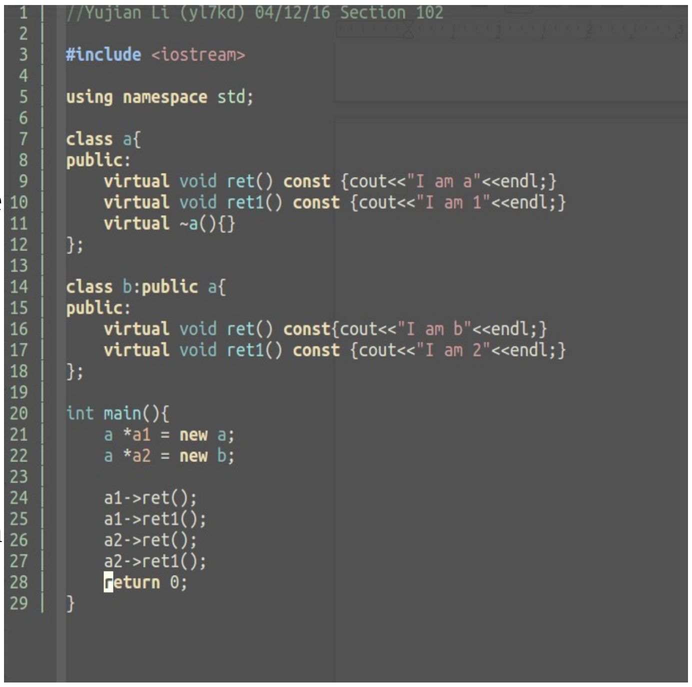
//Yujian Li (yl7kd) 04/12/16 Section 102

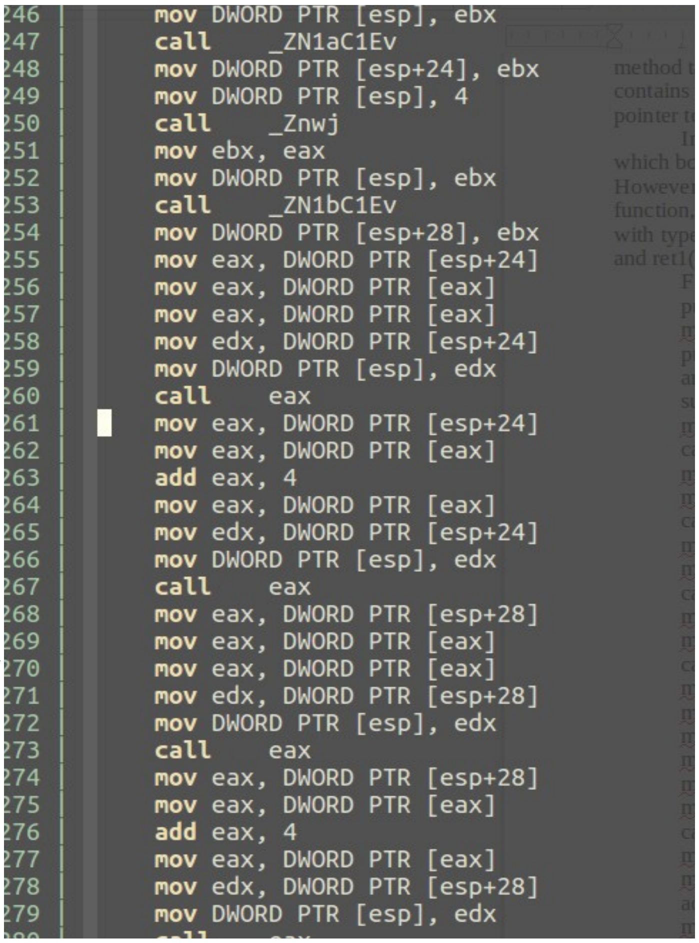
In Lab Report

As we have discussed in class, the nature of dynamic dispatch makes the complier does not know which member function to invoke until it generates the code. Before runtime, all the virtual function's address are stored inside a virtual method table waiting to be invoked, and the object, which contains the pointer to the virtual method table shall call the pointer to finish the implementation.



```cpp
//Yujian Li (yl7kd) 04/12/16 Section 102

#include <iostream>

using namespace std;

class a{
public:
    virtual void ret() const {cout<<"I am a"<<endl;}
    virtual void ret1() const {cout<<"I am 1"<<endl;}
    virtual ~a(){}
};

class b:public a{
public:
    virtual void ret() const{cout<<"I am b"<<endl;}
    virtual void ret1() const {cout<<"I am 2"<<endl;}
};

int main(){
    a *a1 = new a;
    a *a2 = new b;

    a1->ret();
    a1->ret1();
    a2->ret();
    a2->ret1();
    return 0;
}
```

In the snippet I created, I built two objects a, and b which both contains two virtual functions ret() and ret1(). However, these two have different behaviors. In the main function, I created a “a1” object with type a and “a2” object with type b. Later I invoked the two member functions (ret() and ret1()) inside two objects.



```
mov DWORD PTR [esp], ebx
call    _ZN1aC1Ev
mov DWORD PTR [esp+24], ebx
mov DWORD PTR [esp], 4
call    _Znwj
mov ebx, eax
mov DWORD PTR [esp], ebx
call    _ZN1bC1Ev
mov DWORD PTR [esp+28], ebx
mov eax, DWORD PTR [esp+24]
mov eax, DWORD PTR [eax]
mov eax, DWORD PTR [eax]
mov edx, DWORD PTR [esp+24]
mov DWORD PTR [esp], edx
call    eax
mov eax, DWORD PTR [esp+24]
mov eax, DWORD PTR [eax]
add eax, 4
mov eax, DWORD PTR [eax]
mov edx, DWORD PTR [esp+24]
mov DWORD PTR [esp], edx
call    eax
mov eax, DWORD PTR [esp+28]
mov eax, DWORD PTR [eax]
mov eax, DWORD PTR [eax]
mov edx, DWORD PTR [esp+28]
mov DWORD PTR [esp], edx
call    eax
mov eax, DWORD PTR [esp+28]
mov eax, DWORD PTR [eax]
add eax, 4
mov eax, DWORD PTR [eax]
mov edx, DWORD PTR [esp+28]
mov DWORD PTR [esp], edx
```

From the assembly code complied, I found that after the assembly has invoked the function to create two objects respectively, it then move the DWORD PTR [esp+24] to the register eax. As mentioned before, the address toward the virtual method table. Hence moving the pointer at esp+24 shall bring us to the virtual method table. Then it takes the pointer address of eax which is the address in the virtual
method table that pointing toward the actual member function. In the end, it will take the address of the member function “mov eax, DWORD PTR [eax]” and call the function “call eax”. When invoking the second function inside the object, it will do the same operation as before but when reaching the virtual method table, it will add “4” to the register eax since all member functions inside the virtual method table are located in order by 4 byte difference. When it tried to call a different object, it will then do the same process again with a virtual method table address stored in the object (DWORD PTR [esp+24] for object a and DWORD PTR [esp+28] for object b]). As we

have mentioned before, the calling process for dynamic dispatch is different from the previous method calling techniques since the method's name is never shown but represented by a pointer instead. Since the complier will not know which function to invoker, it is important to use the virtual method table to point toward the actual method and connect with the actual object or it will be impossible for the code to know which method to invoke in the end.