Samreen Azam (sa3tnc)
11/21/19
postlab9.pdf

Dynamic Dispatch:

Dynamic dispatch refers to the process of determining which version of a polymorphic function should be called during runtime[1]. To see how this is implemented, I created two simple classes: Food and Cookie, which inherits Food. Their implementations in C++ are shown below.

```
class Food {
 public:
  virtual void buyFood() {
    cout << "You just bought food.\n";
  }
  virtual void checkPrice() {
    cout << "Food is $1.";
  }
};

class Cookie: public Food { //inherits Food
 public:
  void buyFood() {
    cout << "You just bought a cookie.\n";
  }
  void checkPrice() {
    cout << "Cookies are $1.";
  }
};
```

```
int main() {
  Food *f;
  Food f2;
  Cookie c;
  f = &c; //use reference
  //method calls:
  f->buyFood();
  f->checkPrice();
  c.buyFood();
  c.checkPrice();
  f2.buyFood();
  f2.checkPrice();
  return 0;
}
```

Output:

```
You just bought a cookie.
Cookies are $1.
You just bought a cookie.
Cookies are $1.
You just bought food.
Food is $1.
```

This occurs because f is set to point to c, and thus can use Cookie's buyFood() and checkPrice() methods. For c, the overridden methods in its class are called. Finally, f2 only has access to the operations in the Food class, so it can only call the original virtual methods.

Next, I generated assembly code via godbolt.com. The following image depicts the function calls in main:

```
76      lea     rax, [rbp-24]
77      mov     rdi, rax
78      call    Cookie::buyFood()
79      lea     rax, [rbp-24]
80      mov     rdi, rax
81      call    Cookie::checkPrice()
82      lea     rax, [rbp-16]
83      mov     rdi, rax
84      call    Food::buyFood()
85      lea     rax, [rbp-16]
86      mov     rdi, rax
87      call    Food::checkPrice()
88      mov     eax, 0
89      leave
90      ret
```

The methods in the Food class are only called at the end, at Lines 84 and 87. On the other hand, the Cookie class methods are called first. This conveys how only f2 uses the methods in the Food class whereas c and f call Cookie's methods. Since f is a reference type that holds the address of c, this is expected.

Inheritance:

In order to explain how the layout of data and the implementations of the constructor and destructor of a class are presented in assembly, I modified the Food and Cookie classes. As seen below, the updated classes include constructors, destructors, and private fields that can be printed.

```
class Food {
 public:
  Food() {
    ingredients = "yummy stuff";
    cout << "Food object created!\n";
  }
  ~Food() {
     cout << "Food object destroyed!\n";
  }
  void printIng() {
    cout << ingredients << endl;
  }

 private:
   string ingredients;
};
```

```
class Cookie: public Food { //inherits Food
 public:
  Cookie() { //constructor
    isGlutenFree = "nope";
    cout << "Cookie object created!\n";
  }
  ~Cookie() { //destructor
    cout << "Cookie object destroyed!\n";
  }
  void printData() {
    printIng(); //print Food data as well
    cout << isGlutenFree << endl;
  }

 private:
  string isGlutenFree;
};
```

The main method creates objects and calls their functions. The output is shown on the right.

```
int main() {
  Food f;
  Cookie c;
  cout << "Food data: \n";
  f.printIng();
  cout << "Cookie data: \n";
  c.printData();
```

```
Food object created!
Food object created!
Cookie object created!
Food data:
yummy stuff
Cookie data:
yummy stuff
nope
Cookie object destroyed!
Food object destroyed!
Food object destroyed!
```

The output of the main method shows that the constructor and destructor of Food is actually called twice; this relates to how all Cookie objects inherit from the Food class and can also be considered Food objects. In other words, a subclass has an "is-a" relationship with its parent[2], so we can say that "a Cookie is a Food". Therefore, when a Cookie object is declared, its parent's constructor is called in addition to its own. Likewise, both destructors are called when deleting a Cookie object. It is also notable that when creating an object of a child class, the constructor of the parent class is called first. This contrasts from the destruction of the object since the parent's destructor was called after the child's.

Observing the assembly code for these classes provides further support for this. It also allows for greater insight about how data is laid in memory as well as the procedure for constructing and destroying objects within the class hierarchy.

Firstly, when creating data members to be stored in memory, the class hierarchy is reflected in the order in which variables are stored. Upon instantiating an object, These fields are also stored sequentially in memory[3], so the variable isGlutenFree would always be initialized and saved after ingredients. Some annotated screenshots of the main method in assembly are included below.

```
150  main:
151          push    rbp
152          mov     rbp, rsp          Make space on stack
153          push    rbx
154          sub     rsp, 104
155          lea     rax, [rbp-48]
156          mov     rdi, rax          Call Food constructor to create f
157          call    Food::Food() [complete object constructor]
158          lea     rax, [rbp-112]
159          mov     rdi, rax          Call Cookie constructor to make c
160          call    Cookie::Cookie() [complete object constructor]
161          mov     esi, OFFSET FLAT:.LC6
162          mov     edi, OFFSET FLAT:_ZSt4cout
163          call    std::basic_ostream<char, std::char_traits<char>
164          lea     rax, [rbp-48]
165          mov     rdi, rax          Call printIng() for f
166          call    Food::printIng()
167          mov     esi, OFFSET FLAT:.LC7
168          mov     edi, OFFSET FLAT:_ZSt4cout
169          call    std::basic_ostream<char, std::char_traits<char>
170          lea     rax, [rbp-112]
171          mov     rdi, rax          Call printData() for c
172          call    Cookie::printData()
173          mov     ebx, 0
```

After the objects are created and all operations are performed in main, the destructor is called just before the main method ends to destruct f and c:

```
174          lea     rax, [rbp-112]
175          mov     rdi, rax
176          call    Cookie::~Cookie() [complete object destructor]
177          lea     rax, [rbp-48]
178          mov     rdi, rax
179          call    Food::~Food() [complete object destructor]
180          mov     eax, ebx
181          jmp     .L19
182          mov     rbx, rax
183          lea     rax, [rbp-112]
184          mov     rdi, rax
185          call    Cookie::~Cookie() [complete object destructor]
186          jmp     .L16
187          mov     rbx, rax
188  .L16:
189          lea     rax, [rbp-48]
190          mov     rdi, rax
191          call    Food::~Food() [complete object destructor]
192          mov     rax, rbx
193          mov     rdi, rax
194          call    _Unwind_Resume
```

Moreover, I noticed a few key operations in the Cookie class constructor that were not present in the Food class. These differences exemplify how Cookie extends the Food class. For instance, when a Food object is created, only the constructor of the Food class is called by main. The process is slightly more complex when making a Cookie object. When calling the Cookie constructor, the Food constructor's base constructor is called *before* Cookie's complete constructor is. Since Food's constructor is called first, the "Food object was created!" message appears before the "Cookie object was created!" message. This is congruent to the output printed to the console when running the C++ program. Then, when the program moves out of the scope of the object, the destructor is called and the Cookie object's life cycle ends. Cookie's base constructor and destructor are shown below.

```
71          mov     rbp, rsp
72          push    rbx
73          sub     rsp, 24
74          mov     QWORD PTR [rbp-24], rdi   Make space on stack for object
75          mov     rax, QWORD PTR [rbp-24]
76          mov     rdi, rax
77          call    Food::Food() [base object constructor]   Call base constructor of Food class
78          mov     rax, QWORD PTR [rbp-24]
79          add     rax, 32
80          mov     rdi, rax                                 Call complete object constructor of Cookie class
81          call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::basic_string() [complete object constructor]
82          mov     rax, QWORD PTR [rbp-24]
83          add     rax, 32                                  Initialize isGlutenFree
84          mov     esi, OFFSET FLAT:.LC3
85          mov     rdi, rax
86          call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::operator=(char const*)
87          mov     esi, OFFSET FLAT:.LC4
88          mov     edi, OFFSET FLAT:_ZSt4cout               Print message
89          call    std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, std::cha
90          jmp     .L10
91          mov     rbx, rax
92          mov     rax, QWORD PTR [rbp-24]
93          add     rax, 32
94          mov     rdi, rax                                 Call complete destructor for Cookie
95          call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string() [complete object destructor]
96          mov     rax, QWORD PTR [rbp-24]
97          mov     rdi, rax
98          call    Food::~Food() [base object destructor]   Call base destructor of Food
```

```
107 .LC5:
108         .string "Cookie object destroyed!\n"
109 Cookie::~Cookie() [base object destructor]:
110         push    rbp
111         mov     rbp, rsp
112         sub     rsp, 16
113         mov     QWORD PTR [rbp-8], rdi
114         mov     esi, OFFSET FLAT:.LC5
115         mov     edi, OFFSET FLAT:_ZSt4cout               Print message
116         call    std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, std::char
117         mov     rax, QWORD PTR [rbp-8]
118         add     rax, 32                                  Deallocate memory, call complete
119         mov     rdi, rax                                 object destructor
120         call    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string() [complete object destructor]
121         mov     rax, QWORD PTR [rbp-8]
122         mov     rdi, rax
123         call    Food::~Food() [base object destructor]   Call base destructor of Food class
124         nop
125         leave
126         ret
```

Sources:

1. https://condor.depaul.edu/ichu/csc447/notes/wk10/Dynamic2.htm

2. https://runestone.academy/runestone/books/published/cppds/Introduction/ObjectOrientedProgrammingDerivedClasses.html

3. https://stackoverflow.com/questions/12378271/what-does-an-object-look-like-in-memory