示例代码:

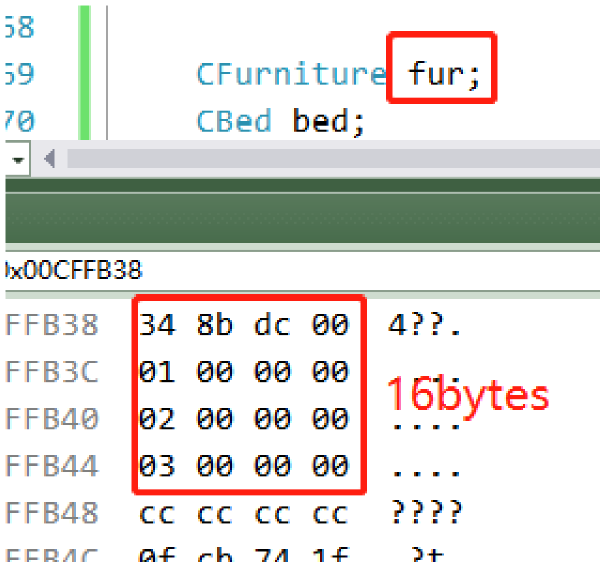
```cpp
/*
1. 单重继承有虚函数
*/

#include "stdafx.h"
#include <iostream>
using namespace std;

// 家具类
class CFurniture {
public:
    CFurniture() {
        printf("CFurniture::CFurniture()\r\n");
        m_nFurniture = 1;
        m_nFurniture2 = 2;
        m_nFurniture3 = 3;
    }

    ~CFurniture() {
        printf("CFurniture::~CFurniture()\r\n");
        m_nFurniture = 0;
    }

    virtual void sleep() {
        printf("CFurniture::sleep()\r\n");
    }

    void sit() {
        printf("CFurniture::sit()\r\n");
    }

    int m_nFurniture;

protected:
    int m_nFurniture2;

private:
    int m_nFurniture3;
```

```cpp
};

// 床类
class CBed : public CFurniture {
public:
    CBed() {
        printf("CBed::CBed()\r\n");
        m_nBed = 2;
    }

    ~CBed() {
        printf("CBed::~CBed()\r\n");
        m_nBed = 0;
    }

    virtual void sleep() {
        printf("CBed::sleep()\r\n");
    }

    void sit() {
        printf("CBed::sit()\r\n");
    }

private:
    int m_nBed;
};

int main()
{

    CFurniture fur;
    CBed bed;

    int nSizeFur = sizeof(CFurniture);
    cout << nSizeFur << endl; // 16
    int nSizeBed = sizeof(CBed);
    cout << nSizeBed << endl; // 20

    // 向上转型
    // 1 基类对象指向派生类对象
    fur = bed;
    fur.sit(); // normal function
    fur.sleep(); // virtual function
    // 这种方式不会发生虚函数调用，依然是调用各自类中的成员函数。对象赋值对成员对象和this指针无影响。

    // 2 基类指针指向派生类对象的地址
    CFurniture* pfur = &bed;
    pfur->sit(); // normal function
    pfur->sleep();// virtual function
    // 由于基类通过指针调用成员函数，而sleep()成员函数同时也是虚函数，这时会发生虚调用，间接调用指针指向的对象的虚函数。

    // 3 基类指针通过引用指向派生类对象
    CFurniture &rfur = bed;
    rfur.sit(); // normal function
    rfur.sleep(); // virtual function
    // 由于基类通过引用调用成员函数，而sleep()成员函数同时也是虚函数，发生虚调用，间接调用引用的那个对象的虚函数。

    return 0;
}
/*
output:

CFurniture::CFurniture()
CFurniture::CFurniture()
CBed::CBed()
16
20
CFurniture::sit()
CFurniture::sleep()
CFurniture::sit()
CBed::sleep()
CFurniture::sit()
CBed::sleep()
CBed::~CBed()
CFurniture::~CFurniture()
CFurniture::~CFurniture()
*/
```

对象大小

```
    int nSizeFur = sizeof(CFurniture);
    cout << nSizeFur << endl; // 16
    int nSizeBed = sizeof(CBed);
    cout << nSizeBed << endl; // 20
```

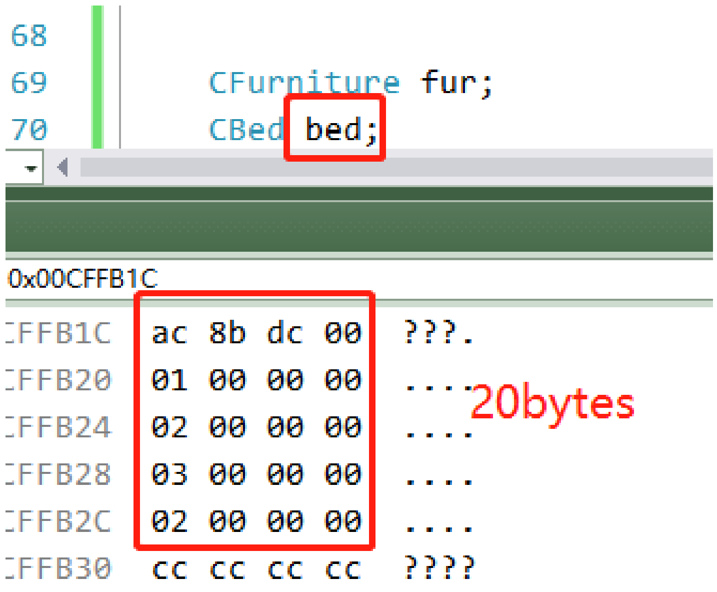
`CFurniture`



```
    CFurniture fur;
    CBed bed;
```

```
x00CFFB38
FFB38  34 8b dc 00  4??.
FFB3C  01 00 00 00  ....
FFB40  02 00 00 00  ....
FFB44  03 00 00 00  ....
FFB48  cc cc cc cc  ????
```

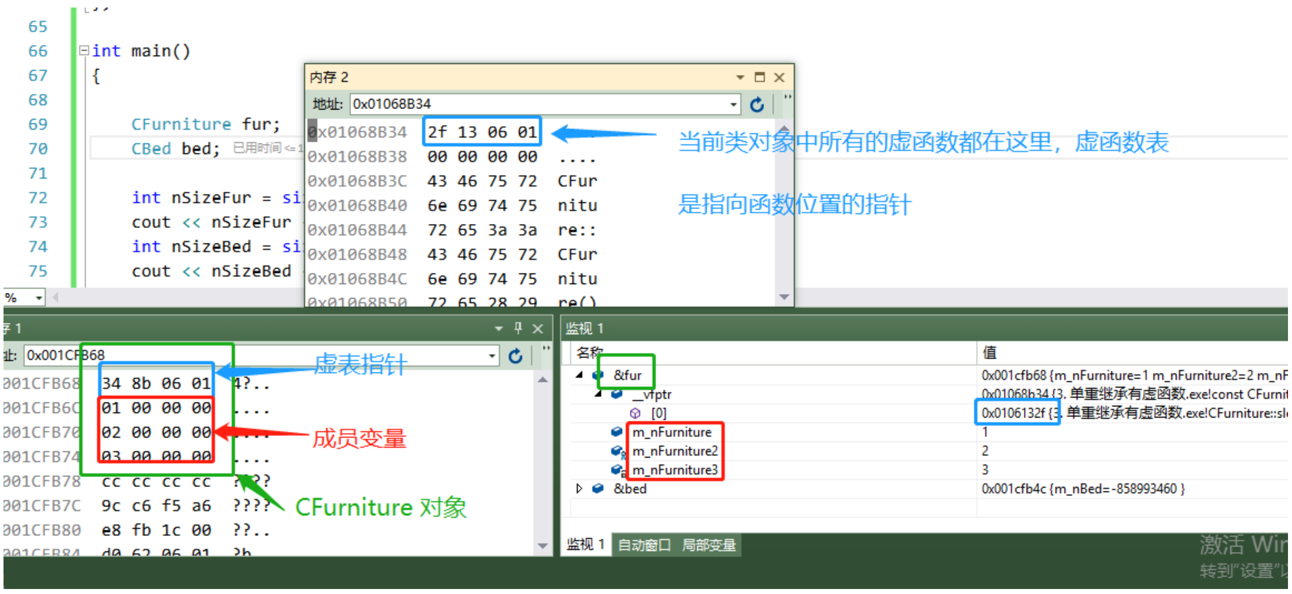
`CBed`



```
    CFurniture fur;
    CBed bed;
```

```
0x00CFFB1C
FFB1C  ac 8b dc 00  ???.
FFB20  01 00 00 00  ....
FFB24  02 00 00 00  ....
FFB28  03 00 00 00  ....
FFB2C  02 00 00 00  ....
FFB30  cc cc cc cc  ????
```

对象内存分析



```
int main()
{

    CFurniture fur;
    CBed bed;

    int nSizeFur = si
    cout << nSizeFur
    int nSizeBed = si
    cout << nSizeBed
```

```
内存 2
地址: 0x01068B34
0x01068B34  2f 13 06 01
0x01068B38  00 00 00 00  ....
0x01068B3C  43 46 75 72  CFur
0x01068B40  6e 69 74 75  nitu
0x01068B44  72 65 3a 3a  re::
0x01068B48  43 46 75 72  CFur
0x01068B4C  6e 69 74 75  nitu
0x01068B50  72 65 28 29  re()
```

```
存 1
址: 0x001CFB68
001CFB68  34 8b 06 01  4?..
001CFB6C  01 00 00 00  ....
001CFB70  02 00 00 00  ....
001CFB74  03 00 00 00  ....
001CFB78  cc cc cc cc  ????
001CFB7C  9c c6 f5 a6  ????
001CFB80  e8 fb 1c 00  ??..
001CFB84  d0 62 06 01  ?b..
```

名称	值
&fur	0x001cfb68 {m_nFurniture=1 m_nFurniture2=2 m_nF
__vfptr	0x01068b34 {3. 单重继承有虚函数.exe!const CFurnit
[0]	0x0106132f {3. 单重继承有虚函数.exe!CFurniture::sl
m_nFurniture	1
m_nFurniture2	2
m_nFurniture3	3
&bed	0x001cfb4c {m_nBed=-858993460 }

函数调用

见代码示例。