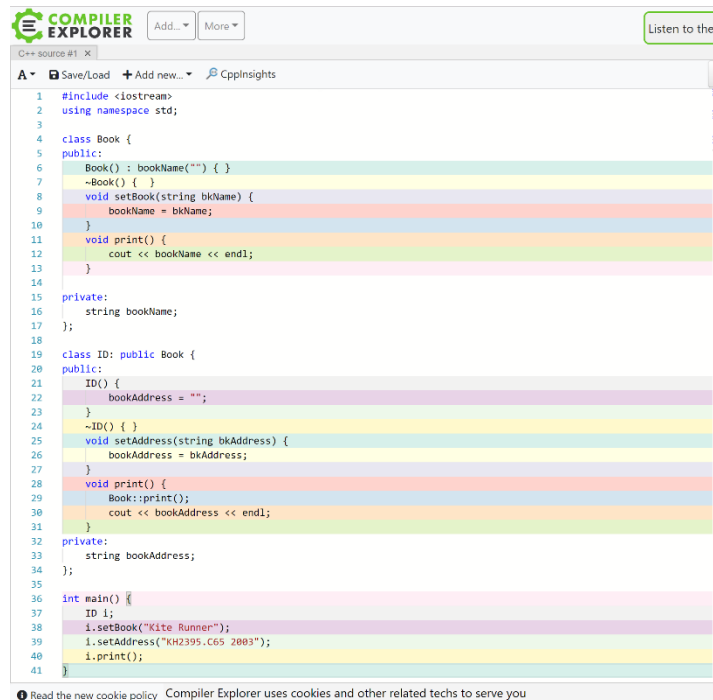


Eza Rasheed
er6qt
04-10-19
Inlab9.pdf

Inheritance:

Below, I wrote a C++ code based off of the example given in the lecture slides in order to examine the assembly code for multiple inheritance. ID is the “child class of Book and it inherits the data member bookName field from the Book parent class. In the code below, you can see that both the Book class and ID class have a constructor, destructor (~), a print method, and a private field (bookName, bookAddress). The data members inherited by ID from Book is the bookName field and it includes data member of its own, which is the bookAddress field. In the main method, I created the object “ID i”, initialized values into the private fields of bookName and bookAddress and then called the print method on the object the private fields were being called on.



```
1 #include <iostream>
2 using namespace std;
3
4 class Book {
5 public:
6     Book() : bookName("") { }
7     ~Book() { }
8     void setBook(string bkName) {
9         bookName = bkName;
10    }
11    void print() {
12        cout << bookName << endl;
13    }
14 private:
15     string bookName;
16 };
17
18 class ID: public Book {
19 public:
20     ID() {
21         bookAddress = "";
22     }
23     ~ID() { }
24     void setAddress(string bkAddress) {
25         bookAddress = bkAddress;
26     }
27     void print() {
28         Book::print();
29         cout << bookAddress << endl;
30     }
31 private:
32     string bookAddress;
33 };
34
35 int main() {
36     ID i;
37     i.setBook("Kite Runner");
38     i.setAddress("KH2395.C65 2003");
39     i.print();
40 }
```

Using the compiler explorer, I generated the assembly code to see where in memory the data members were being laid out and stored in the ID object. Looking at when the private fields in the main method are being stored, you can see that in order to store the bookName and bookAddress fields, space is made on the stack; This space stores the bookName field inherited from the Book class and the bookAddress, which is in the ID class itself. ID assigns space on the stack to call the bookName field (1). Also, the bookName field, which is inherited by the ID class from the Book class is allocated and stored (2), and the bookAddress field in the ID class itself has space allocated on the stack to store the book’s address (3).

1.

```

74 ID::ID() [base object constructor]:
75     push    rbp
76     mov     rbp, rsp
77     push    rbp
78     sub     rsp, 24
79     mov     QWORD PTR [rbp-24], rdi
80     mov     rax, QWORD PTR [rbp-24]
81     mov     rdi, rax
82     call    Book::Book() [base object constructor]
83     mov     rax, QWORD PTR [rbp-24]
84     add     rax, 32
85     mov     rdi, rax
86     call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
87     mov     rax, QWORD PTR [rbp-24]
88     add     rax, 32
89     mov     esi, OFFSET FLAT:.LC0
90     mov     rdi, rax
91     call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
92     jmp     .L11
93     mov     rbx, rax
94     mov     rax, QWORD PTR [rbp-24]
95     add     rax, 32
96     mov     rdi, rax
97     call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
98     mov     rax, QWORD PTR [rbp-24]
99     mov     rdi, rax
100    call    Book::~Book() [base object destructor]
101    mov     rax, rbx
102    mov     rdi, rax
103    call    _Unwind_Resume

```

2.

```

34 Book::~Book() [base object destructor]:
35     push    rbp
36     mov     rbp, rsp
37     sub     rsp, 16
38     mov     QWORD PTR [rbp-8], rdi
39     mov     rax, QWORD PTR [rbp-8]
40     mov     rdi, rax
41     call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
42     nop
43     leave
44     ret
45 Book::setBook(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
46     push    rbp
47     mov     rbp, rsp
48     sub     rsp, 16
49     mov     QWORD PTR [rbp-8], rdi
50     mov     QWORD PTR [rbp-16], rsi
51     mov     rax, QWORD PTR [rbp-8]
52     mov     rdx, QWORD PTR [rbp-16]
53     mov     rsi, rdx
54     mov     rdi, rax
55     call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
56     nop
57     leave
58     ret

```

3.

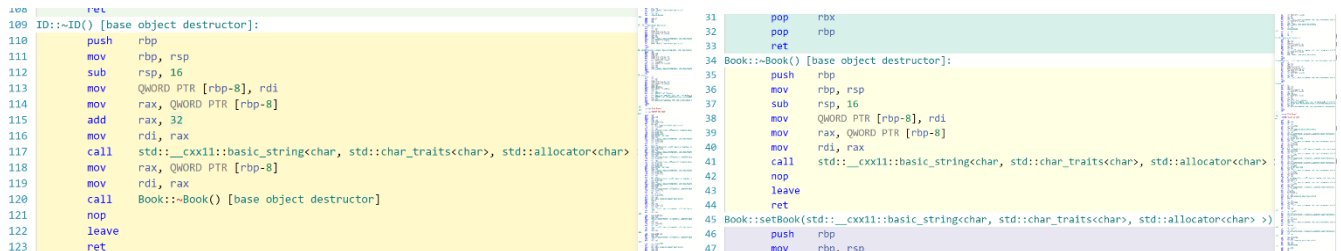
```

113     mov     QWORD PTR [rbp-8], rdi
114     mov     rax, QWORD PTR [rbp-8]
115     add     rax, 32
116     mov     rdi, rax
117     call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
118     mov     rax, QWORD PTR [rbp-8]
119     mov     rdi, rax
120     call    Book::~Book() [base object destructor]
121     nop
122     leave
123     ret
124 ID::setAddress(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
125     push    rbp
126     mov     rbp, rsp
127     sub     rsp, 16
128     mov     QWORD PTR [rbp-8], rdi
129     mov     QWORD PTR [rbp-16], rsi
130     mov     rax, QWORD PTR [rbp-8]
131     lea     rdx, [rax+32]
132     mov     rax, QWORD PTR [rbp-16]
133     mov     rsi, rax
134     mov     rdi, rdx
135     call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
136     nop
137     leave
138     ret
139 ID::print():

```

The construction and destruction of objects happens in this class hierarchy by first, calling the Book class's constructor to initialize the data members which will be inherited from this parent Book class. Then the ID class's constructor is called to initialize the new data members found within the instance of the object (ID i) itself, in this case bookAddress. When a user-defined object is instantiated, the default constructor is called to allocate space in memory for the two private fields called on the object. When it goes out of scope, the destructor is called for the class object to deallocate memory and "do other cleanup for a class object and its class members when the object is out of scope or explicitly deleted(destroyed)." This will reallocate space taken up

by every object, which then allows space to be freed up in memory and able to be used up again. This process in assembly code using a simple class hierarchy is nearly the same as written in the C++ code. First the object is instantiated, and then the constructor in the parent class (Book) is called, followed by the child class (ID). When the whole function is done running, the destructor of each class is called on the two private fields, bookName and bookAddress. In the screenshot at the end, you can see the assembly code of the class hierarchy for this.

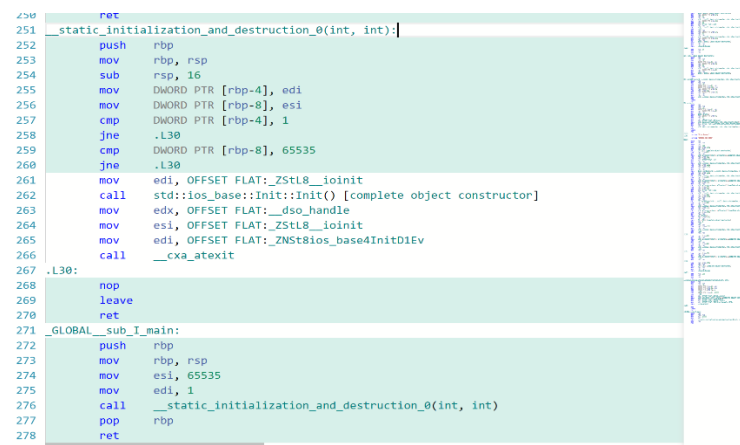


```

108 ID::~ID() [base object destructor]:
109     push    rbp
110     mov     rbp, rsp
111     sub     rsp, 16
112     mov     QWORD PTR [rbp-8], rdi
113     mov     rax, QWORD PTR [rbp-8]
114     add     rax, 32
115     mov     rdi, rax
116     call    std::_cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string
117     mov     rax, QWORD PTR [rbp-8]
118     mov     rdi, rax
119     call    Book::~Book() [base object destructor]
120     nop
121     leave
122     ret
123
31     pop     rbx
32     pop     rbp
33     ret
34 Book::~Book() [base object destructor]:
35     push    rbp
36     mov     rbp, rsp
37     sub     rsp, 16
38     mov     QWORD PTR [rbp-8], rdi
39     mov     rax, QWORD PTR [rbp-8]
40     mov     rdi, rax
41     call    std::_cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string
42     nop
43     leave
44     ret
45 Book::setBook(std::_cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >):
46     push    rbp
47     mov     rbp, rsp

```

The screenshots above show the assembly code for when the destructors are being implemented. At the end of the main, static initialization and destruction is called, which is when the destructors and constructors for both the classes are called. Destructors are called in a reverse order from how the constructors get called, for a reason I am not too sure of; first, the ID class's destructor is called, and then, the Book class's destructor is called. Looking at the assembly code below, you can see that the destructors and constructors are getting called at the end of the main function.



```

250     ret
251 __static_initialization_and_destruction_0(int, int):
252     push    rbp
253     mov     rbp, rsp
254     sub     rsp, 16
255     mov     DWORD PTR [rbp-4], edi
256     mov     DWORD PTR [rbp-8], esi
257     cmp     DWORD PTR [rbp-4], 1
258     jne     .L30
259     cmp     DWORD PTR [rbp-8], 65535
260     jne     .L30
261     mov     edi, OFFSET FLAT:_ZStL8__ioinit
262     call    std::ios_base::Init::Init() [complete object constructor]
263     mov     edx, OFFSET FLAT:_dso_handle
264     mov     esi, OFFSET FLAT:_ZStL8__ioinit
265     mov     edi, OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
266     call    __cxa_atexit
267 .L30:
268     nop
269     leave
270     ret
271 GLOBAL __sub_I_main:
272     push    rbp
273     mov     rbp, rsp
274     mov     esi, 65535
275     mov     edi, 1
276     call    __static_initialization_and_destruction_0(int, int)
277     pop     rbp
278     ret

```

Sources:

<http://aaronbloomfield.github.io/pdr/slides/code/09-advanced-cpp/name-contact.cpp>

<https://www.geeksforgeeks.org/multiple-inheritance-in-c/>

https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.cbclx01/cplr380.htm