# 333 Section SOLUTIONS 7 - C++ Casting and Inheritance

## Exercise 1

### "Smart" LinkedList

Consider the `IntNode` struct below. Convert the `IntNode` struct to be "smart" by using smart pointers! Using `unique_ptr` for value as only one Node should look at a certain value (prevent double delete), and `shared_ptr` for next value, as the client code uses `shared_ptr` for the iterator.

```cpp
#include <memory>
using std::shared_ptr;
using std::unique_ptr;
using std::weak_ptr;

template <typename T>
struct IntNode {
  IntNode(int* val, IntNode* node): value(unique_ptr<int>(val)),
                                    next(shared_ptr<IntNode>(node)){}
  ~IntNode() { delete value; }
  unique_ptr<int> value;
  shared_ptr<IntNode> next;
};
```
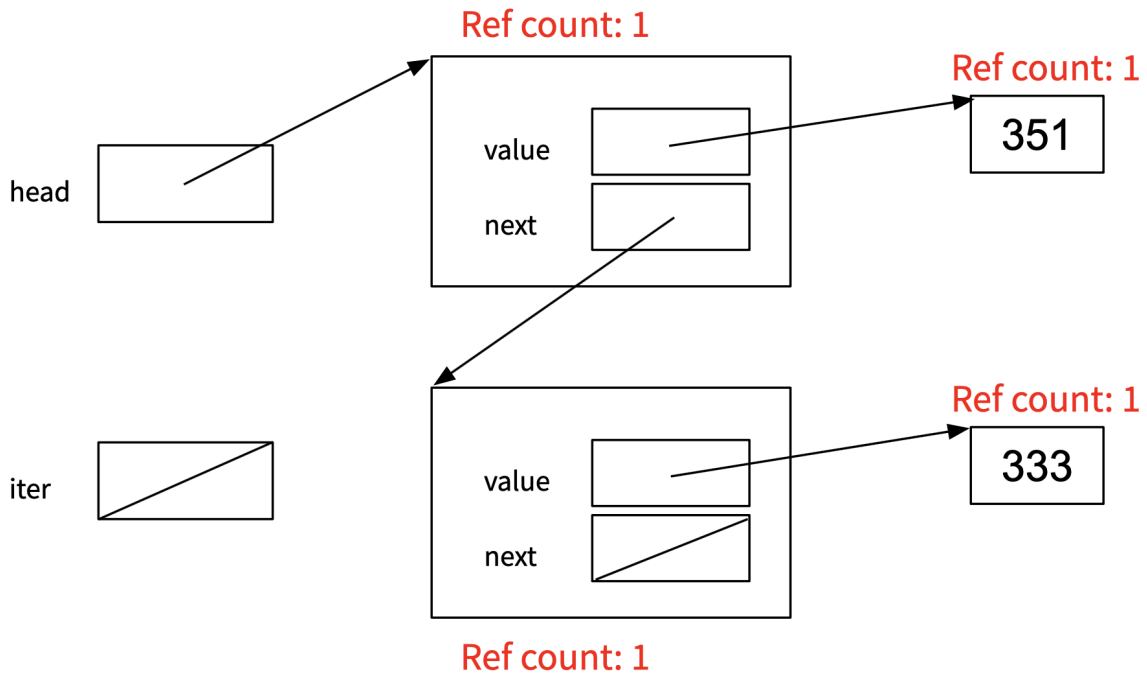
After the conversion, draw a memory diagram with the reference count for blocks of memory.

```cpp
#include <iostream>

using std::cout;
using std::endl;

int main() {
  shared_ptr<IntNode> head =
      shared_ptr<IntNode>(new IntNode(new int(351), nullptr));
  head->next = shared_ptr<IntNode>(new IntNode(new int(333),
                                               nullptr));
  shared_ptr<IntNode> iter = head;
  while (iter != nullptr) {
    cout << *(iter->value) << endl;
    iter = iter->next;
  }
}
```

Ref count: 1

Ref count: 1

351

head

value

next

iter

Ref count: 1

333

value

next

Ref count: 1

This memory diagram is just before we exit the while loop.

# Exercise 2

For each of the following, write down if we are using static dispatch, dynamic dispatch, or triggering a compile error.

```
class Base {
  void Foo();          static dispatch
  void Bar();          static dispatch
  virtual void Baz();  dynamic dispatch
};

class Derived : public Base {
  virtual void Foo();   dynamic dispatch (For more derived class)
  void Bar() override;  compiler error!
  void Baz();           dynamic dispatch
};
```

If there are any style mistakes or bugs, how would you fix them?

```
1.
class Derived : public Base {
  ...
  void Bar() override; → void Bar();
  ...
};

Using static dispatch, fixes compiler error.

2.
class Derived : public Base {
  ...
  void Baz(); → void Baz() override;
};

Will still be dynamic dispatch, but override with Derived's
implementation.
```

# Exercise 3

Exercise 3A – Create an Animal Abstract class. It should have a protected member legs variable and a public num_legs member function. Try to use good style!

```cpp
class Animal {
 public:
  Animal() = default;
  virtual ~Animal() {}
  virtual int num_legs() const = 0;
 protected:
  int    legs;
};
```

Exercise 3B
Now that you have made an abstract Animal class, try to make an implementation with a derived class of Animal.

This is an open-ended question, so you are free to be imaginative with your implementation of the abstract Animal Class!

```cpp
public Dog {
 public:
  Dog(int legs, string breed) : legs(legs), breed(breed) {}
  virtual ~Dog() {}
  int num_legs() const override {
    return legs;
  }
  int getBreed() const override {
    return breed;
  }
 protected:
  string breed;
};
```

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

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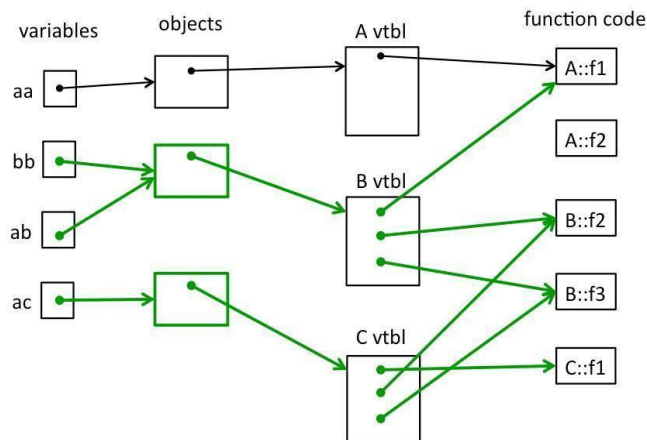
```cpp
int main() {
  A* aa = new A();
  B* bb = new B();
  A* ab = bb;
  A* ac = new C();
  aa->f1();
  cout << "---" << endl;
  bb->f1();
  cout << "---" << endl;
  bb->f2();
  cout << "---" << endl;
  ab->f2();
  cout << "---" << endl;
  bb->f3();
  cout << "---" << endl;
  ac->f1();
  return EXIT_SUCCESS;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

```
Output:
A::f2
A::f1
---
A::f2
A::f1
---
B::f2
---
A::f2
---
A::f2
A::f1
B::f3
---
B::f2
C::f1
```