333 Section 6 - C++ Casting and Inheritance

Welcome back to section! We're happy to have you here ʕ•ᴥ•ʔ

Casting in C++

While in C++, we want to use casts that are more explicit in their behavior. This gives us a better understanding of what happens when we read our code, because C-style casts can do many (sometimes unwanted) things. There are four types of casts we will use in C++:

- `static_cast<type_to>(expression);`
  Casting between related types
- `dynamic_cast<type_to>(expression);`
  Casting pointers of similar types (only used with inheritance)
- `const_cast<type_to>(expression);`
  Adding or removing **const**-ness of a type
- `reinterpret_cast<type_to>(expression);`
  Casting between incompatible types of the **same size** (doesn't do float conversion)

Inheritance in C++

A **Derived** class inherits from a **base** class (*Similar to*: A subclass inherits from a superclass)

- A derived class inherits all **non-private** member variables and functions (**except** for `ctor, cctor, dtor, op=`)
- Aside: We will be only using **public** inheritance in CSE 333

Abstract Class Examples: Fruit (Abstract) & Banana (Derived)

```
#include <string>
using std::string;

class Fruit {
 public:
  Fruit() = default;
  virtual ~Fruit() {}
  // A fun fact
  virtual string FunFact() = 0;

};
```

```
#include <string>
using std::string;

class Banana : public Fruit {
 public:
  Banana() = default;
  virtual ~Banana() = default;
  string FunFact() override {
    return "It's a berry";
  }
};
```

Style Considerations

- Use `virtual` **only once** when first defined in the base class
- All derived classes of a base class should use `override` to check at compile time that a function uses dynamic dispatch
- Call `dtors` of a base class as `virtual` – guarantees all derived classes will use dynamic dispatch for their destructors

Exercise 1

For each of the following, write down if we are using static dispatch, dynamic dispatch, or triggering a compile error. If there are any style mistakes or bugs, how would you fix them?

```
class Base {
  void Foo();            _____________________________
  void Bar();            _____________________________
  virtual void Baz();    _____________________________
};

class Derived : public Base {
  virtual void Foo();    _____________________________
  void Bar() override;   _____________________________
  void Baz();            _____________________________
};
```

Exercise 2

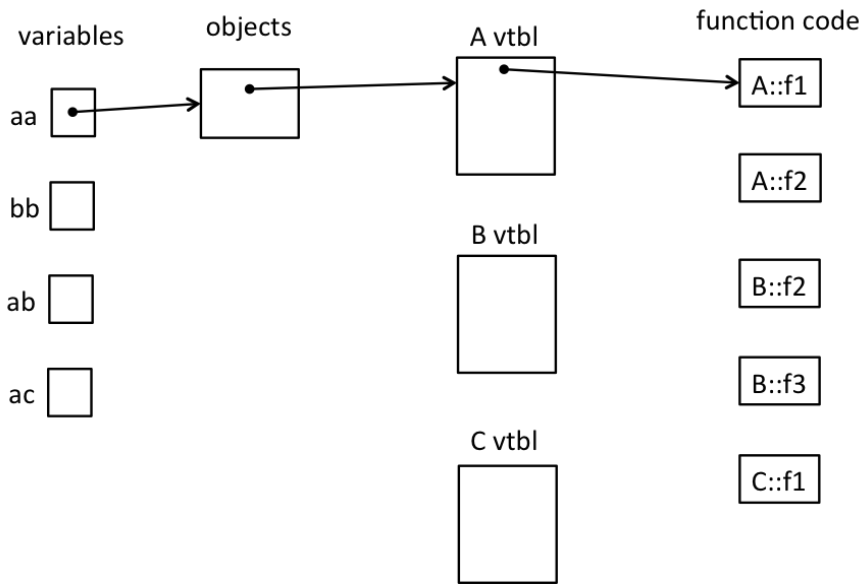
Consider the program on the following page, which does compile and execute with no errors, except that it leaks memory (which doesn't matter for this question).

(a) Complete the diagram on the next page by adding the remaining objects and all of the additional pointers needed to link variables, objects, virtual function tables, and function bodies. Be sure that the order of pointers in the virtual function tables is clear (i.e., which one is first, then next, etc.). One of the objects and a couple of the pointers are already included to help you get started.

(b) Write the output produced when this program is executed. If the output doesn't fit in one column in the space provided, write multiple vertical columns showing the output going from top to bottom, then successive columns to the right

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

class A {
 public:
  virtual void f1() { f2(); cout << "A::f1" << endl; }
  void f2() { cout << "A::f2" << endl; }
};
class B : public A {
 public:
  virtual void f3() { f1(); cout << "B::f3" << endl; }
  virtual void f2() { cout << "B::f2" << endl; }
};
class C : public B {
 public:
  void f1() { f2(); cout << "C::f1" << endl; }
};
```



```
int main() {
  A* aa = new A();
  B* bb = new B();
  A* ab = bb;
  A* ac = new C();
  aa->f1();
  cout << "---" << endl;
  bb->f1();
  cout << "---" << endl;
  bb->f2();
  cout << "---" << endl;
  ab->f2();
  cout << "---" << endl;
  bb->f3();
  cout << "---" << endl;
  ac->f1();
  return EXIT_SUCCESS;
}
```

Output: