333 Section 6 - C++ Inheritance, Templates, and STL

Welcome back to section! We're glad that you're here :)

C++ Inheritance

Access Specifiers:

- `public:` visible to all other classes
- `protected:` visible to this class and its *derived* classes
- `private:` visible only to the current class

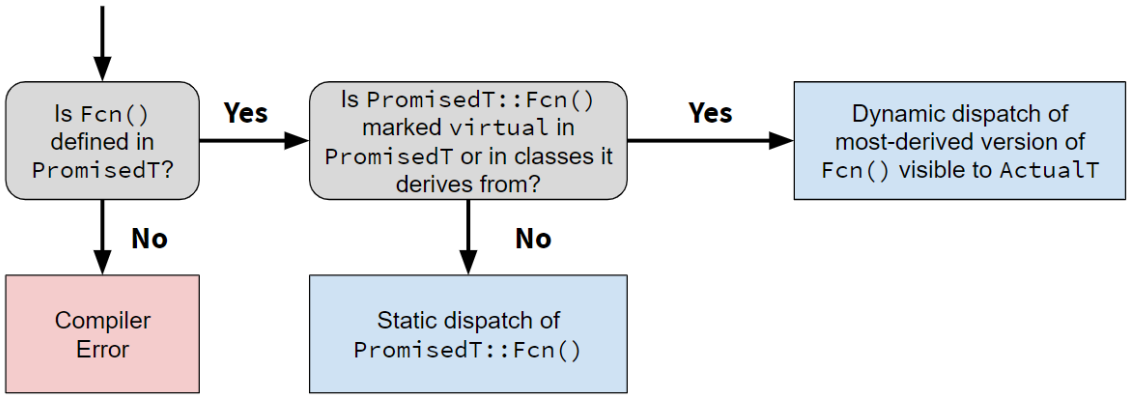
What's different in C++ (compared to Java)?

- *Static vs. dynamic dispatch* – in Java, all method calls are dynamic dispatch
- Pure virtual functions, abstract classes, why no Java "interfaces"
- Assignment slicing, using class hierarchies with STL

`virtual` keyword: Prefix a member function's declaration with this to use dynamic dispatch. Note: derived (child) functions don't need to repeat the virtual keyword, but it traditionally often do.

`override` keyword (C++11): Postfix a member function's declaration with this to tell the compiler that this method should be overriding an inherited virtual function – good to use if available.

```
PromisedT *ptr = new ActualT();
ptr->Fcn();  // which version is called?
```



Exercise:

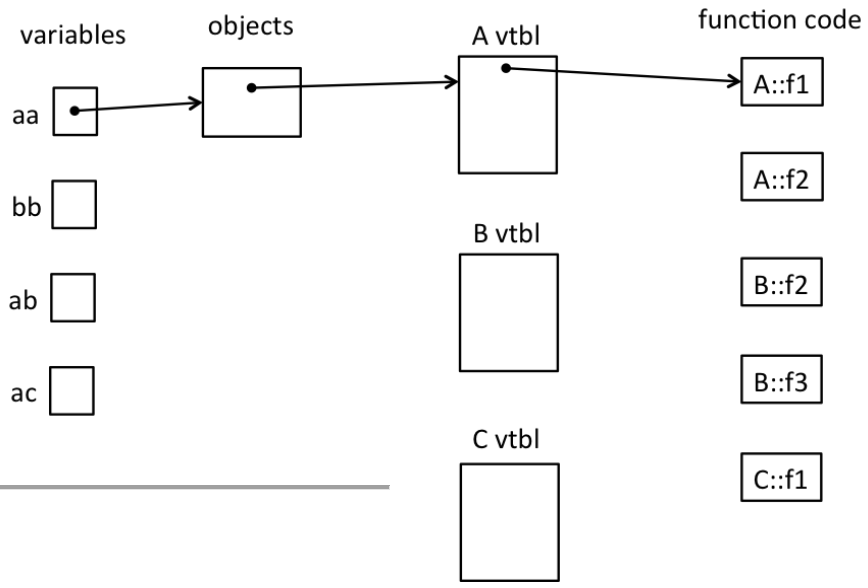
1) Inheritance & Virtual Function

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:

Ex2. 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"; }
         void m4() { cout << "p";}
};

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

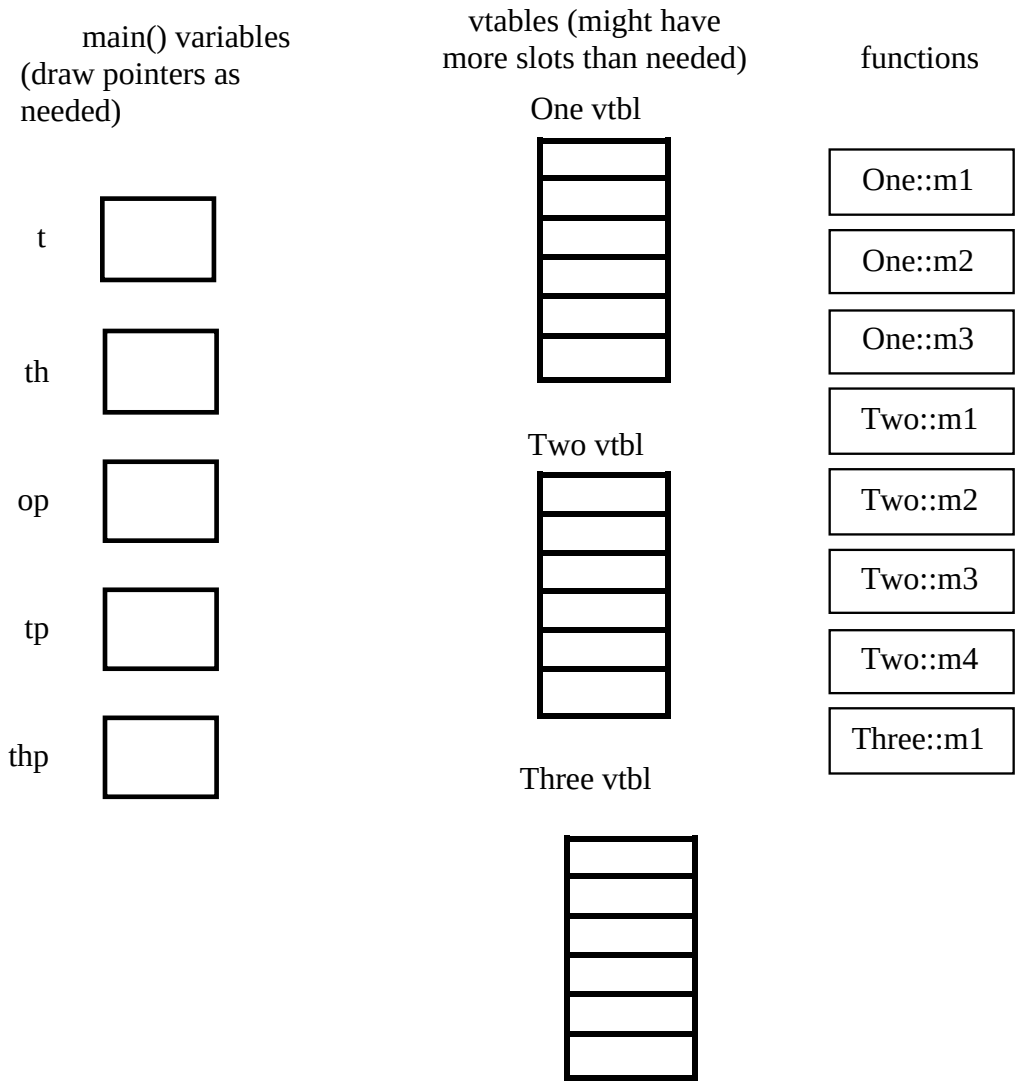
```
int main() {
    Two t;
    Three th;
    One *op = &t;
    Two *tp = &th;
    Three *thp = &th;

    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?

(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.

(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.



C++ Templates

An example converting an existing function to use templates is below (notice that in the template version N is also passed in via template parameter whereas in the regular version it is a parameter):

Non-Template:

```
int modulo(int arg, int n) {
  int result = arg % n;
  return result;
}
```

Template:

```
template<typename T, int N = 2>
T modulo(T arg) {
  T result = arg % N;
  return result;
}
```

Exercise:

3) Templates & Things

This C++ code defines a class that implements a linked list of integers and a small main program that uses it. Convert the `List` class below into a template that can store lists of any values, not just `int`s (*i.e.* use the template parameter `T` instead of `int`). Mark the necessary changes directly on the code (including in `main`).

```
#include <iostream>
using namespace std;
class List {
 public:
  // construct empty list
  List() : head_(nullptr) { }

  // add new node with value n to the front of the list
  virtual void add(int n) {
    Link *p = new Link(n, head_);
    head_ = p;
  }

 private:
  struct Link {  // nodes for the linked list
    int val;
    Link * next;
    Link(int n, Link* nxt): val(n), next(nxt) { }
  };
  // List instance variable
  Link *head_;  // head of list or nullptr if list is empty
};  // end of List class

int main() {
  List nums;
  nums.add(1);
  nums.add(2);
  return EXIT_SUCCESS;
}
```

C++'s Standard Template Library (STL)

Containers, iterators, algorithms (sort, find, etc.), numerics

- ***general*** – `.begin()`, `.end()`, `.size()`, `.erase()`
- ***template <class T> class std::vectors*** – `.operator[]()`, `.push_back()`, `.pop_back()`
- ***template <class T> class std::list*** – `.push_back()`, `.pop_back()`, `.push_front()`, `.pop_front()`, `.sort()`
- ***template <class Key, class T> class std::map*** – `.operator[]()`, `.insert()`, `.find()`, `.count()`
- ***template <class T1, class T2> struct std::pair*** – `.first`, `.second`

Exercises:

4) Standard Template Library

Complete the function `ChangeWords` below. This function has as inputs a vector of strings, and a map of <string, string> key-value pairs. The function should return a new vector<string> value (not a pointer) that is a copy of the original vector except that every string in the original vector that is found as a key in the map should be replaced by the corresponding value from that key-value pair.

Example: if vector `words` is `{"the", "secret", "number", "is", "xlii"}` and map `subs` is `{{"secret", "magic"}, {"xlii", "42"}}`, then `ChangeWords(words, subs)` should return a new vector `{"the", "magic", "number", "is", "42"}`.

Hint: Remember that if `m` is a map, then referencing `m[k]` will insert a new key-value pair into the map if `k` is not already a key in the map. You need to be sure your code doesn't alter the map by adding any new key-value pairs. (Technical nit: `subs` is not a const parameter because you might want to use its `operator[]` in your solution, and `[]` is not a const function. It's fine to use `[]` as long as you don't actually change the contents of the map `subs`.)

Write your code below. Assume that all necessary headers have already been written for you.

```
using namespace std;
vector<string> ChangeWords(const vector<string> &words,
                           map<string,string> &subs) {
```

```
}
```

5) STL Debugging

Here is a little program that has a small class `Thing` and main function (assume that necessary `#includes` and `using namespace std;` are included).

```
class Thing {
 public:
  Thing(int n): n_(n) { }
  int getThing() const { return n_; }
  void setThing(int n) { n_ = n; }
 private:
  int n_;
};

int main() {
  Thing t(17);
  vector<Thing> v;
  v.push_back(t);
}
```

This code compiled and worked as expected, but then we added the following two lines of code (plus the appropriate `#include <set>`):

```
set<Thing> s;
s.insert(t);
```

The second line (`s.insert(t)`) failed to compile and produced dozens of spectacular compiler error messages, all of which looked more-or-less like this (edited to save space):

```
In file included from string:48:0, from bits/locale_classes.h:40, from
bits/ios_base.h:41,from ios:42,from ostream:38, from /iostream:39,from
thing.cc:3: bits/stl_function.h: In instantiation of 'bool
std::less<_Tp>::operator()(const _Tp&, const _Tp&) const [with _Tp =
Thing]': <<many similar lines omitted>> thing.cc:37:13: required from
here bits/stl_function.h:
387:20: error: no match for 'operator<' (operand types are 'const
Thing' and 'const Thing') { return __x < __y; }
```

What on earth is wrong? Somehow class `Thing` doesn't work with `set<Thing>` even though `insert` is the correct function to use here. (a) What is the most likely reason, and (b) what would be needed to fix the problem? (Be brief but precise – you don't need to write code in your answer, but you can if that helps make your explanation clear.)