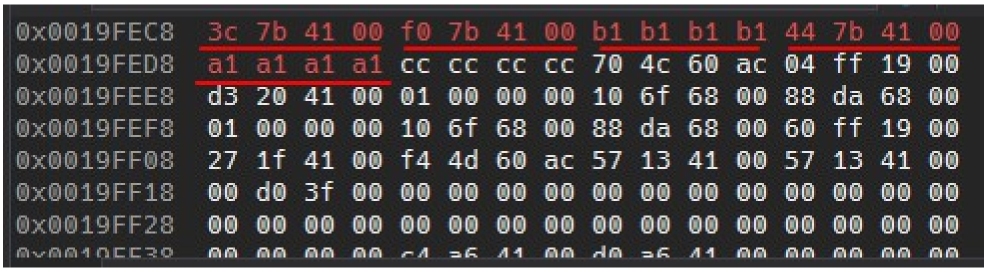


```
B::foo:
004113B1 E9 3A 2F 00 00       jmp         B::foo (04142F0h)
A::foo:
004113B6 E9 55 04 00 00       jmp         A::foo (0411810h)
A::foo:
004113BB E9 50 04 00 00       jmp         A::foo (0411810h)
004113C0 CC                   int         3
```

子类新加虚函数

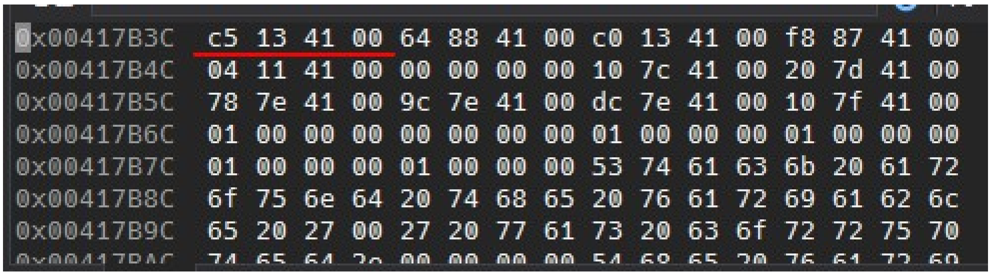
```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    virtual int foo()
    {
        return _b;
    }

    virtual int bar()
    {
        return _a;
    }

    int _b = 0xb1b1b1b1;
};
```

内存布局依次是：子类虚表指针、"偏移块指针"、子类数据、虚基类虚表指针和虚基类数据

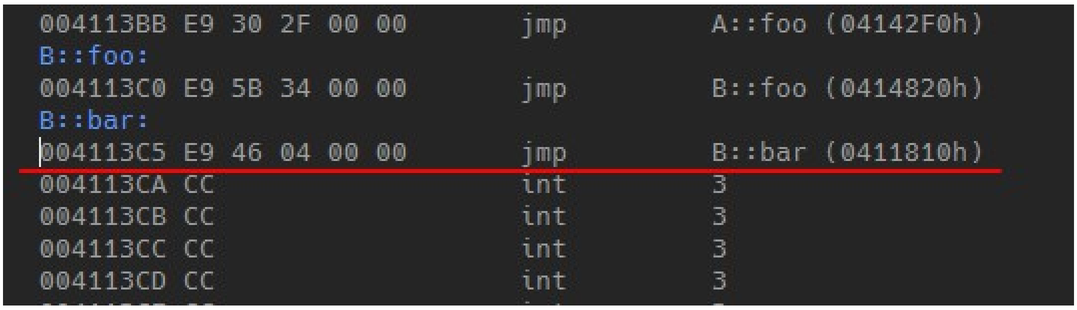


```
0x0019FEC8  3c 7b 41 00 f0 7b 41 00 b1 b1 b1 b1 44 7b 41 00
0x0019FED8  a1 a1 a1 a1 cc cc cc cc 70 4c 60 ac 04 ff 19 00
0x0019FEE8  d3 20 41 00 01 00 00 00 10 6f 68 00 88 da 68 00
0x0019FEF8  01 00 00 00 10 6f 68 00 88 da 68 00 60 ff 19 00
0x0019FF08  27 1f 41 00 f4 4d 60 ac 57 13 41 00 57 13 41 00
0x0019FF18  00 d0 3f 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF28  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF38  00 00 00 00 c4 a6 41 00 d0 a6 41 00 00 00 00 00
```

子类虚表如下：

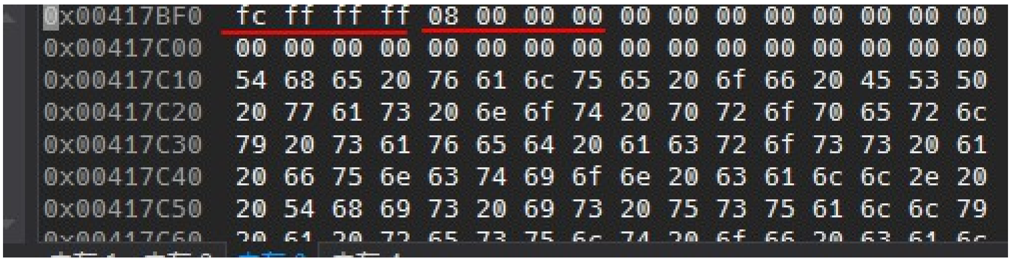


```
0x00417B3C  c5 13 41 00 64 88 41 00 c0 13 41 00 f8 87 41 00
0x00417B4C  04 11 41 00 00 00 00 00 10 7c 41 00 20 7d 41 00
0x00417B5C  78 7e 41 00 9c 7e 41 00 dc 7e 41 00 10 7f 41 00
0x00417B6C  01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0x00417B7C  01 00 00 00 01 00 00 00 53 74 61 63 6b 20 61 72
0x00417B8C  6f 75 6e 64 20 74 68 65 20 76 61 72 69 61 62 6c
0x00417B9C  65 20 27 00 27 20 77 61 73 20 63 6f 72 72 75 70
0x00417BAC  74 65 64 2e 00 00 00 00 54 68 65 20 76 61 72 69
```



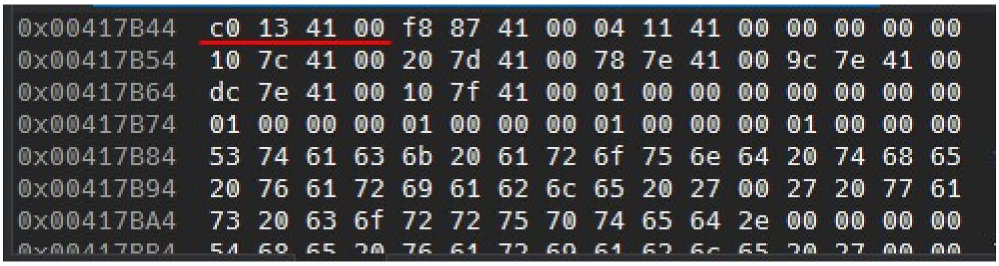
```
004113BB E9 30 2F 00 00        jmp          A::foo (04142F0h)
B::foo:
004113C0 E9 5B 34 00 00        jmp          B::foo (0414820h)
B::bar:
004113C5 E9 46 04 00 00        jmp          B::bar (0411810h)
004113CA CC                    int          3
004113CB CC                    int          3
004113CC CC                    int          3
004113CD CC                    int          3
```

"偏移块"如下：

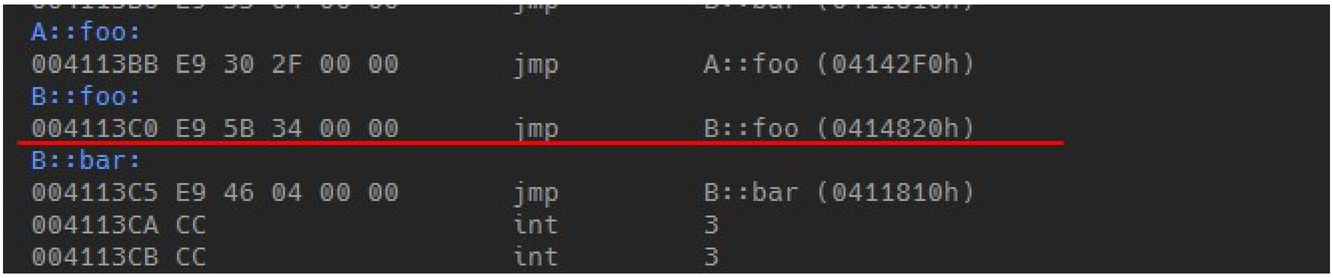


```
0x00417BF0  fc ff ff ff 08 00 00 00 00 00 00 00 00 00 00 00
0x00417C00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00417C10  54 68 65 20 76 61 6c 75 65 20 6f 66 20 45 53 50
0x00417C20  20 77 61 73 20 6e 6f 74 20 70 72 6f 70 65 72 6c
0x00417C30  79 20 73 61 76 65 64 20 61 63 72 6f 73 73 20 61
0x00417C40  20 66 75 6e 63 74 69 6f 6e 20 63 61 6c 6c 2e 20
0x00417C50  20 54 68 69 73 20 69 73 20 75 73 75 61 6c 6c 79
0x00417C60  20 61 20 72 65 73 75 6c 74 20 6f 66 20 63 61 6c
```

虚基类虚表如下：



```
0x00417B44  c0 13 41 00 f8 87 41 00 04 11 41 00 00 00 00 00
0x00417B54  10 7c 41 00 20 7d 41 00 78 7e 41 00 9c 7e 41 00
0x00417B64  dc 7e 41 00 10 7f 41 00 01 00 00 00 00 00 00 00
0x00417B74  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0x00417B84  53 74 61 63 6b 20 61 72 6f 75 6e 64 20 74 68 65
0x00417B94  20 76 61 72 69 61 62 6c 65 20 27 00 27 20 77 61
0x00417BA4  73 20 63 6f 72 72 75 70 74 65 64 2e 00 00 00 00
0x00417BB4  54 68 65 20 76 61 72 69 61 62 6c 65 20 27 00 00
```



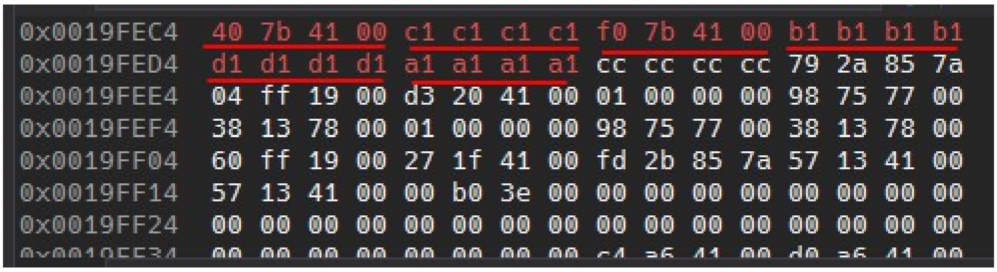
```
A::foo:
004113BB E9 30 2F 00 00       jmp          A::foo (04142F0h)
B::foo:
004113C0 E9 5B 34 00 00       jmp          B::foo (0414820h)
B::bar:
004113C5 E9 46 04 00 00       jmp          B::bar (0411810h)
004113CA CC                   int          3
004113CB CC                   int          3
```

多重虚继承

无虚函数

```
class A {
public:
    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    int _b = 0xb1b1b1b1;
};


class C : virtual public A {
public:
    int _c = 0xc1c1c1c1;
};

class D : public C, public B {
public:
    int _d = 0xd1d1d1d1;
};
```

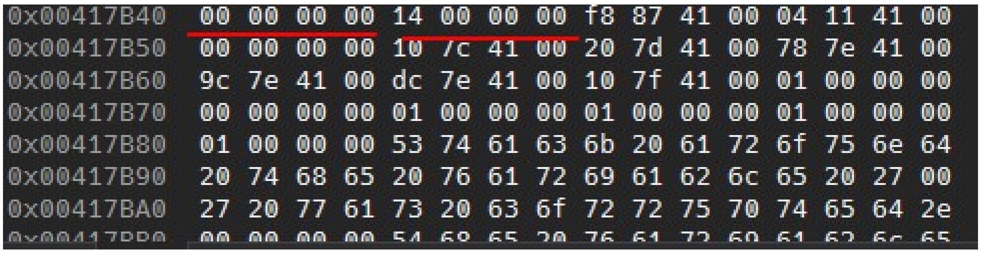
子类的内存分布为：第一顺位父类的“偏移块指针”、第一顺位父类数据、第二顺位父类的“偏移块指针”、第二顺位父类数据、子类数据和最后的虚基类数据



```
0x0019FEC4  40 7b 41 00 c1 c1 c1 c1 f0 7b 41 00 b1 b1 b1 b1
0x0019FED4  d1 d1 d1 d1 a1 a1 a1 a1 cc cc cc cc 79 2a 85 7a
0x0019FEE4  04 ff 19 00 d3 20 41 00 01 00 00 00 98 75 77 00
0x0019FEF4  38 13 78 00 01 00 00 00 98 75 77 00 38 13 78 00
0x0019FF04  60 ff 19 00 27 1f 41 00 fd 2b 85 7a 57 13 41 00
0x0019FF14  57 13 41 00 00 b0 3e 00 00 00 00 00 00 00 00 00
0x0019FF24  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF34  00 00 00 00 00 00 00 00 c4 a6 41 00 d0 a6 41 00
```

第一顺位父类的“偏移块”如下：

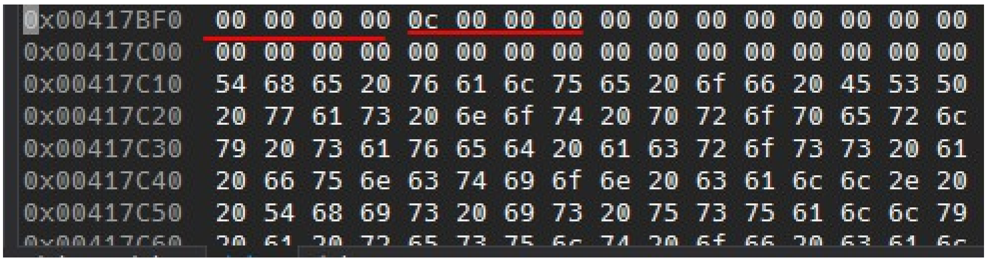
本类到本“偏移块指针”的偏移为`0x00000000`，虚基类与本“偏移块指针”的偏移为`0x00000014`



```
0x00417B40  00 00 00 00 14 00 00 00 f8 87 41 00 04 11 41 00
0x00417B50  00 00 00 00 10 7c 41 00 20 7d 41 00 78 7e 41 00
0x00417B60  9c 7e 41 00 dc 7e 41 00 10 7f 41 00 01 00 00 00
0x00417B70  00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0x00417B80  01 00 00 00 53 74 61 63 6b 20 61 72 6f 75 6e 64
0x00417B90  20 74 68 65 20 76 61 72 69 61 62 6c 65 20 27 00
0x00417BA0  27 20 77 61 73 20 63 6f 72 72 75 70 74 65 64 2e
0x00417BB0  00 00 00 00 54 68 65 20 76 61 72 69 61 62 6c 65
```

第二顺位父类的“偏移块”如下：

本类到本“偏移块指针”的偏移为`0x00000000`，虚基类与本“偏移块指针”的偏移为`0x0000000c`



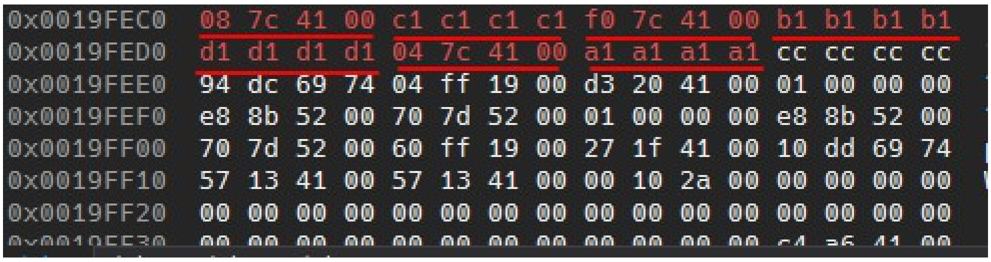
```
0x00417BF0  00 00 00 00 0c 00 00 00 00 00 00 00 00 00 00 00
0x00417C00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00417C10  54 68 65 20 76 61 6c 75 65 20 6f 66 20 45 53 50
0x00417C20  20 77 61 73 20 6e 6f 74 20 70 72 6f 70 65 72 6c
0x00417C30  79 20 73 61 76 65 64 20 61 63 72 6f 73 73 20 61
0x00417C40  20 66 75 6e 63 74 69 6f 6e 20 63 61 6c 6c 2e 20
0x00417C50  20 54 68 69 73 20 69 73 20 75 73 75 61 6c 6c 79
0x00417C60  20 61 20 72 65 73 75 6c 74 20 6f 66 20 63 61 6c
```

有虚函数

只有虚基类有虚函数

```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    int _b = 0xb1b1b1b1;
};


class C : virtual public A {
public:
    int _c = 0xc1c1c1c1;
};

class D : public C, public B {
public:
    int _d = 0xd1d1d1d1;
};
```

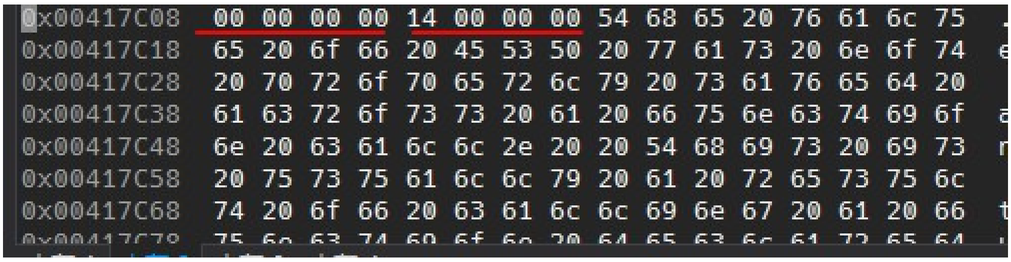
子类的内存分布为：第一顺位父类的“偏移块指针”、第一顺位父类数据、第二顺位父类的“偏移块指针”、第二顺位父类数据、子类数据、虚基类虚表指针和最后的虚基类数据



```
0x0019FEC0  08 7c 41 00 c1 c1 c1 c1 f0 7c 41 00 b1 b1 b1 b1
0x0019FED0  d1 d1 d1 d1 04 7c 41 00 a1 a1 a1 a1 cc cc cc cc
0x0019FEE0  94 dc 69 74 04 ff 19 00 d3 20 41 00 01 00 00 00
0x0019FEF0  e8 8b 52 00 70 7d 52 00 01 00 00 00 e8 8b 52 00
0x0019FF00  70 7d 52 00 60 ff 19 00 27 1f 41 00 10 dd 69 74
0x0019FF10  57 13 41 00 57 13 41 00 00 10 2a 00 00 00 00 00
0x0019FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF30  00 00 00 00 00 00 00 00 00 00 00 00 c4 a6 41 00
```

第一顺位父类的“偏移块”如下：

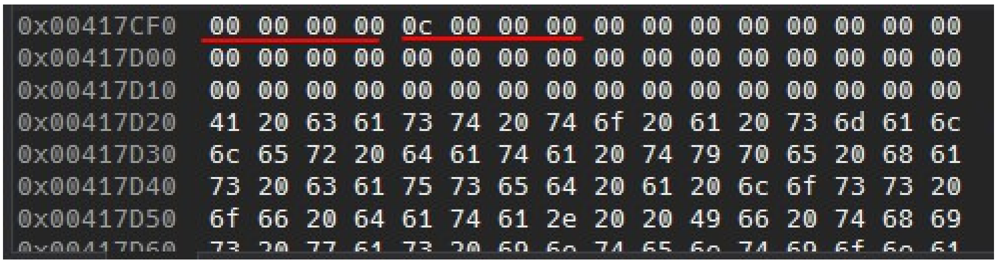
本类到本“偏移块指针”的偏移为`0x00000000`，虚基类与本“偏移块指针”的偏移为`0x00000014`



```
0x00417C08  00 00 00 00 14 00 00 00 54 68 65 20 76 61 6c 75
0x00417C18  65 20 6f 66 20 45 53 50 20 77 61 73 20 6e 6f 74
0x00417C28  20 70 72 6f 70 65 72 6c 79 20 73 61 76 65 64 20
0x00417C38  61 63 72 6f 73 73 20 61 20 66 75 6e 63 74 69 6f
0x00417C48  6e 20 63 61 6c 6c 2e 20 20 54 68 69 73 20 69 73
0x00417C58  20 75 73 75 61 6c 6c 79 20 61 20 72 65 73 75 6c
0x00417C68  74 20 6f 66 20 63 61 6c 6c 69 6e 67 20 61 20 66
0x00417C78  75 6e 63 74 69 6f 6e 20 64 65 63 6c 61 72 65 64
```

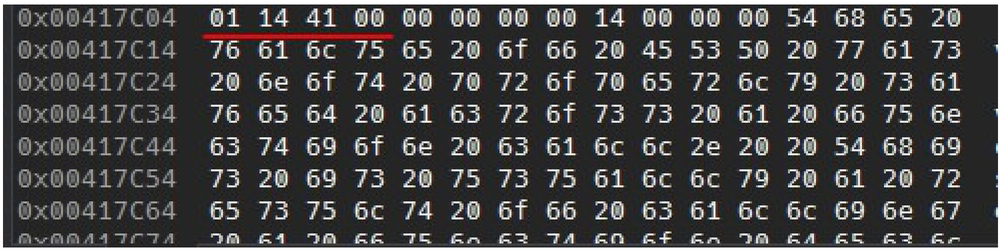
第二顺位父类的“偏移块”如下：

本类到本“偏移块指针”的偏移为`0x00000000`，虚基类与本“偏移块指针”的偏移为`0x0000000c`

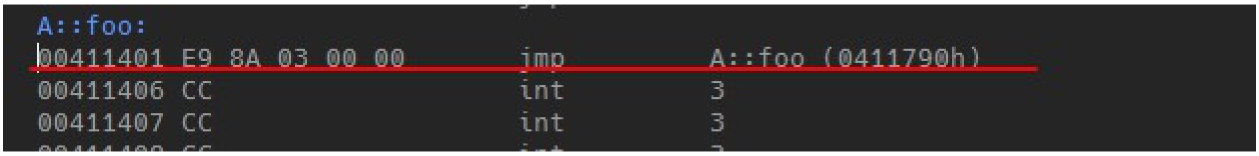


```
0x00417CF0  00 00 00 00 0c 00 00 00 00 00 00 00 00 00 00 00
0x00417D00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00417D10  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00417D20  41 20 63 61 73 74 20 74 6f 20 61 20 73 6d 61 6c
0x00417D30  6c 65 72 20 64 61 74 61 20 74 79 70 65 20 68 61
0x00417D40  73 20 63 61 75 73 65 64 20 61 20 6c 6f 73 73 20
0x00417D50  6f 66 20 64 61 74 61 2e 20 20 49 66 20 74 68 69
0x00417D60  73 20 77 61 73 20 69 6e 74 65 6e 74 69 6f 6e 61
```

虚基类虚表如下：



```
0x00417C04  01 14 41 00 00 00 00 00 14 00 00 00 54 68 65 20
0x00417C14  76 61 6c 75 65 20 6f 66 20 45 53 50 20 77 61 73
0x00417C24  20 6e 6f 74 20 70 72 6f 70 65 72 6c 79 20 73 61
0x00417C34  76 65 64 20 61 63 72 6f 73 73 20 61 20 66 75 6e
0x00417C44  63 74 69 6f 6e 20 63 61 6c 6c 2e 20 20 54 68 69
0x00417C54  73 20 69 73 20 75 73 75 61 6c 6c 79 20 61 20 72
0x00417C64  65 73 75 6c 74 20 6f 66 20 63 61 6c 6c 69 6e 67
0x00417C74  20 61 20 66 75 6e 63 74 69 6f 6e 20 64 65 63 6c
```



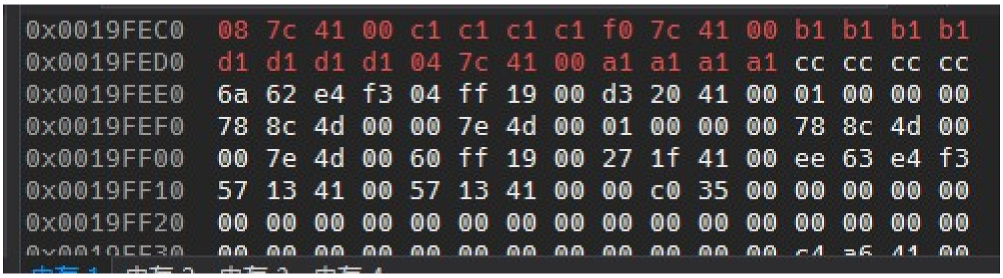
```
A::foo:
00411401 E9 8A 03 00 00       jmp         A::foo (0411790h)
00411406 CC                   int         3
00411407 CC                   int         3
00411408 CC                   int         3
```

中间类重写虚基类的虚函数

```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    virtual int foo()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};


class C : virtual public A {
public:
    int _c = 0xc1c1c1c1;
};

class D : public C, public B {
public:
    int _d = 0xd1d1d1d1;
};
```

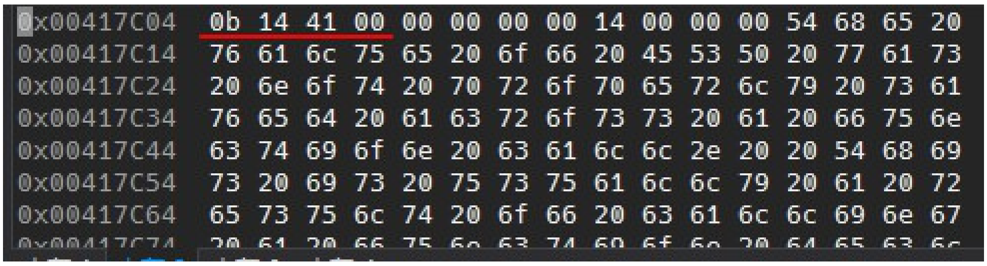
内存布局同上例一致，中间类的虚函数地址存放到虚基类的虚表中

注意：顺位受是否有虚表影响，有虚表则顺位提前

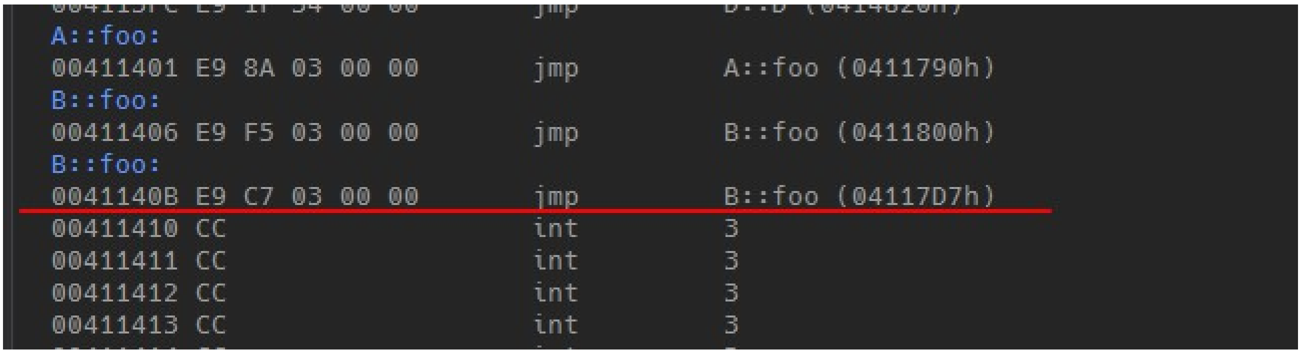


```
0x0019FEC0  08 7c 41 00 c1 c1 c1 c1 f0 7c 41 00 b1 b1 b1 b1
0x0019FED0  d1 d1 d1 d1 04 7c 41 00 a1 a1 a1 a1 cc cc cc cc
0x0019FEE0  6a 62 e4 f3 04 ff 19 00 d3 20 41 00 01 00 00 00
0x0019FEF0  78 8c 4d 00 00 7e 4d 00 01 00 00 00 78 8c 4d 00
0x0019FF00  00 7e 4d 00 60 ff 19 00 27 1f 41 00 ee 63 e4 f3
0x0019FF10  57 13 41 00 57 13 41 00 00 c0 35 00 00 00 00 00
0x0019FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF30  00 00 00 00 00 00 00 00 00 00 00 00 c4 a6 41 00
```

虚基类虚表如下：



```
0x00417C04  0b 14 41 00 00 00 00 00 14 00 00 00 54 68 65 20
0x00417C14  76 61 6c 75 65 20 6f 66 20 45 53 50 20 77 61 73
0x00417C24  20 6e 6f 74 20 70 72 6f 70 65 72 6c 79 20 73 61
0x00417C34  76 65 64 20 61 63 72 6f 73 73 20 61 20 66 75 6e
0x00417C44  63 74 69 6f 6e 20 63 61 6c 6c 2e 20 20 54 68 69
0x00417C54  73 20 69 73 20 75 73 75 61 6c 6c 79 20 61 20 72
0x00417C64  65 73 75 6c 74 20 6f 66 20 63 61 6c 6c 69 6e 67
0x00417C74  20 61 20 66 75 6e 63 74 69 6f 6e 20 64 65 63 6c
```

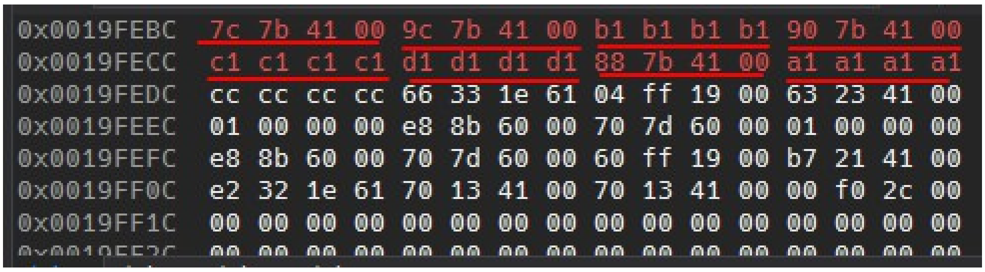


```
A::foo:
00411401 E9 8A 03 00 00       jmp         A::foo (0411790h)
B::foo:
00411406 E9 F5 03 00 00       jmp         B::foo (0411800h)
B::foo:
0041140B E9 C7 03 00 00       jmp         B::foo (04117D7h)
00411410 CC                   int         3
00411411 CC                   int         3
00411412 CC                   int         3
00411413 CC                   int         3
```

中间类新加虚函数

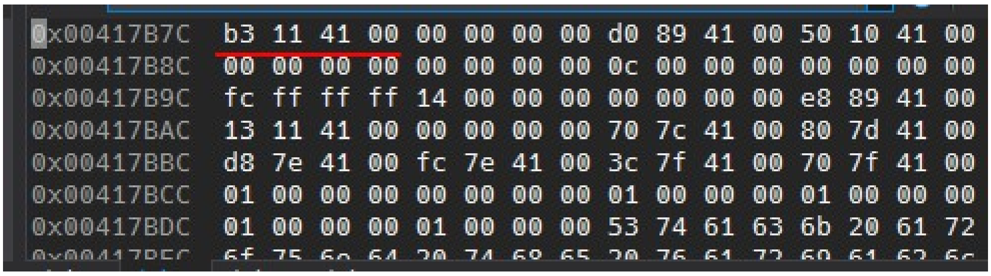
```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    virtual int foo()
    {
        return _b;
    }

    virtual int bar()
    {
        return _b;
    }

    int _b = 0xb1b1b1b1;
};


class C : virtual public A {
public:
    int _c = 0xc1c1c1c1;
};

class D : public C, public B {
public:
    int _d = 0xd1d1d1d1;
};
```

子类的内存分布为：第一顺位父类的虚表指针、第一顺位父类的"偏移块指针"、第一顺位父类数据、第二顺位父类的"偏移块指针"、第二顺位父类数据、子类数据、虚基类虚表指针和最后的虚基类数据

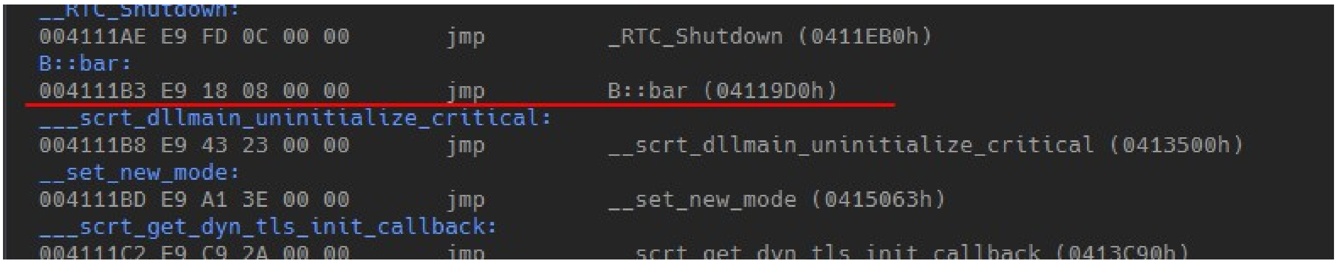


```
0x0019FEBC  7c 7b 41 00 9c 7b 41 00 b1 b1 b1 b1 90 7b 41 00
0x0019FECC  c1 c1 c1 c1 d1 d1 d1 d1 88 7b 41 00 a1 a1 a1 a1
0x0019FEDC  cc cc cc cc 66 33 1e 61 04 ff 19 00 63 23 41 00
0x0019FEEC  01 00 00 00 e8 8b 60 00 70 7d 60 00 01 00 00 00
0x0019FEFC  e8 8b 60 00 70 7d 60 00 60 ff 19 00 b7 21 41 00
0x0019FF0C  e2 32 1e 61 70 13 41 00 70 13 41 00 00 f0 2c 00
0x0019FF1C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF2C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

第一顺位父类虚表：

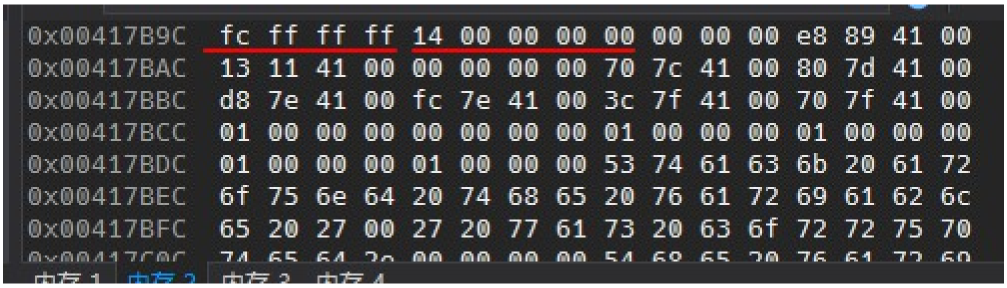


```
0x00417B7C  b3 11 41 00 00 00 00 00 d0 89 41 00 50 10 41 00
0x00417B8C  00 00 00 00 00 00 00 00 0c 00 00 00 00 00 00 00
0x00417B9C  fc ff ff ff 14 00 00 00 00 00 00 00 e8 89 41 00
0x00417BAC  13 11 41 00 00 00 00 00 70 7c 41 00 80 7d 41 00
0x00417BBC  d8 7e 41 00 fc 7e 41 00 3c 7f 41 00 70 7f 41 00
0x00417BCC  01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0x00417BDC  01 00 00 00 01 00 00 00 53 74 61 63 6b 20 61 72
0x00417BEC  6f 75 6e 64 20 74 68 65 20 76 61 72 69 61 62 6c
```



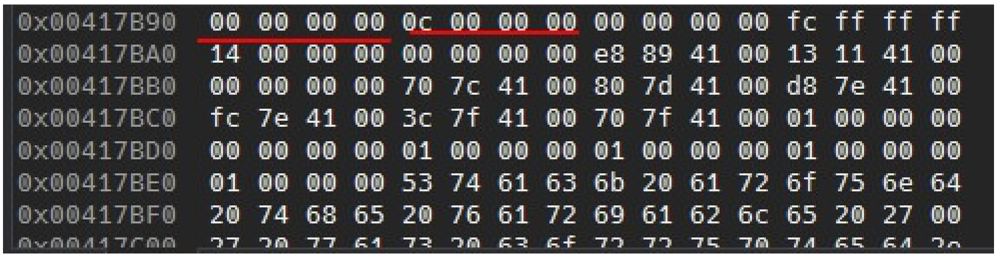
```
__RTC_Shutdown:
004111AE E9 FD 0C 00 00        jmp         _RTC_Shutdown (0411EB0h)
B::bar:
004111B3 E9 18 08 00 00        jmp         B::bar (04119D0h)
___scrt_dllmain_uninitialize_critical:
004111B8 E9 43 23 00 00        jmp         __scrt_dllmain_uninitialize_critical (0413500h)
__set_new_mode:
004111BD E9 A1 3E 00 00        jmp         __set_new_mode (0415063h)
___scrt_get_dyn_tls_init_callback:
004111C2 E9 C9 2A 00 00        jmp         __scrt_get_dyn_tls_init_callback (0413C90h)
```

第一顺位父类的“偏移块”：



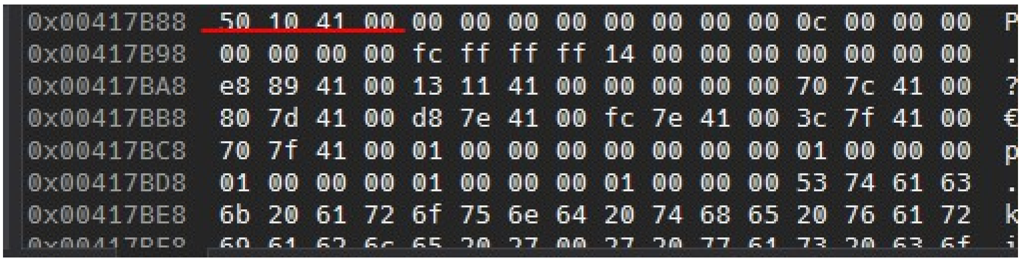
```
0x00417B9C  fc ff ff ff 14 00 00 00 00 00 00 00 e8 89 41 00
0x00417BAC  13 11 41 00 00 00 00 00 70 7c 41 00 80 7d 41 00
0x00417BBC  d8 7e 41 00 fc 7e 41 00 3c 7f 41 00 70 7f 41 00
0x00417BCC  01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0x00417BDC  01 00 00 00 01 00 00 00 53 74 61 63 6b 20 61 72
0x00417BEC  6f 75 6e 64 20 74 68 65 20 76 61 72 69 61 62 6c
0x00417BFC  65 20 27 00 27 20 77 61 73 20 63 6f 72 72 75 70
0x00417C0C  74 65 64 2e 00 00 00 00 54 68 65 20 76 61 72 69
```

第二顺位父类的“偏移块”：

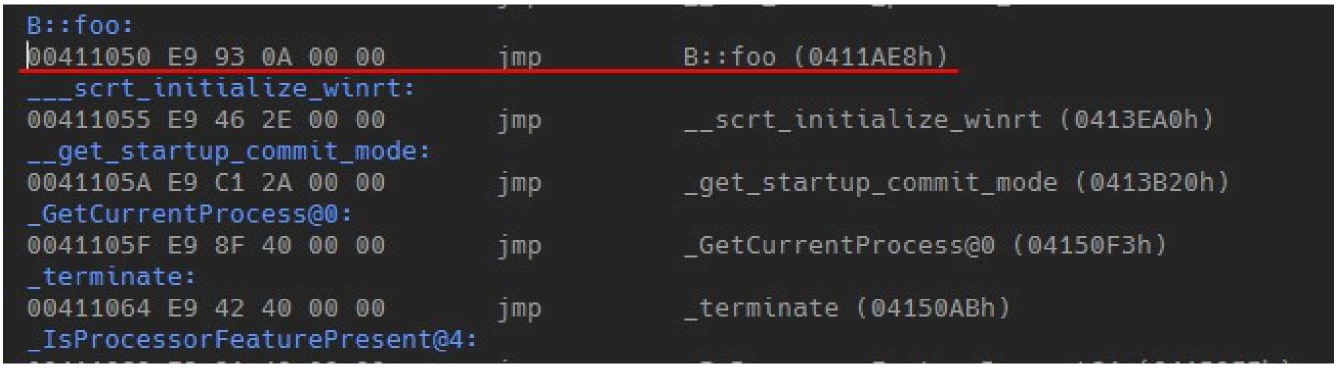


```
0x00417B90  00 00 00 00 0c 00 00 00 00 00 00 00 fc ff ff ff
0x00417BA0  14 00 00 00 00 00 00 00 e8 89 41 00 13 11 41 00
0x00417BB0  00 00 00 00 70 7c 41 00 80 7d 41 00 d8 7e 41 00
0x00417BC0  fc 7e 41 00 3c 7f 41 00 70 7f 41 00 01 00 00 00
0x00417BD0  00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0x00417BE0  01 00 00 00 53 74 61 63 6b 20 61 72 6f 75 6e 64
0x00417BF0  20 74 68 65 20 76 61 72 69 61 62 6c 65 20 27 00
0x00417C00  27 20 77 61 73 20 63 6f 72 72 75 70 74 65 64 2e
```

虚基类虚表：



```
0x00417B88  50 10 41 00 00 00 00 00 00 00 00 00 0c 00 00 00  P
0x00417B98  00 00 00 00 fc ff ff ff 14 00 00 00 00 00 00 00  .
0x00417BA8  e8 89 41 00 13 11 41 00 00 00 00 00 70 7c 41 00  ?
0x00417BB8  80 7d 41 00 d8 7e 41 00 fc 7e 41 00 3c 7f 41 00  €
0x00417BC8  70 7f 41 00 01 00 00 00 00 00 00 00 01 00 00 00  p
0x00417BD8  01 00 00 00 01 00 00 00 01 00 00 00 53 74 61 63  .
0x00417BE8  6b 20 61 72 6f 75 6e 64 20 74 68 65 20 76 61 72  k
0x00417BF8  69 61 62 6c 65 20 27 00 27 20 77 61 73 20 63 6f  i
```

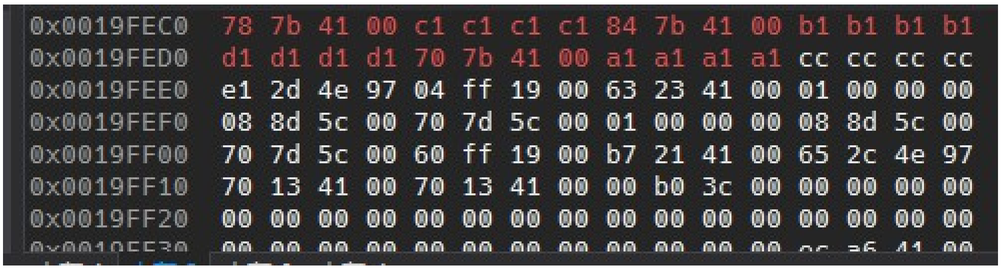


```
B::foo:
00411050 E9 93 0A 00 00       jmp         B::foo (0411AE8h)
___scrt_initialize_winrt:
00411055 E9 46 2E 00 00       jmp         __scrt_initialize_winrt (0413EA0h)
__get_startup_commit_mode:
0041105A E9 C1 2A 00 00       jmp         _get_startup_commit_mode (0413B20h)
_GetCurrentProcess@0:
0041105F E9 8F 40 00 00       jmp         _GetCurrentProcess@0 (04150F3h)
_terminate:
00411064 E9 42 40 00 00       jmp         _terminate (04150ABh)
_IsProcessorFeaturePresent@4:
```

重写的虚函数位于虚基类虚表中，新加的虚函数位于中间类自己的虚表中

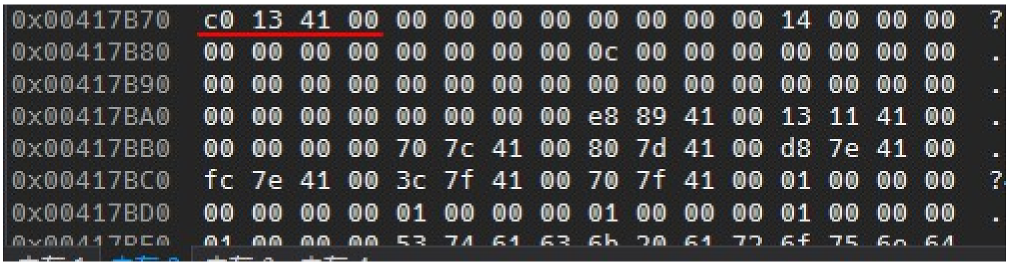
子类重写虚基类的虚函数

```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    int _b = 0xb1b1b1b1;
};


class C : virtual public A {
public:
    int _c = 0xc1c1c1c1;
};

class D : public C, public B {
public:
    virtual int foo()
    {
        return _d;
    }

    int _d = 0xd1d1d1d1;
};
```

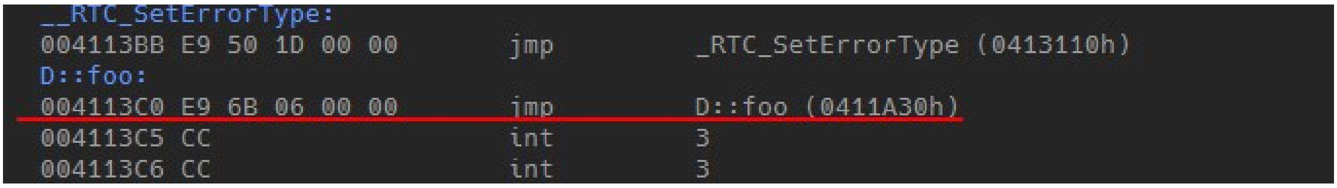


```
0x0019FEC0  78 7b 41 00 c1 c1 c1 c1 84 7b 41 00 b1 b1 b1 b1
0x0019FED0  d1 d1 d1 d1 70 7b 41 00 a1 a1 a1 a1 cc cc cc cc
0x0019FEE0  e1 2d 4e 97 04 ff 19 00 63 23 41 00 01 00 00 00
0x0019FEF0  08 8d 5c 00 70 7d 5c 00 01 00 00 00 08 8d 5c 00
0x0019FF00  70 7d 5c 00 60 ff 19 00 b7 21 41 00 65 2c 4e 97
0x0019FF10  70 13 41 00 70 13 41 00 00 b0 3c 00 00 00 00 00
0x0019FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

此时子类的虚函数地址会放到虚基类的虚表中



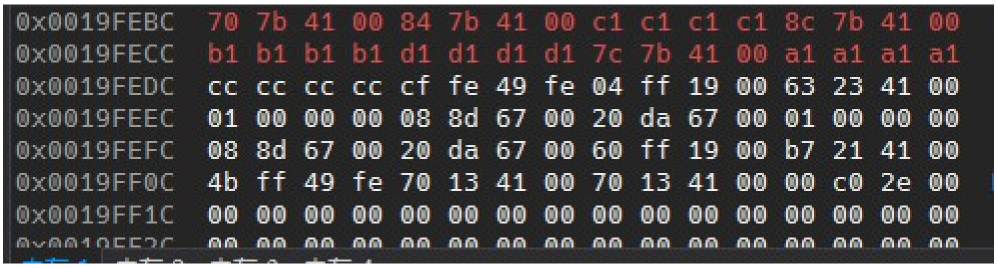
```
0x00417B70  c0 13 41 00 00 00 00 00 00 00 00 00 14 00 00 00  ?
0x00417B80  00 00 00 00 00 00 00 00 0c 00 00 00 00 00 00 00  .
0x00417B90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .
0x00417BA0  00 00 00 00 00 00 00 00 e8 89 41 00 13 11 41 00  .
0x00417BB0  00 00 00 00 70 7c 41 00 80 7d 41 00 d8 7e 41 00  .
0x00417BC0  fc 7e 41 00 3c 7f 41 00 70 7f 41 00 01 00 00 00  ?
0x00417BD0  00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00  .
0x00417BE0  01 00 00 00 53 74 61 63 6b 20 61 72 6f 75 6e 64
```



```
  __RTC_SetErrorType:
  004113BB E9 50 1D 00 00       jmp          _RTC_SetErrorType (0413110h)
  D::foo:
  004113C0 E9 6B 06 00 00       jmp          D::foo (0411A30h)
  004113C5 CC                   int          3
  004113C6 CC                   int          3
```

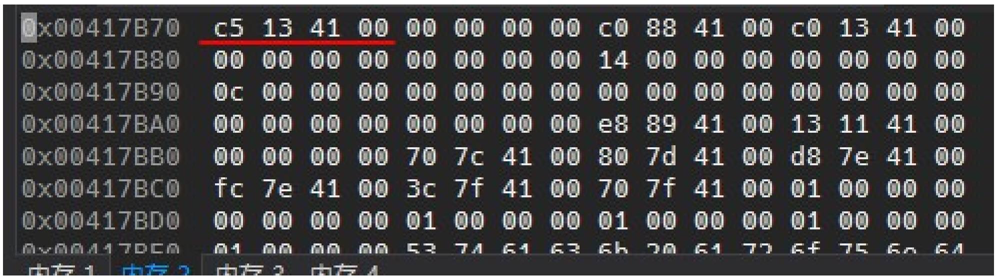
子类新加虚函数

```
class A {
public:
    virtual int foo()
    {
        return _a;
    }

    int _a = 0xa1a1a1a1;
};

class B : virtual public A {
public:
    int _b = 0xb1b1b1b1;
};


class C : virtual public A {
public:
    int _c = 0xc1c1c1c1;
};

class D : public C, public B {
public:
    virtual int foo()
    {
        return _d;
    }

    virtual int bar()
    {
        return _d;
    }

    int _d = 0xd1d1d1d1;
};
```

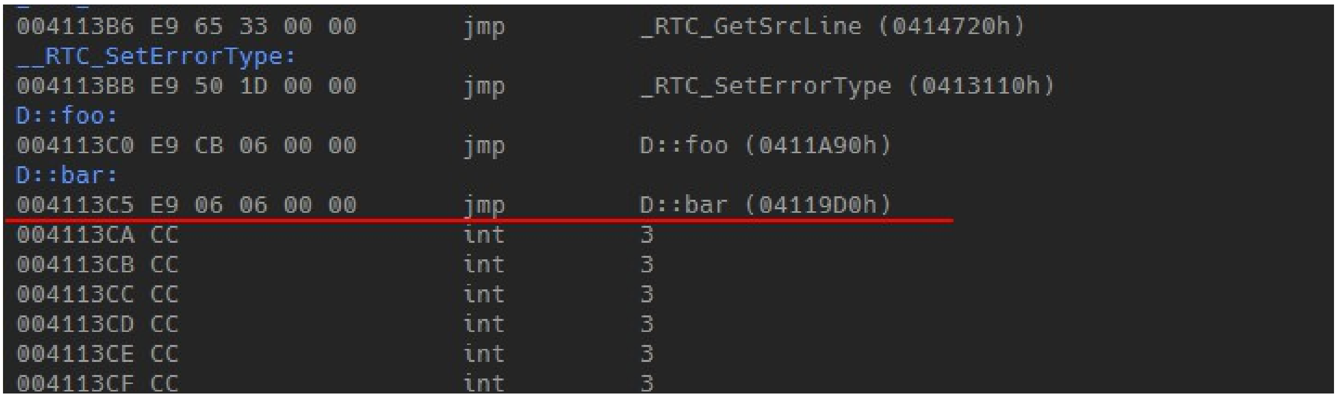


```
0x0019FEBC  70 7b 41 00 84 7b 41 00 c1 c1 c1 c1 8c 7b 41 00
0x0019FECC  b1 b1 b1 b1 d1 d1 d1 d1 7c 7b 41 00 a1 a1 a1 a1
0x0019FEDC  cc cc cc cc cf fe 49 fe 04 ff 19 00 63 23 41 00
0x0019FEEC  01 00 00 00 08 8d 67 00 20 da 67 00 01 00 00 00
0x0019FEFC  08 8d 67 00 20 da 67 00 60 ff 19 00 b7 21 41 00
0x0019FF0C  4b ff 49 fe 70 13 41 00 70 13 41 00 00 c0 2e 00
0x0019FF1C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF2C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

子类新加虚函数挂靠到第一顺位父类的虚表中



```
0x00417B70  c5 13 41 00 00 00 00 00 c0 88 41 00 c0 13 41 00
0x00417B80  00 00 00 00 00 00 00 00 14 00 00 00 00 00 00 00
0x00417B90  0c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00417BA0  00 00 00 00 00 00 00 00 e8 89 41 00 13 11 41 00
0x00417BB0  00 00 00 00 70 7c 41 00 80 7d 41 00 d8 7e 41 00
0x00417BC0  fc 7e 41 00 3c 7f 41 00 70 7f 41 00 01 00 00 00
0x00417BD0  00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0x00417BE0  01 00 00 00 53 74 61 63 6b 20 61 72 6f 75 6e 64
```



```
004113B6 E9 65 33 00 00       jmp          _RTC_GetSrcLine (0414720h)
__RTC_SetErrorType:
004113BB E9 50 1D 00 00       jmp          _RTC_SetErrorType (0413110h)
D::foo:
004113C0 E9 CB 06 00 00       jmp          D::foo (0411A90h)
D::bar:
004113C5 E9 06 06 00 00       jmp          D::bar (04119D0h)
004113CA CC                   int          3
004113CB CC                   int          3
004113CC CC                   int          3
004113CD CC                   int          3
004113CE CC                   int          3
004113CF CC                   int          3
```