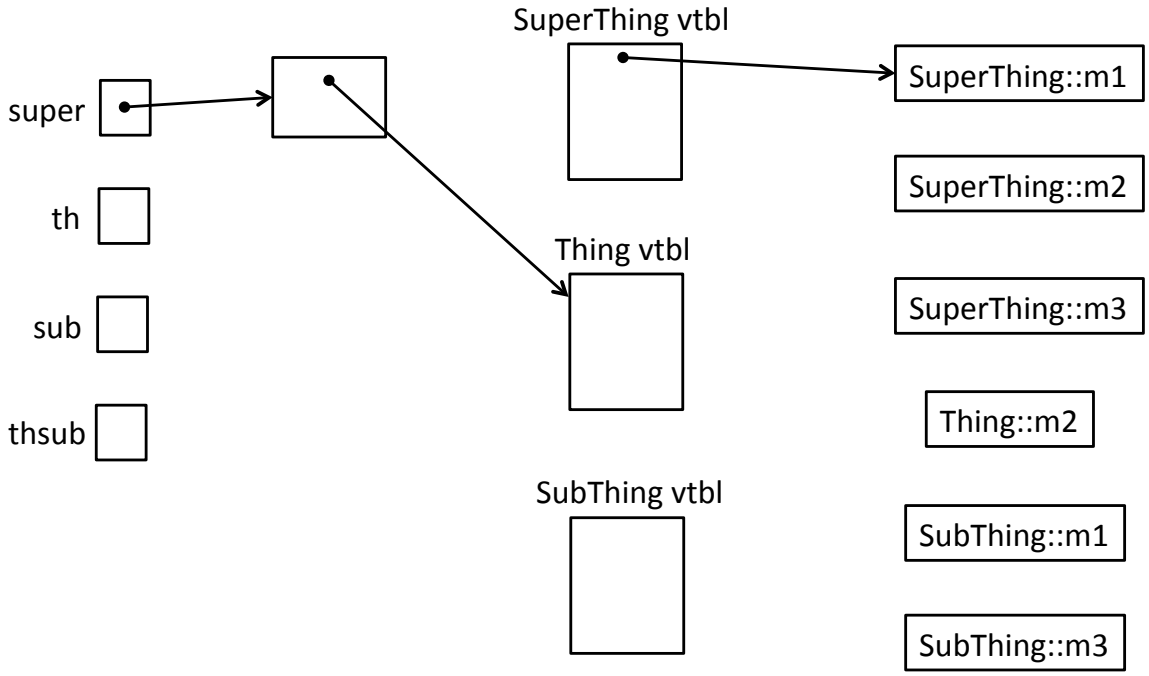
Question 1. The always entertaining virtual function question. The following program compiles, runs, and produces output with no error messages or other problems. Answer questions about it on the next page.

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

class SuperThing {
public:
  virtual void m1() { m2(); cout << "super::m1" << endl; }
          void m2() {       cout << "super::m2" << endl; }
          void m3() {       cout << "super::m3" << endl; }
};
class Thing: public SuperThing {
public:
  virtual void m2() { m1(); cout << "thing::m2" << endl; }
};
class SubThing: public Thing {
public:
  virtual void m1() {       cout << "sub::m1" << endl; }
          void m3() { m2(); cout << "sub::m3" << endl; }
};

int main() {
  SuperThing *super = new Thing();
  Thing      *th    = (Thing*)super;
  SubThing   *sub   = new SubThing();
  Thing      *thsub = sub;
  ///// HERE /////
  cout << "---" << endl;
  th->m1();
  th->m3();
  cout << "---" << endl;
  sub->m1();
  sub->m3();
  cout << "---" << endl;
  thsub->m1();
  thsub->m3();

  return 0;
}
```

(Question continued on next page – you may remove this page if you wish.)

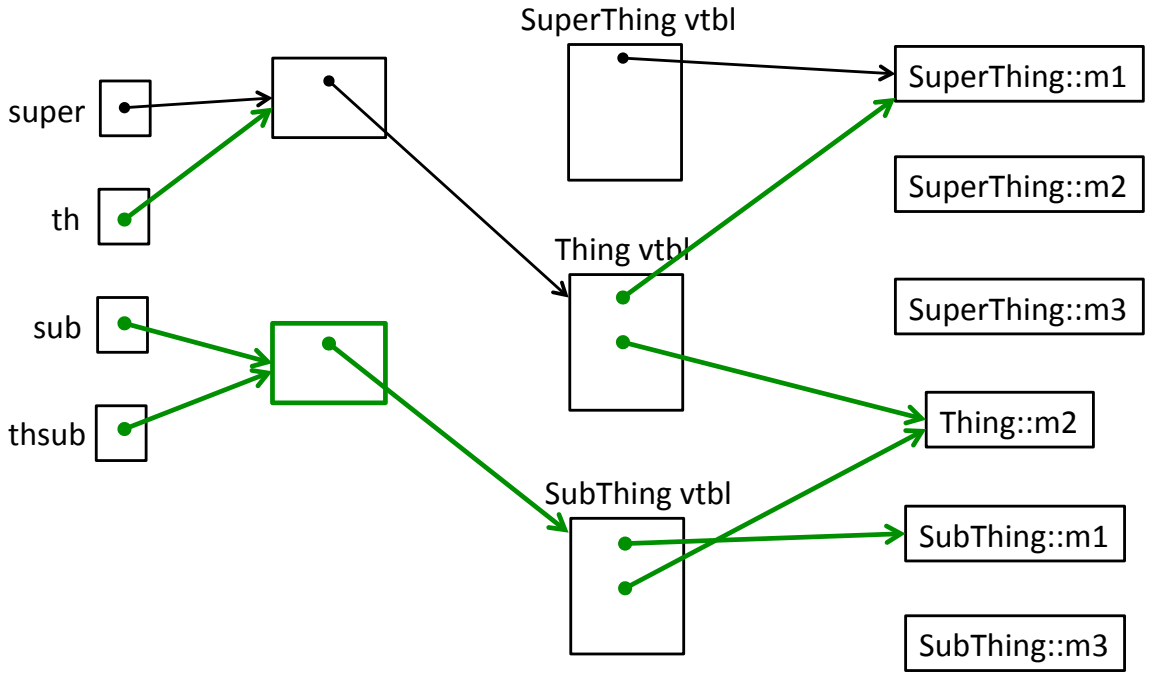
(a) Complete the following diagram to show the runtime state of the program when execution reaches the comment `///// HERE /////` in function `main`. The diagram should include the variables in `main` (already supplied), the objects they point to, pointers from objects to their vtables, and pointers from vtables to the correct functions. To save time, boxes for the variables in main, the vtables, the functions, and the first object created by the program, have been provided for you. A couple of the arrows representing some of the pointers are also included to get you started. You need to supply all additional objects and pointers needed (if any). Be sure that the order of pointers in the various vtables is clear.



(b) What does this program print when it is executed?

SAMPLE SOLUTION

Question 1 (a) Complete the following diagram to show the runtime state of the program when execution reaches the comment `///// HERE /////` in function `main`. The diagram should include the variables in `main` (already supplied), the objects they point to, pointers from objects to their vtables, and pointers from vtables to the correct functions. To save time, boxes for the variables in main, the vtables, the functions, and the first object created by the program, have been provided for you. A couple of the arrows representing some of the pointers are also included to get you started. You need to supply all additional objects and pointers needed (if any). Be sure that the order of pointers in the various vtables is clear.



(b) (12 points) What does this program print when it is executed?

```
---
super::m2
super::m1
super::m3
---
sub::m1
sub::m1
thing::m2
sub::m3
---
sub::m1
super::m3
```

NOTES

Some things discussed during section –

When **`th->m1()`** is called, the vtable entry shows that **`SuperThing::m1()`** is called. The call to **`m2()`** in **`SuperThing::m1()`** calls **`SuperThing::m2()`** because of static dispatch. There is no vtable entry for **`SuperThing::m2(),`** so **`SuperThing::m2()`** is the most derived at this point. If **`SuperThing::m2()`** were declared **`virtual`**, then **`Thing::m2()`** would have been invoked. You can test this yourself.

The next two calls are simple enough to follow, so we'll skip them. When **`sub->m3()`** is called, in **`SubThing::m3()`**, the call **`m2()`** invokes **`Thing::m2()`**, which then calls **`SubThing::m1()`** using dynamic dispatch because **`SubThing::m1()`** is the most derived at this point.

Similarly, **`thsub->m1()`** calls the most derived **`SubThing::m1()`** [Dynamic Dispatch], whereas **`thsub->m3()`** calls **`SuperThing::m3()`** [Static Dispatch].

Question 2. `virtual` reality. The following program compiles, runs, and produces output with no error messages or other problems. Notice that there are no `virtual` functions in the code. Answer questions about it at the bottom of the page.

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

class A {
public:
  void m1() { cout << "!"; }
  virtual void m2() { cout << "a"; }
  void m3() { cout << "o"; }
  void m4() { cout << "s"; }
};

class B: public A {
public:
  virtual void m1() { cout << "H"; }
  void m2() { cout << "3"; }
  void m3() { cout << "c"; }
  virtual void m4() { cout << "k"; }
};

class C: public B {
public:
  void m1() { cout << "r"; }
  void m3() { cout << "l"; }
  void m4() { cout << " "; }
};
```

```
int main() {
  B b;
  C c;
  A *aPtr = &b;
  B *bPtr = &c;

  aPtr->m2();
  bPtr->m2();
  bPtr->m2();
  bPtr->m4();
  bPtr->m1();
  aPtr->m3();
  bPtr->m3();

  aPtr = &c;
  bPtr = &b;
  bPtr->m4();
  aPtr->m4();
  aPtr->m1();
  cout << endl;
  return 0;
}
```

(a) What does this program print when it executes? Answer the question for the program exactly as written with no `virtual` functions.

a33kHocks!

(b) Modify the program above by adding the `virtual` keyword in appropriate places so that the modified program will print `333 rocks!` (including the `!` and the space between `333` and `rocks`, but no spaces before or after). You may not make any other changes to the code other than showing where to add `virtual`. There may be more than one possible solution. Any correct solution is acceptable.

See code changes above. Those are the minimum ones needed.