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.

```cpp
#include <iostream>
using namespace std;

class Book {
public:
    Book() : bookName("") { }
    ~Book() {  }
    void setBook(string bkName) {
        bookName = bkName;
    }
    void print() {
        cout << bookName << endl;
    }

private:
    string bookName;
};

class ID: public Book {
public:
    ID() {
        bookAddress = "";
    }
    ~ID() { }
    void setAddress(string bkAddress) {
        bookAddress = bkAddress;
    }
    void print() {
        Book::print();
        cout << bookAddress << endl;
    }
private:
    string bookAddress;
};

int main() {
    ID i;
    i.setBook("Kite Runner");
    i.setAddress("KH2395.C65 2003");
    i.print();
}
```

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.

```
ID::ID() [base object constructor]:
        push    rbp
        mov     rbp, rsp
        push    rbx
        sub     rsp, 24
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Book::Book() [base object constructor]
        mov     rax, QWORD PTR [rbp-24]
        add     rax, 32
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        mov     rax, QWORD PTR [rbp-24]
        add     rax, 32
        mov     esi, OFFSET FLAT:.LC0
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        jmp     .L11
        mov     rbx, rax
        mov     rax, QWORD PTR [rbp-24]
        add     rax, 32
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Book::~Book() [base object destructor]
        mov     rax, rbx
        mov     rdi, rax
        call    _Unwind_Resume
```

2.

```
Book::~Book() [base object destructor]:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        nop
        leave
        ret
Book::setBook(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     QWORD PTR [rbp-16], rsi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdx, QWORD PTR [rbp-16]
        mov     rsi, rdx
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        nop
        leave
        ret
```

3.

```
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        add     rax, 32
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Book::~Book() [base object destructor]
        nop
        leave
        ret
ID::setAddress(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     QWORD PTR [rbp-16], rsi
        mov     rax, QWORD PTR [rbp-8]
        lea     rdx, [rax+32]
        mov     rax, QWORD PTR [rbp-16]
        mov     rsi, rax
        mov     rdi, rdx
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        nop
        leave
        ret
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.

```
        ret
ID::~ID() [base object destructor]:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        add     rax, 32
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Book::~Book() [base object destructor]
        nop
        leave
        ret
```

```
        pop     rbx
        pop     rbp
        ret
Book::~Book() [base object destructor]:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
        nop
        leave
        ret
Book::setBook(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
        push    rbp
        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.

```
        ret
__static_initialization_and_destruction_0(int, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        mov     DWORD PTR [rbp-8], esi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L30
        cmp     DWORD PTR [rbp-8], 65535
        jne     .L30
        mov     edi, OFFSET FLAT:_ZStL8__ioinit
        call    std::ios_base::Init::Init() [complete object constructor]
        mov     edx, OFFSET FLAT:__dso_handle
        mov     esi, OFFSET FLAT:_ZStL8__ioinit
        mov     edi, OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
        call    __cxa_atexit
.L30:
        nop
        leave
        ret
_GLOBAL__sub_I_main:
        push    rbp
        mov     rbp, rsp
        mov     esi, 65535
        mov     edi, 1
        call    __static_initialization_and_destruction_0(int, int)
        pop     rbp
        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