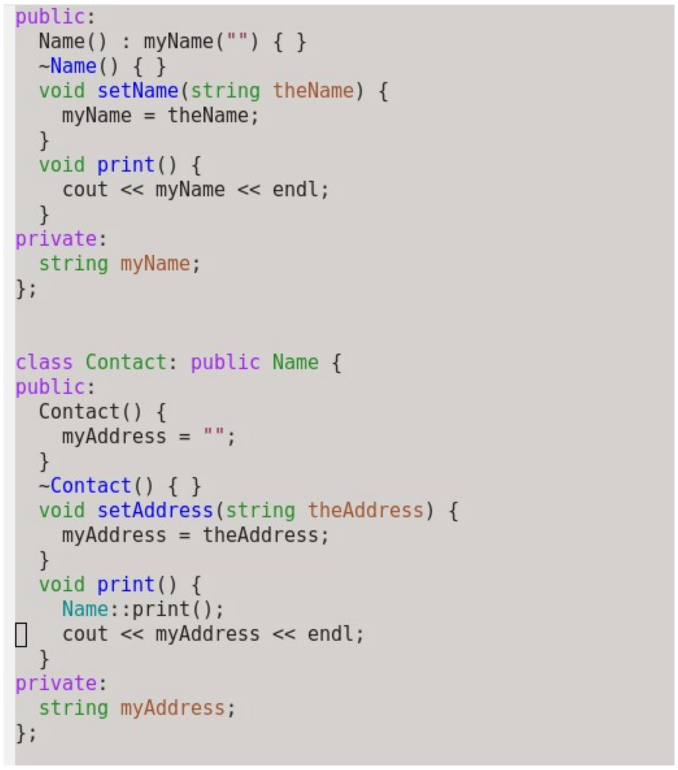
Maddie Stigler
mgs4ff
11/18/14
inlab9.pdf

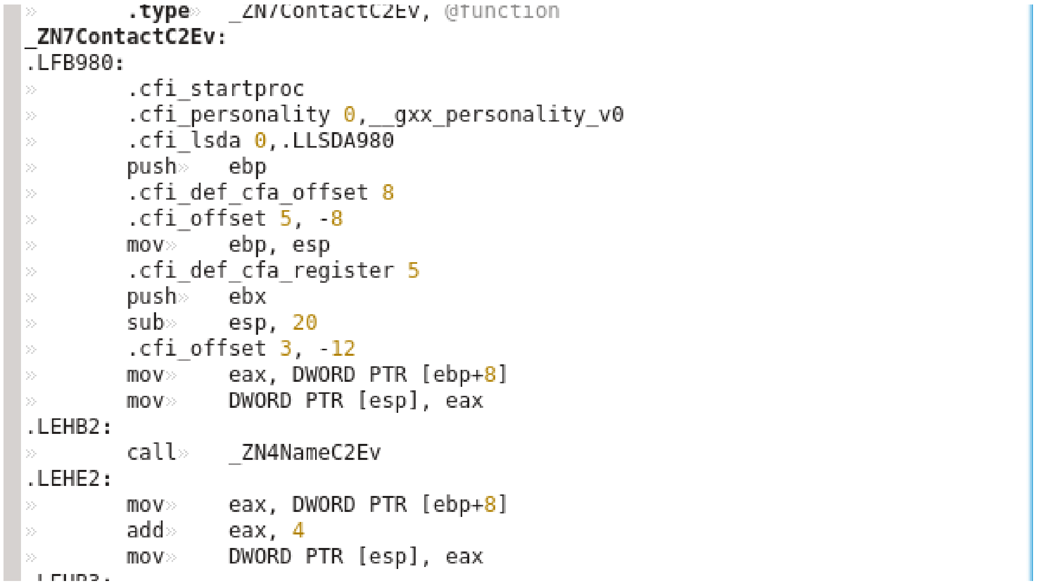
1. Inheritance

In order to examine how inheritance is received in x86 assembly code, I created an instance of inheritance through the use of a Contact class inheriting a Name class. Both classes have a constructor, destructor, print method, and a private field. In the main method, I created an instance of a contact, initialized both of its private fields and then called its print method. The data member that the Contact class inherits is the name field, and the data member that it includes on its own is the address field. Both are printed in the print function.



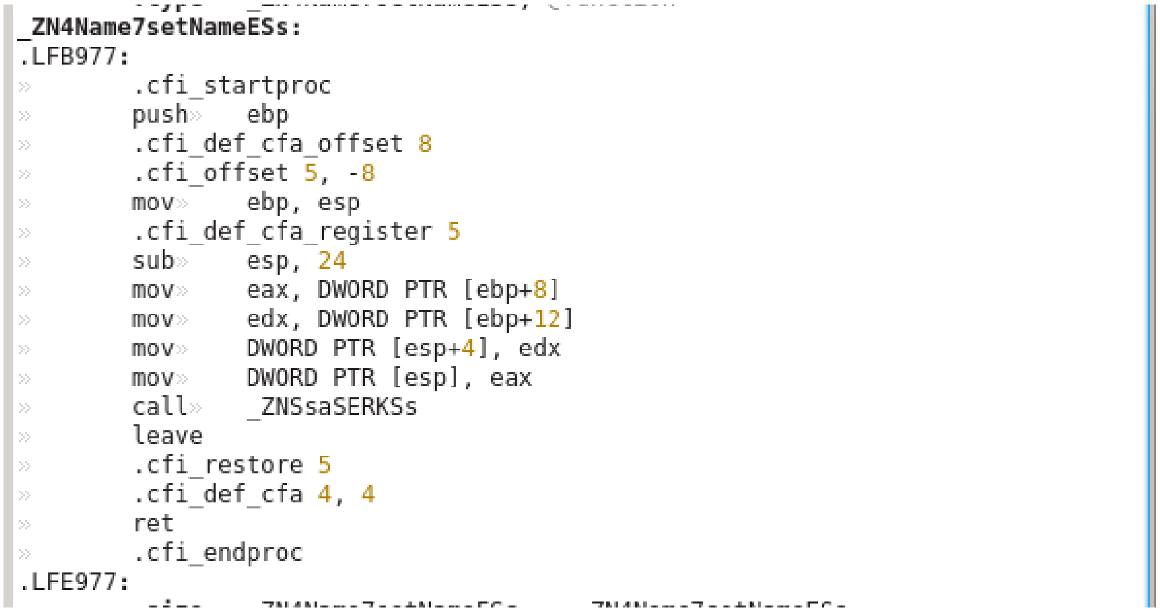
```
public:
  Name() : myName("") { }
  ~Name() { }
  void setName(string theName) {
    myName = theName;
  }
  void print() {
    cout << myName << endl;
  }
private:
  string myName;
};


class Contact: public Name {
public:
  Contact() {
    myAddress = "";
  }
  ~Contact() { }
  void setAddress(string theAddress) {
    myAddress = theAddress;
  }
  void print() {
    Name::print();
    cout << myAddress << endl;
  }
private:
  string myAddress;
};
```

When converted to assembly using the g++ compiler, the resulting assembly code is used to illustrate where in memory data members are stored for the Contact object. First, space is allocated on the stack to store the two data fields initiated in the main method. This space is used to store both the name field inherited from the super class(Name) and the address field defined in the derived class(Contact). This is shown under the Contact label and again in the field label:



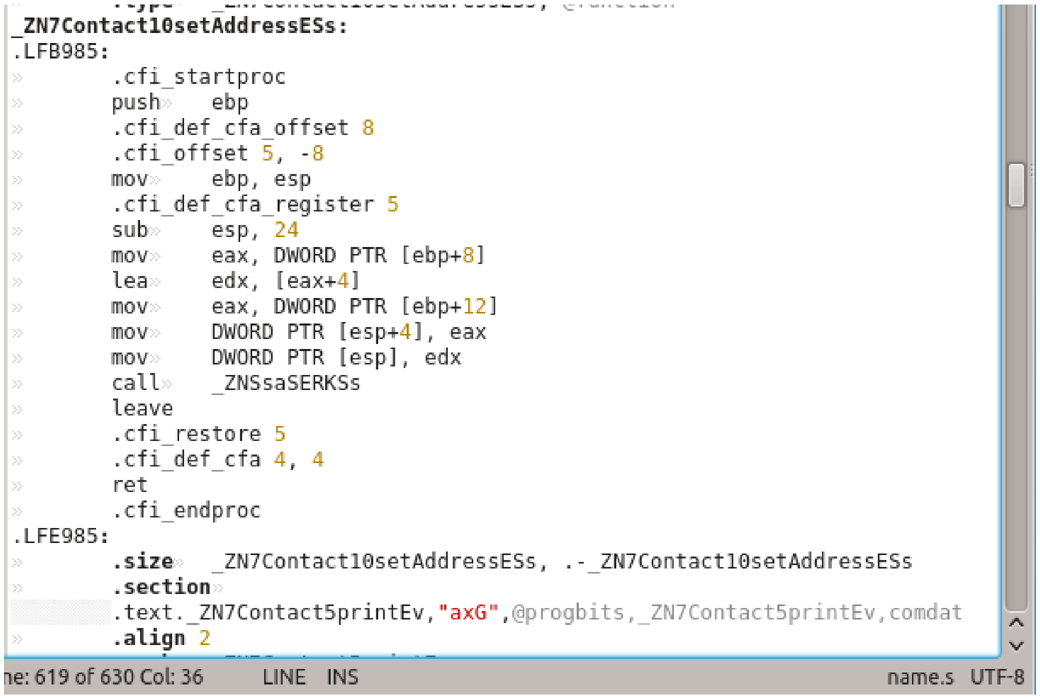
```
»       .type»  _ZN7ContactC2Ev, @function
_ZN7ContactC2Ev:
.LFB980:
»       .cfi_startproc
»       .cfi_personality 0,__gxx_personality_v0
»       .cfi_lsda 0,.LLSDA980
»       push»   ebp
»       .cfi_def_cfa_offset 8
»       .cfi_offset 5, -8
»       mov»    ebp, esp
»       .cfi_def_cfa_register 5
»       push»   ebx
»       sub»    esp, 20
»       .cfi_offset 3, -12
»       mov»    eax, DWORD PTR [ebp+8]
»       mov»    DWORD PTR [esp], eax
.LEHB2:
»       call»   _ZN4NameC2Ev
.LEHE2:
»       mov»    eax, DWORD PTR [ebp+8]
»       add»    eax, 4
»       mov»    DWORD PTR [esp], eax
```

*Contact allocates space on the stack and then calls the Name label that contains the Name field.



```
_ZN4Name7setNameESs:
.LFB977:
»       .cfi_startproc
»       push»   ebp
»       .cfi_def_cfa_offset 8
»       .cfi_offset 5, -8
»       mov»    ebp, esp
»       .cfi_def_cfa_register 5
»       sub»    esp, 24
»       mov»    eax, DWORD PTR [ebp+8]
»       mov»    edx, DWORD PTR [ebp+12]
»       mov»    DWORD PTR [esp+4], edx
»       mov»    DWORD PTR [esp], eax
»       call»   _ZNSsaSERKSs
»       leave
»       .cfi_restore 5
»       .cfi_def_cfa 4, 4
»       ret
»       .cfi_endproc
.LFE977:
```

*The name field is allocated and stored on the stack. This field is inherited by Contact from the Name class



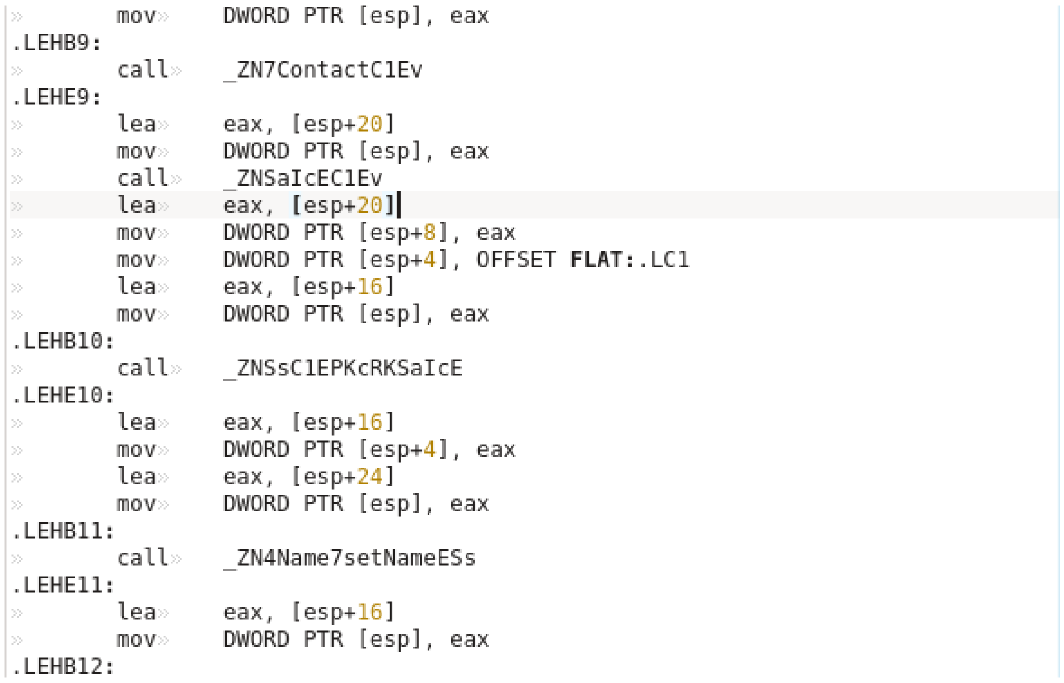
```
_ZN7Contact10setAddressESs:
.LFB985:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 24
        mov     eax, DWORD PTR [ebp+8]
        lea     edx, [eax+4]
        mov     eax, DWORD PTR [ebp+12]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], edx
        call    _ZNSsaSERKSs
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE985:
        .size   _ZN7Contact10setAddressESs, .-_ZN7Contact10setAddressESs
        .section
        .text._ZN7Contact5printEv,"axG",@progbits,_ZN7Contact5printEv,comdat
        .align 2
```

*Space is also allocated for the derived class's(Contact) fields and used to store the address.

When examining the construction and destruction of objects in assembly, I notice it is relative to the class hierarchy. Initially, the base class's constructor is called to initialize the data members inherited from the base class. Then, the derived class's constructor is then called to initialize the data members added in the derived class.

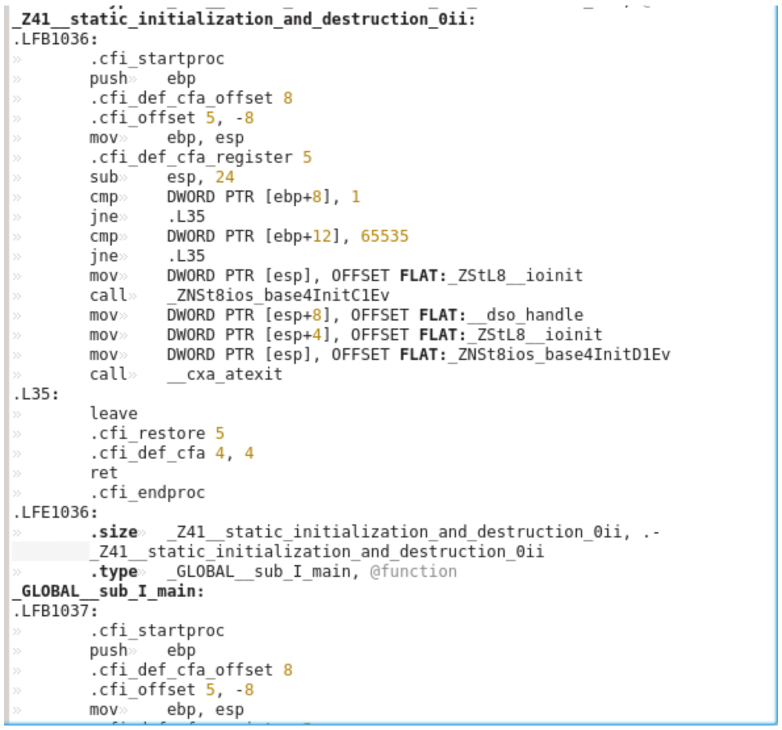
When a user-defined object is instantiated, the constructor is called and space is allocated for the fields defined within the object. When the destructor is called, the assembly destroys the element that the destructor is being called on. The space is reallocated and is now able to be used again. In addition, the destructor will reallocate the space originally allocated by every element it is called on. This means, if a vector were to be initiated and a destructor was called on the vector, it would perform this action on every element contained within the vector.

In terms of class hierarchy, the assembly code lines up fairly well with the c++ code. The object is instantiated and then the constructors for the base class and derived class are called respectively. When the function is over, the destructors are called on the two fields and performed first on the derived class and then on the base class. The general layout of the assembly with respect to class hierarchy is shown below as a snip from the main method.



```
»       mov»    DWORD PTR [esp], eax
.LEHB9:
»       call»   _ZN7ContactC1Ev
.LEHE9:
»       lea»    eax, [esp+20]
»       mov»    DWORD PTR [esp], eax
»       call»   _ZNSaIcEC1Ev
»       lea»    eax, [esp+20]
»       mov»    DWORD PTR [esp+8], eax
»       mov»    DWORD PTR [esp+4], OFFSET FLAT:.LC1
»       lea»    eax, [esp+16]
»       mov»    DWORD PTR [esp], eax
.LEHB10:
»       call»   _ZNSsC1EPKcRKSaIcE
.LEHE10:
»       lea»    eax, [esp+16]
»       mov»    DWORD PTR [esp+4], eax
»       lea»    eax, [esp+24]
»       mov»    DWORD PTR [esp], eax
.LEHB11:
»       call»   _ZN4Name7setNameESs
.LEHE11:
»       lea»    eax, [esp+16]
»       mov»    DWORD PTR [esp], eax
.LEHB12:
```

The assembly code under the initialization/destruction label holds code for constructors and destructors that are eventually called by the main method. When the static initialization and destruction code is called, the Contact class's destructor is called and then the Name class's destructor is called. This occurs in reverse order than when the constructors were called. The constructors were called in order of base class followed by derived class.



```
_Z41__static_initialization_and_destruction_0ii:
.LFB1036:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 24
        cmp     DWORD PTR [ebp+8], 1
        jne     .L35
        cmp     DWORD PTR [ebp+12], 65535
        jne     .L35
        mov     DWORD PTR [esp], OFFSET FLAT:_ZStL8__ioinit
        call    _ZNSt8ios_base4InitC1Ev
        mov     DWORD PTR [esp+8], OFFSET FLAT:__dso_handle
        mov     DWORD PTR [esp+4], OFFSET FLAT:_ZStL8__ioinit
        mov     DWORD PTR [esp], OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
        call    __cxa_atexit
.L35:
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE1036:
        .size   _Z41__static_initialization_and_destruction_0ii, .-
        _Z41__static_initialization_and_destruction_0ii
        .type   _GLOBAL__sub_I_main, @function
_GLOBAL__sub_I_main:
.LFB1037:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
```