333 Section 7 - C++ Templates, STL, and Inheritance

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

C++ Inheritance

Access Specifiers:

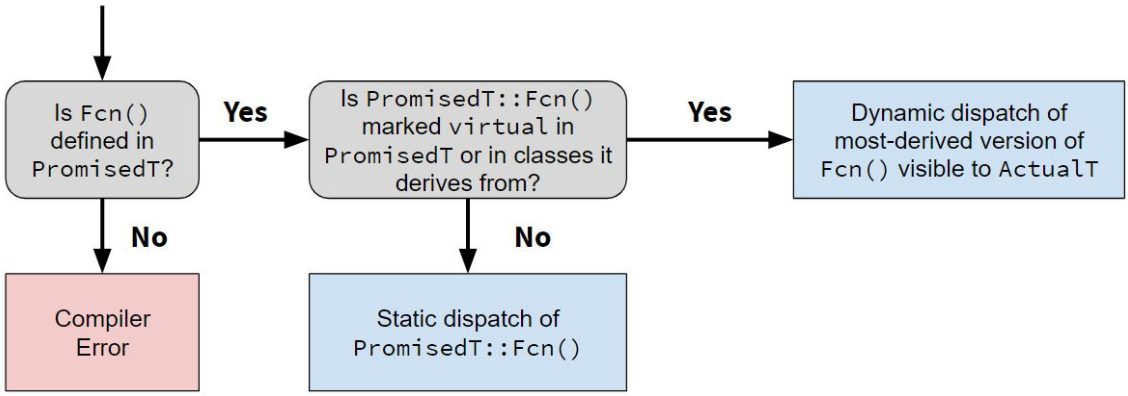
- `public:` visible to all other classes
- `protected:` visible to this class and its *derived* classes
- `private:` visible only to the current class

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).

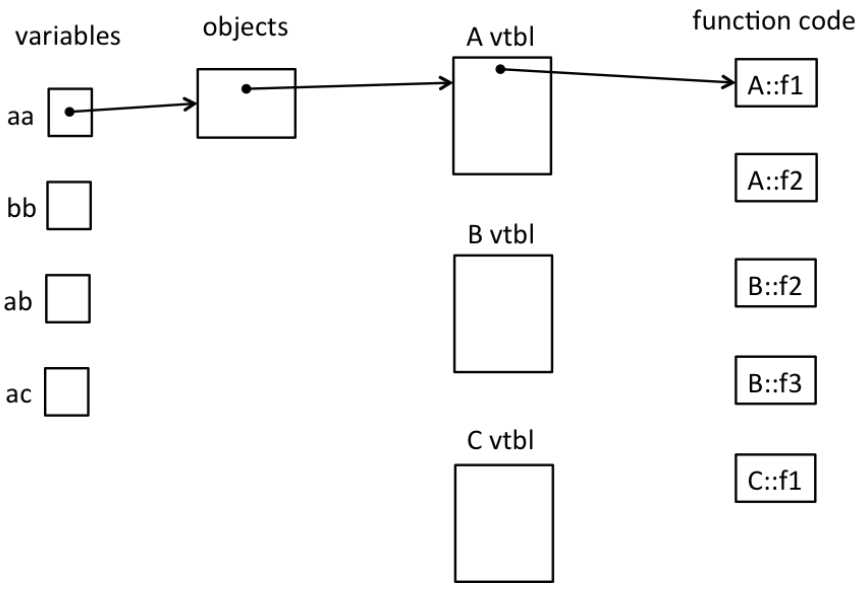
(a) Complete the diagram on the next page by adding the remaining objects and all of the additional pointers needed to link variables, objects, virtual function tables, and function bodies. Be sure that the order of pointers in the virtual function tables is clear (i.e., which one is first, then next, 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:

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:

2) 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:

3) 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) {









}
```

4) STL Debugging

Here is a little program that has a small class `Thing` and main function (assume that necessary `#include`s 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.)

Question 3: C++ STL [20 pts]

For this question, you will use the following Date class:

```
class Date {
 public:
  int month;      // Valid values: 1 - 12
  int day;        // Valid values: 1 - 31

  bool operator==(const Date &other) {
    return month == other.month && day == other.day;
  }
}
```

You are building a C++ class called `LectureSchedule` to keep track of upcoming guest lectures. The class should have an STL map called `lectures_` as its (private) field, which maps from a C++ string object (representing the name of the guest lecturer) to an STL vector containing a list of `Date` objects. You can assume that `using namespace std;` has been written at the top of the file.

a) [4 pts] Write the declaration of the `lectures_` field. For this problem, you should store copies of the data itself -- do not store pointers of any kind in the STL containers.

b) [10 pts] Assume any necessary constructors have been implemented for `LectureSchedule`. Write the implementation of a function called `AddLecture` that takes 2 arguments: a C++ string object representing the lecturer name, and a `Date` object (not a pointer) representing the date of a lecture for them. It should add a mapping from that lecturer to that date accordingly, unless the class already stores a mapping from that lecturer to that date (in which case it should do nothing). The function should return `0` if the mapping was added, or `-1` if the mapping already existed.

```
int LectureSchedule::AddLecture(const string &name, const Date &date) {




}
```

c) [6 pts] Now, instead of storing a copy of the data itself in the `lectures_` field, suppose we want to store pointers to `Date` objects on the heap. For each of the following possible options, indicate whether it would work as intended, would cause a memory leak, or would not work correctly. Assume that the only changes made would be the field type and adding code to dereference the pointer as appropriate whenever accessing the data (even if that requires multiple steps). In particular, the `LectureSchedule` class has no destructor defined beyond the synthesized one.

`raw pointer (Date *)`	`unique_ptr<Date>`	`shared_ptr<Date>`	`weak_ptr<Date>`
WORKS	**WORKS**	**WORKS**	**WORKS**
MEMORY LEAK	**MEMORY LEAK**	**MEMORY LEAK**	**MEMORY LEAK**
INCORRECT	**INCORRECT**	**INCORRECT**	**INCORRECT**

Question 4: Networking [22 pts]

a) [4 pts] In the sockets API we looked at in class, the `sockaddr_in` struct represents an IPv4 address and `sockaddr_in6` represents an IPv6 address. In this question, for each of the structs listed below, describe what they are typically used for in networking code (i.e. why it's important for them to exist even though there is already `sockaddr_in` and `sockaddr_in6`).

i) `struct sockaddr` (Hint: usually used as `struct sockaddr *`)

ii) `struct sockaddr_storage`

b) [4 pts] For each of the following behaviors, identify what networking layer is most closely thought of as being responsible for handling that behavior.

i) Host A tries to send a long message to Host B in another city, broken up into many packets. A packet in the middle does not arrive, so Host A sends it again.

ii) Host A tries to send a message to Host B, but Host C and Host D are also trying to communicate on the same network, so Host A has to avoid interfering.