CSE 333 Section 7 SAMPLE SOLUTIONS - C++ Casting and Inheritance

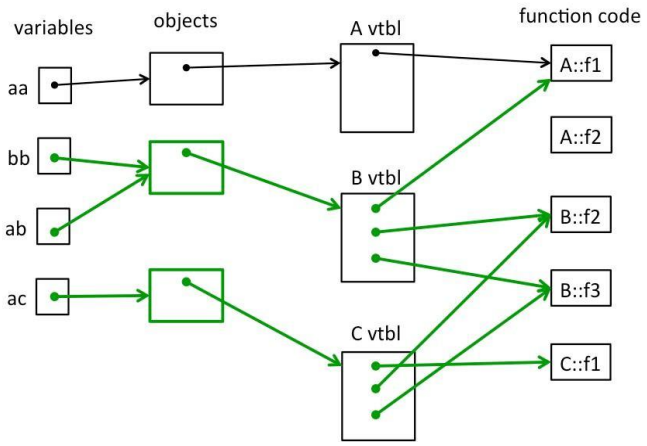
Exercise 1

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

Bonus

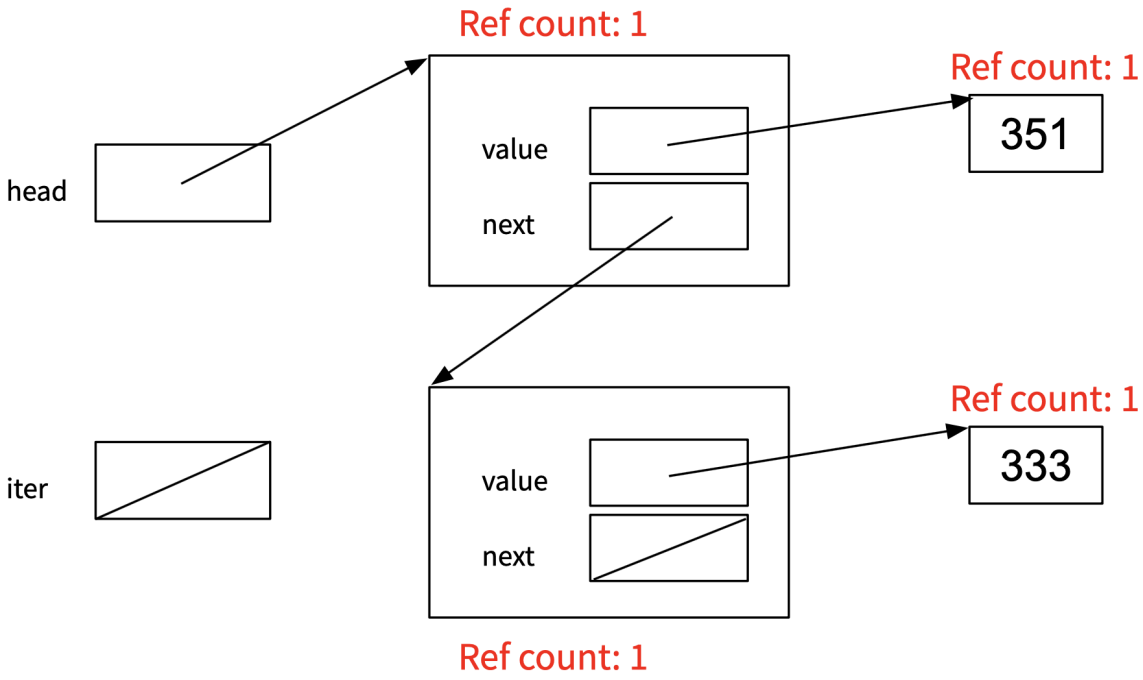
"Smart" LinkedList

Consider the `IntNode` struct below. Convert the `IntNode` struct to be "smart" by using `shared_ptr`.

```
#include <memory>
using std::shared_ptr;

template <typename T>
struct IntNode {
  IntNode(int* val, IntNode* node): value(shared_ptr<int>(val)),
                                    next(shared_ptr<IntNode>(node)){}
  ~IntNode() {~~delete value;~~ }
  shared_ptr<int> value;
  shared_ptr<IntNode> next;
};
```

After the conversion, draw a memory diagram with the reference count for blocks of memory.

```
#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;
  }
}
```



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

Bonus:

Virtual holidays! Consider the following C++ program, which does compile and execute successfully.

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

class One
{
  public:
        void m1() { cout << "H"; }
virtual void m2() { cout << "l"; }
virtual void m3() { cout << "p"; }
};

class Two: public One {
public:
  virtual void m1() { cout << "a"; }
          void m2() { cout << "d"; }
  virtual void m3() { cout << "y"; }
   virtual void m4() { cout << "p";}
};

class Three: public Two
{
   public:
     void m1() { cout << "o"; }
     void m2() { cout << "i"; }
     void m3() { cout << "s"; }
     void m4() { cout << "!"; }
   };
```

```
int main() {
    Two t;
    Three th;
    One *op = &t;
    Two *tp = &th;
    Three *thp = &th;

    op->m1();
    tp->m1();
    op->m3();
    op->m3();
    tp->m3();

    op->m1();
    thp->m1();
    op->m2();
    thp->m2();
    tp->m2();
    tp->m1();
    tp->m3();
    thp->m3();
    tp->m4(); cout <<
    endl;
}
```

(a) (8 points) On the next page, complete the diagram showing all of the variables, objects, virtual method tables (vtables) and functions in this program. Parts of the diagram are supplied for you. **Do not remove** this page from the exam.

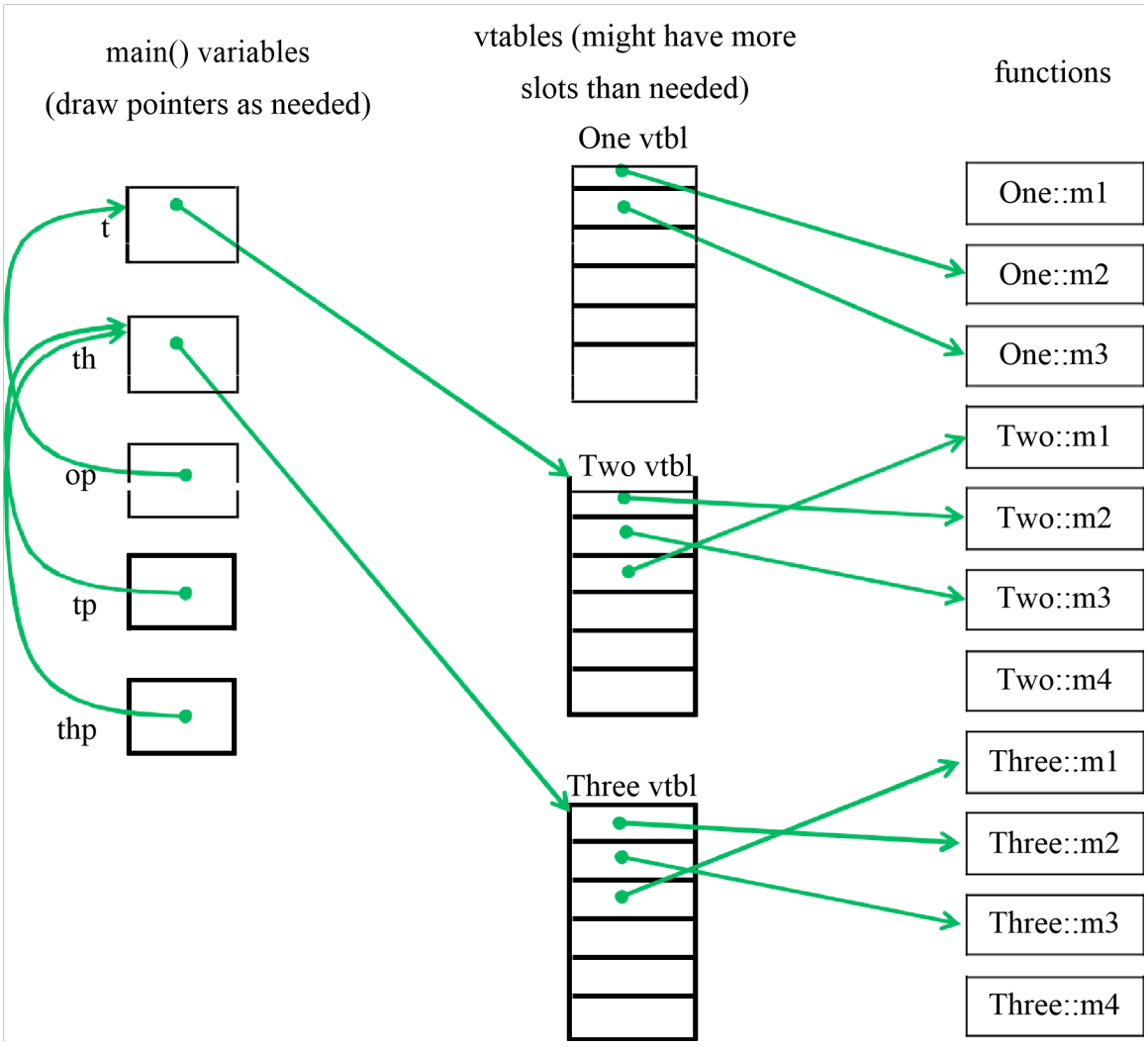
(b) (6 points) What does this program print when it executes?

HoyysHodiiossp

(c) (6 points) Modify the above program by removing and/or adding the virtual keyword in appropriate place(s) so that the modified program prints HappyHolidays! (including the ! at the end). Draw a line through the virtual keyword where it should be deleted and write in virtual where it needs to be added. Do not make any other changes to the program. Any correct solution will receive full credit.

(Changes shown above in bold green)

(cont.) Draw your answer to part (a) here. Complete the vtable diagram below. Draw arrows to show pointers from variables to objects, from objects to vtables, and from vtable slots to functions. Note that there may be more slots provided in the blank vtables than you actually need. Leave any unused slots blank



Notes: The pointers in One's vtable can be in either order – the first slot could point

to m3 and the second one to m2. BUT: once that choice is made, the first two vtable slots in Two's and Three's vtables must point to the appropriate versions of the same two functions in the same order as One's vtable. Two's vtable pointer to m1 must follow those slots, and Three's vtable slots must have exactly the same order.

Functions that are not virtual do not have pointers to them in vtables.

Objects contain only a single pointer to the correct vtable, not multiple pointers to multiple functions or other extra data.