

CSE 333

Section 6

Inheritance & VTable, HW3 Overview, maybe (Templates and STL)

Logistics

Friday (tomorrow)

Exercise 14 @ 10:30 am

Monday

Exercise 13 @ 10:30 am

Wednesday:

NO EXERCISE DUE!!!

Thursday:

HW3 @ 11:59 pm

Section Plan

- Inheritance & VTables
- HW 3 Overview
- STL & Templates (if we have time)

Inheritance

- **Derived** class inherits from the **base** class
 - In 333, we always use *public* inheritance
 - Inherits all *non-private* member variables
 - Inherits all *non-private* member functions
except for ctor, cctor, dtor, op=
- Access specifiers revisited:
 - **Private** members cannot be accessed by derived classes
 - **Protected** members are available to base & derived

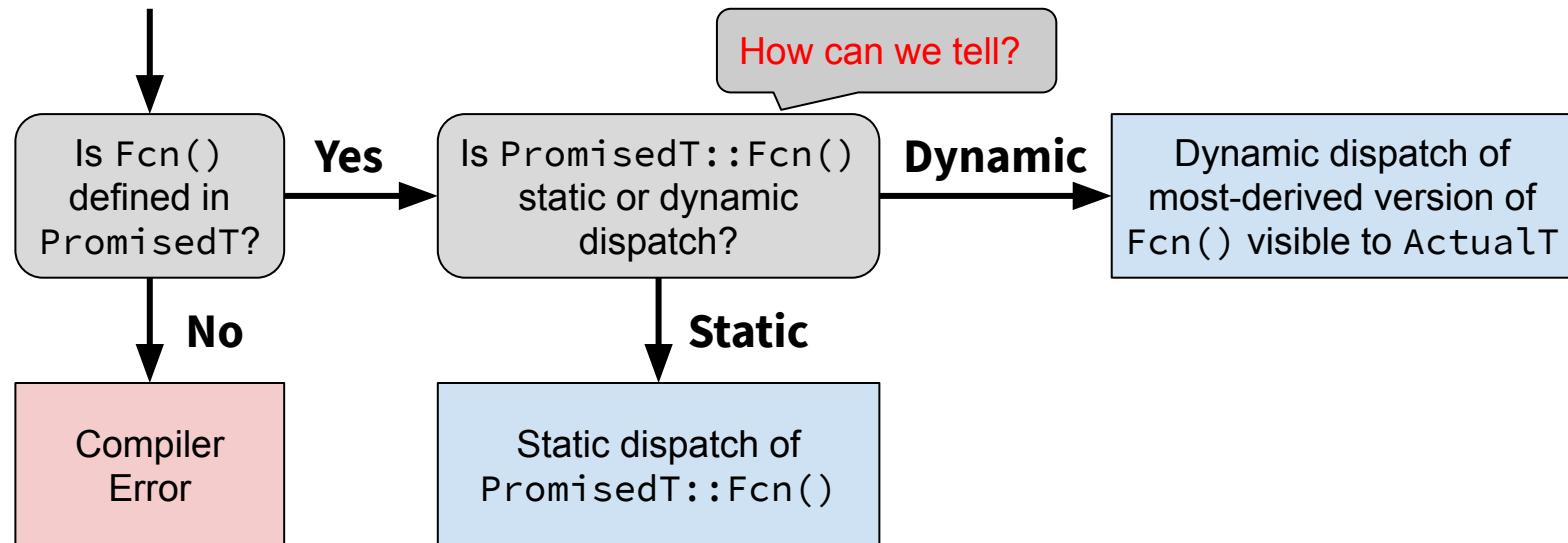
Static vs. Dynamic Dispatch

How to resolve invoking a method via a polymorphic pointer:

1. Static dispatch
 - Default behavior in C++
 - More to come in Friday's lecture!
2. Dynamic dispatch
 - Which implementation is determined *at runtime* via lookup
 - Compiler generates code that accesses function pointers added to the class

Dispatch Decision Tree

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



Dispatch Keywords

- **virtual** – request dynamic dispatch
 - Is “sticky”: overridden virtual method in derived class is still virtual with or without the keyword
- **override** – ensures that the function is virtual and is overriding a virtual function from a base class (`@override` in Java)
 - Generates a compiler error if conditions are not met
 - Catches overloading vs. overriding bugs at compile time

Practice: static, dynamic, or error?

```
class Base {  
    void Foo();           //  
    void Bar();           //  
    virtual void Baz();   //  
};
```

```
class Derived : public Base {  
    virtual void Foo();   //  
    void Bar() override; //  
    void Baz();          //  
};
```

Practice: static, dynamic, or error?

```
class Base {  
    void Foo();           // static dispatch  
    void Bar();           // static dispatch  
    virtual void Baz();   // dynamic dispatch  
};  
  
class Derived : public Base {  
    virtual void Foo();   //  
    void Bar() override; //  
    void Baz();          //  
};
```

Practice: static, dynamic, or error?

```
class Base {  
    void Foo();           // static dispatch  
    void Bar();           // static dispatch  
    virtual void Baz();   // dynamic dispatch  
};  
  
class Derived : public Base {  
    virtual void Foo();   // now dynamic (for more derived)  
    void Bar() override; //  
    void Baz();          //  
};
```

Practice: static, dynamic, or error?

```
class Base {  
    void Foo();           // static dispatch  
    void Bar();           // static dispatch  
    virtual void Baz();   // dynamic dispatch  
};  
  
class Derived : public Base {  
    virtual void Foo();   // now dynamic (for more derived)  
    //void Bar() override; // compiler error  
    void Bar();           // static dispatch  
    void Baz();           //  
};
```

Practice: static, dynamic, or error?

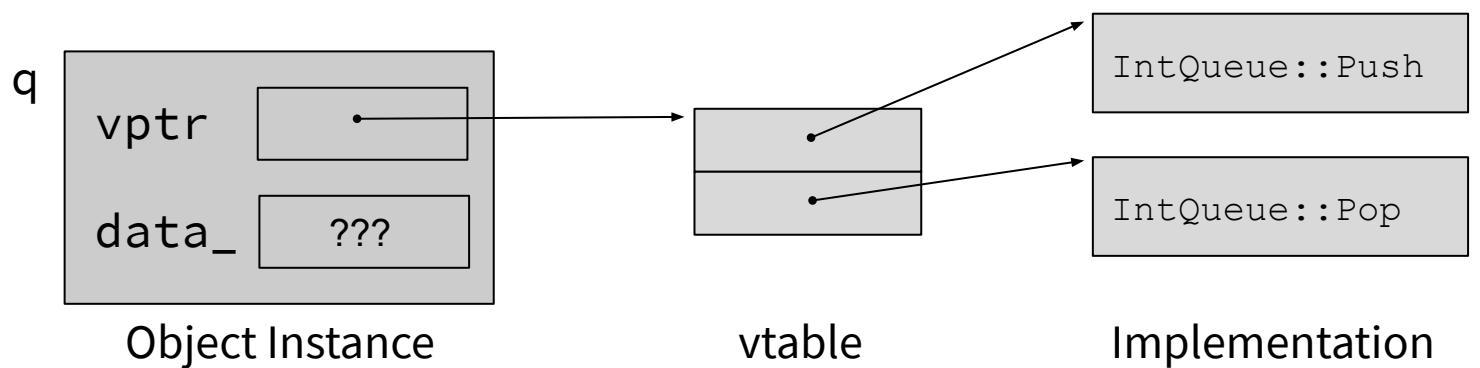
```
class Base {  
    void Foo();           // static dispatch  
    void Bar();           // static dispatch  
    virtual void Baz();   // dynamic dispatch  
};  
  
class Derived : public Base {  
    virtual void Foo();   // now dynamic (for more derived)  
    //void Bar() override; // compiler error  
    void Bar();           // static dispatch  
    void Baz();           // still dynamic (sticky!)  
};
```

Vtable (Virtual Function Table) & Vptr (Vtable pointer)

- vtable: An array of function pointers defined for each class that has at least one virtual method to enable dynamic dispatch
 - One per class
- vptr: Each class object instance has a pointer to that vtable
 - One per object instance

Vtable Diagrams

```
class IntQueue {  
public:  
    virtual void Push(int x);  
    virtual int Pop();  
private:  
    vector<int> data_;  
};  
IntQueue q;
```

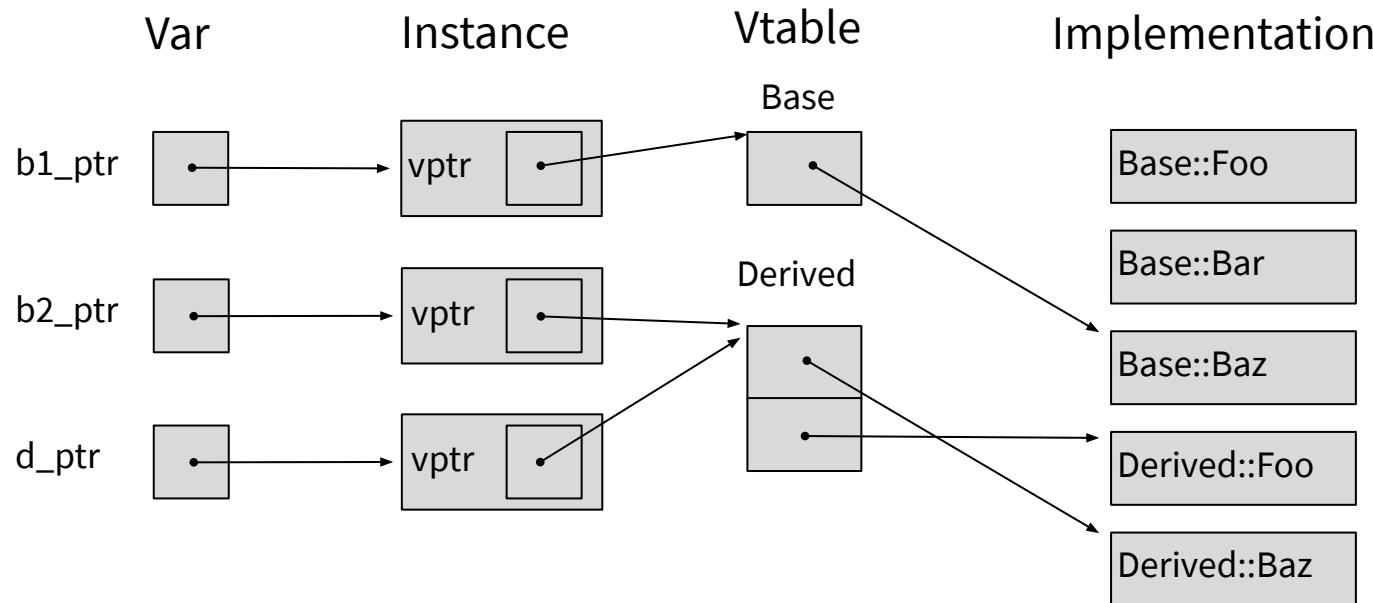


Vtable Diagrams

```
Base *b1_ptr = new Base;           };  
Base *b2_ptr = new Derived;        };  
Derived *d_ptr = new Derived;
```

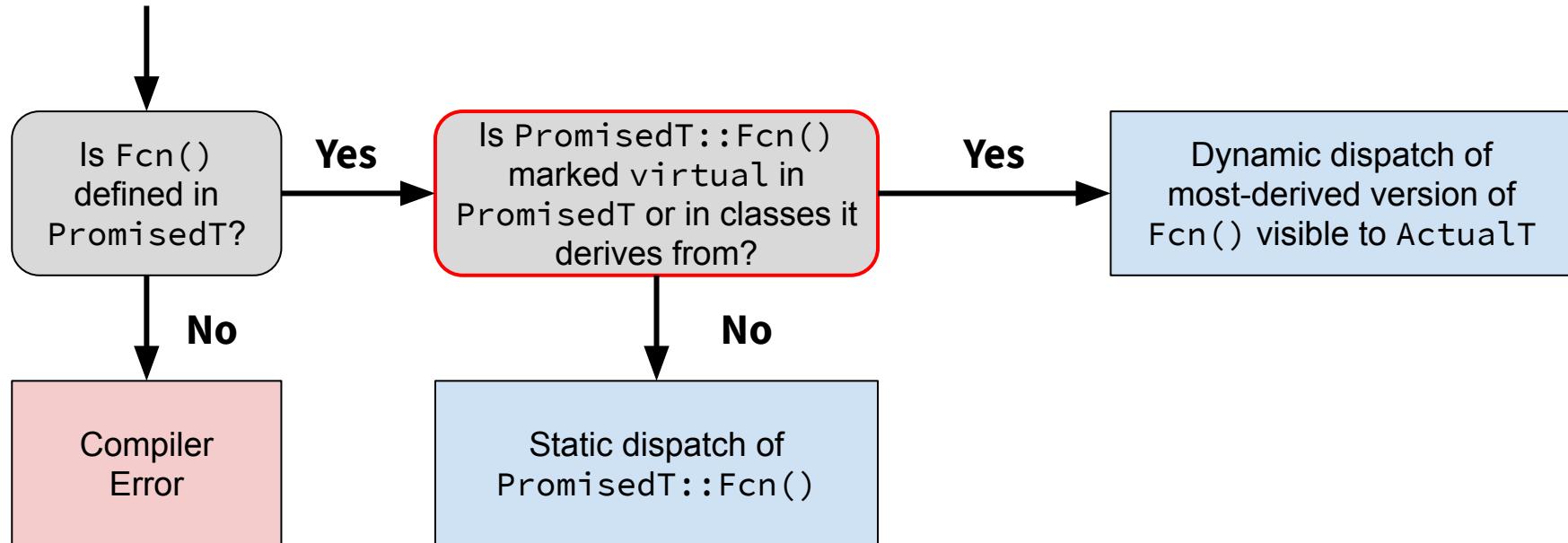
```
class Base {  
    void Foo();  
    void Bar();  
    virtual void Baz();  
};
```

```
class Derived :  
public Base {  
    virtual void Foo();  
    void Baz();  
};
```



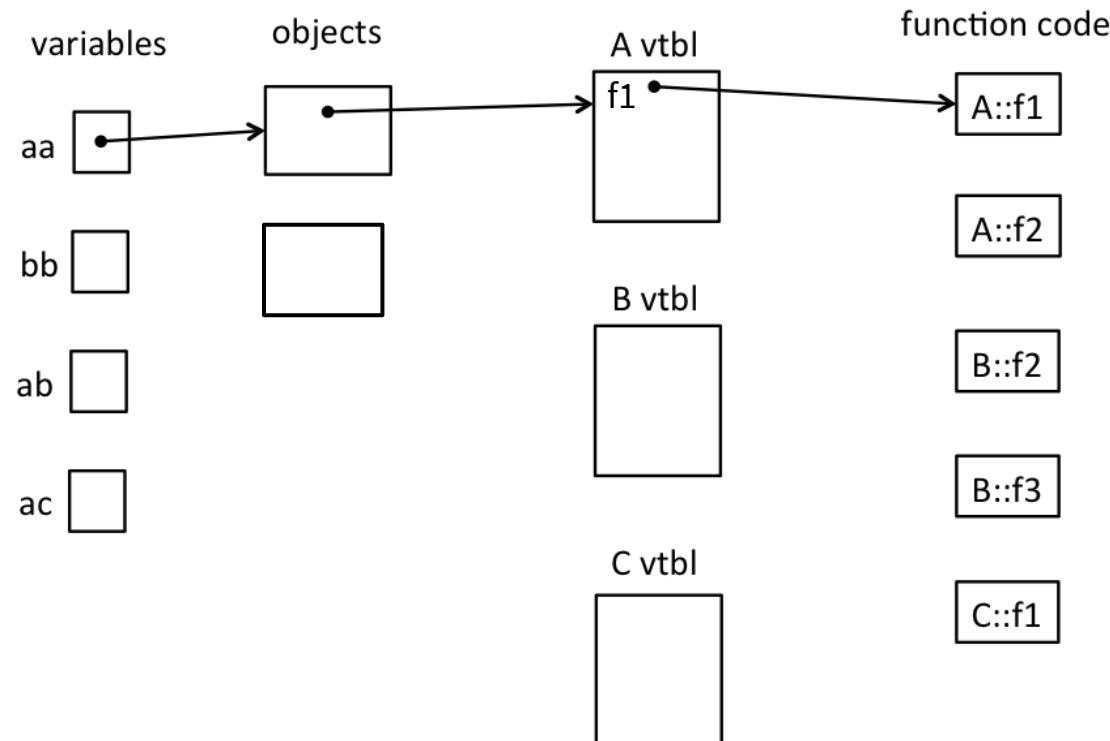
Updated Dispatch Decision Tree

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

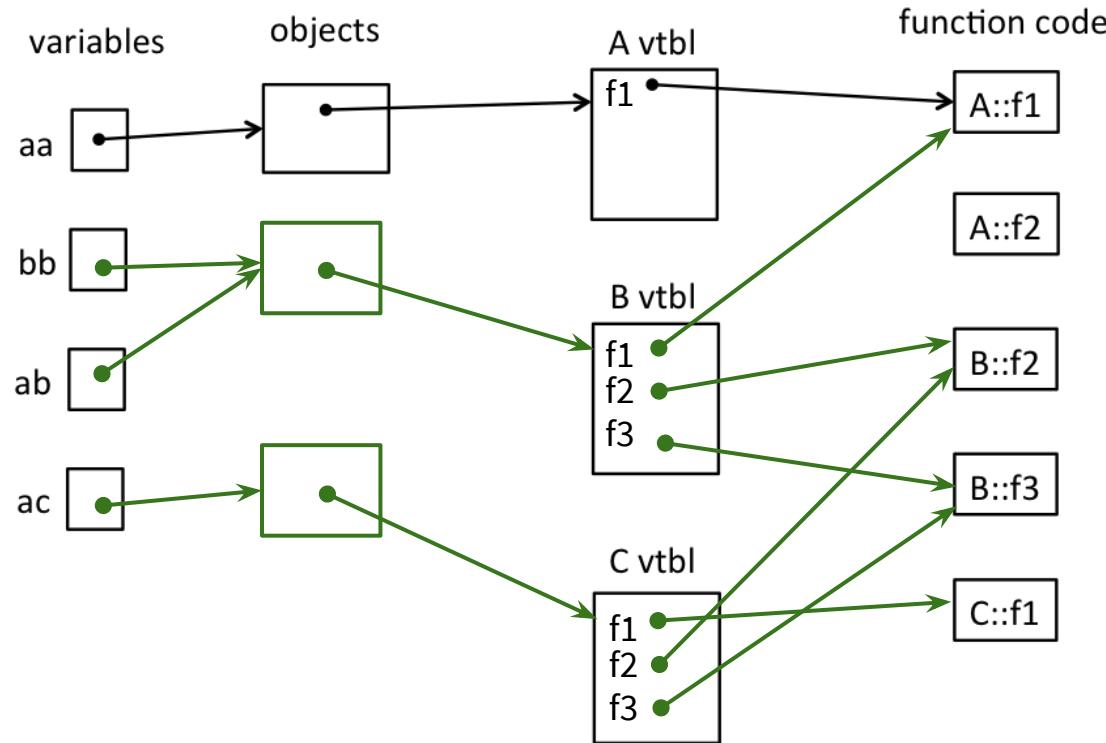


Exercise 1!

Exercise 1



Exercise 1 Solution (pointers)



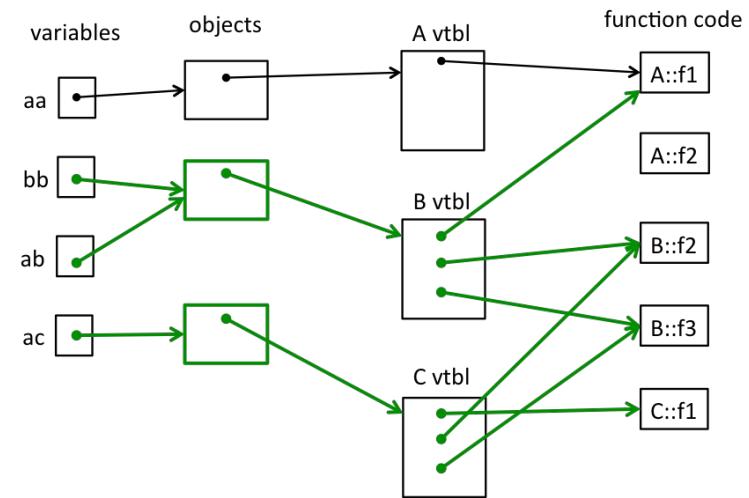
Exercise 1 Solution (output)

```
#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; }
};
```



```
A* aa = new A();
```

```
aa->f1();
```

```
A::f2
```

```
A::f1
```

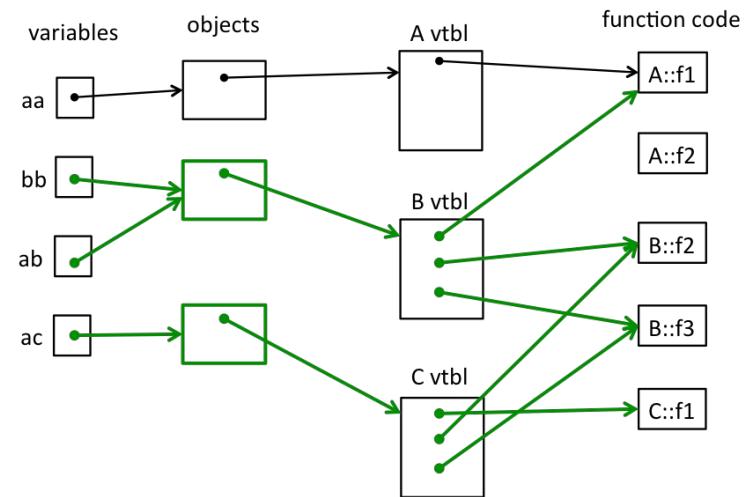
Exercise 1 Solution (output)

```
#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; }
};
```



B* bb = new B();

bb->f1();

A::f2

A::f1

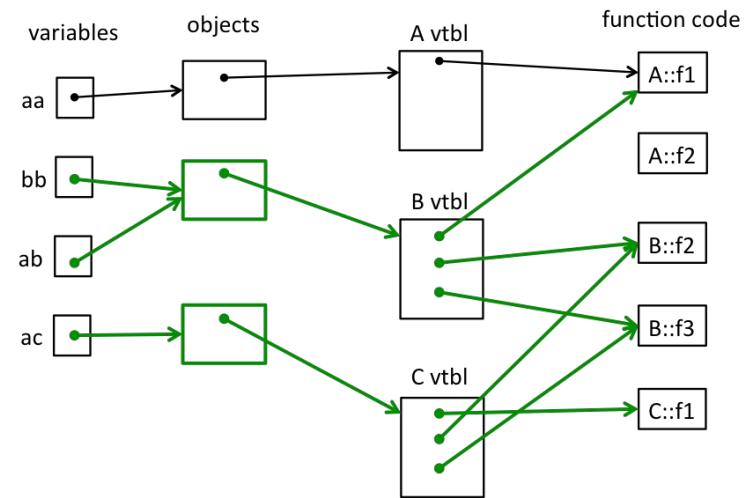
Exercise 1 Solution (output)

```
#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; }
};
```



```
B* bb = new B();
A* ab = bb;
```

```
bb->f2();
cout << "----" << endl;
ab->f2();
```

B::f2

A::f2

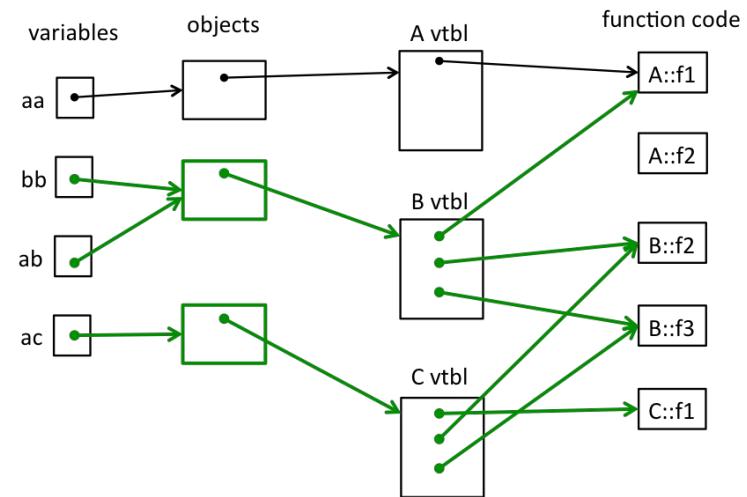
Exercise 1 Solution (output)

```
#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; }
};
```



B* bb = new B();

bb->f3();

A::f2

A::f1

B::f3

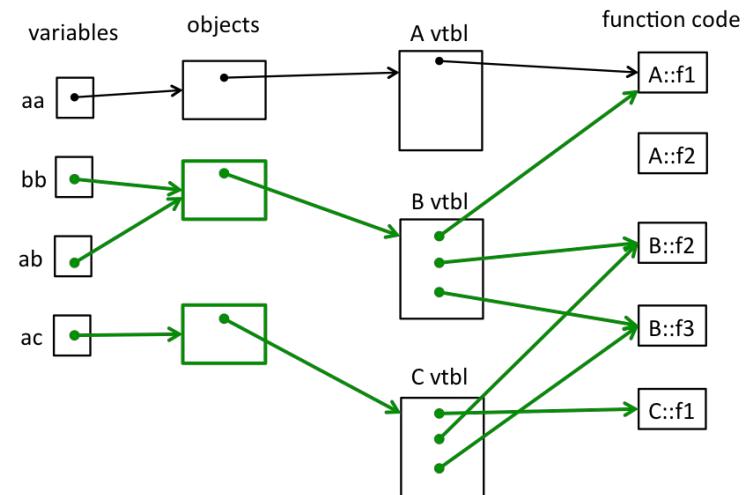
Exercise 1 Solution (output)

```
#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; }
};
```



A* ac = new C();

ac->f1();

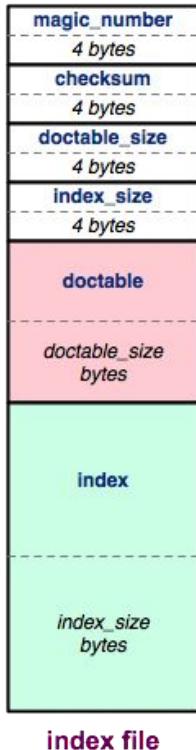
B::f2

C::f1

Any Questions on Inheritance & VTable?

HW 3 Overview!

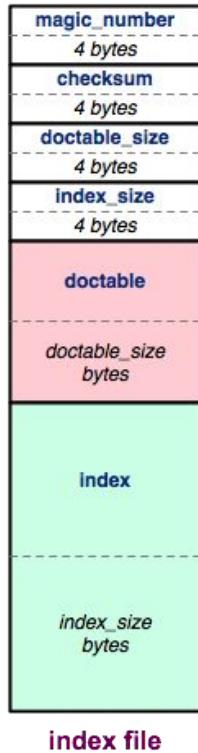
Index File



Crawling a file tree in HW2 takes a long time.

To save time, write the completed DocTable and MemIndex to a File!

Index File Components



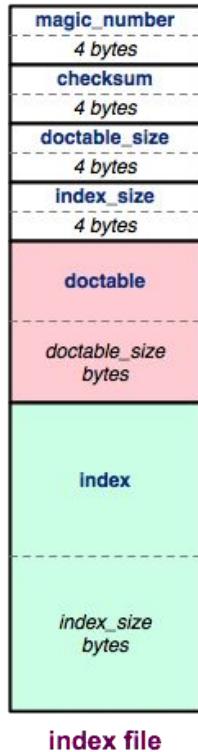
Header (metadata)

DocTable

MemIndex

index file

Index File Header

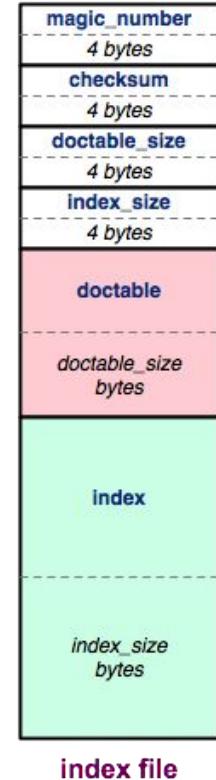


- **magic_number**: 0xCAFEF00D
- **checksum**: mathematical signature
- **doctable_size**: in bytes
- **index_size**: in bytes

Index File Header - HEX

- Find a hex editor/viewer of your choice
 - xxd <indexfile>
 - hexdump -vC <indexfile>

```
0000000: cafe f00d 1c42 4620 0000 205b 0000 075d .....BF .. [...]
00000010: 0000 0400 0000 0000 0000 2014 0000 0001 ..... .
00000020: 0000 2014 0000 0001 0000 2031 0000 0001 ..... 1...
00000030: 0000 204e 0000 0000 0000 206b 0000 0000 .. N..... k...
00000040: 0000 206b 0000 0000 0000 206b 0000 0000 .. k..... k...
00000050: 0000 206b 0000 0000 0000 206b 0000 0000 .. k..... k...
```



The header:

Magic word Checksum Doctable size Index size

man xxd

man hexdump

Byte Ordering and Endianness

- Network (Disk) Byte Order (Big Endian)
 - The most significant byte is stored in the highest address
- Host byte order
 - Might be big or little endian, depending on the hardware
- To convert between orderings, we can use
 - `uint32_t htonl (uint32_t hostlong); // host to network`
 - `uint32_t ntohl (uint32_t hostlong); // network to host`
- Pro-tip:

The structs in HW3 have `toDiskFormat()` and `toHostFormat()` functions that will convert endianness for you.

Hex View

- emacs “M-x hexl-mode”

```
File Edit Options Buffers Tools Hexl Help
87654321 0011 2233 4455 6677 8899 aabb ccdd eeff 0123456789abcdef
00000000: cafe f00d ce52 0578 0000 205e 0000 0944 .....R.x.. ^...D
00000010: 0000 0400 0000 0000 0000 2014 0000 0001 .....
00000020: 0000 2014 0000 0001 0000 2032 0000 0001 .. .... 2...
00000030: 0000 2050 0000 0000 0000 206e 0000 0000 .. P..... n...
00000040: 0000 206e 0000 0000 0000 206e 0000 0000 .. n..... n...
00000050: 0000 206e 0000 0000 0000 206e 0000 0000 .. n..... n...
```

- vim “:%!xxd”

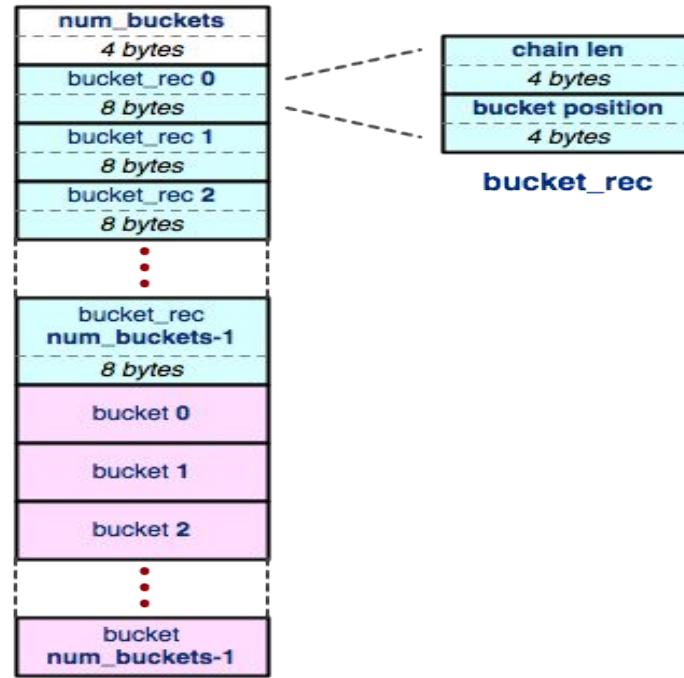
```
00000000: cafe f00d 1c42 4620 0000 205b 0000 075d .....BF .. [...]
00000010: 0000 0400 0000 0000 0000 2014 0000 0001 .....
00000020: 0000 2014 0000 0001 0000 2031 0000 0001 .. .... 1...
00000030: 0000 204e 0000 0000 0000 206b 0000 0000 .. N..... k...
00000040: 0000 206b 0000 0000 0000 206b 0000 0000 .. k..... k...
00000050: 0000 206b 0000 0000 0000 206b 0000 0000 .. k..... k...
```

DocTable & MemIndex

- At their core, both DocTable & MemIndex are HashTables.
- Lets first look at how we write a HashTable.

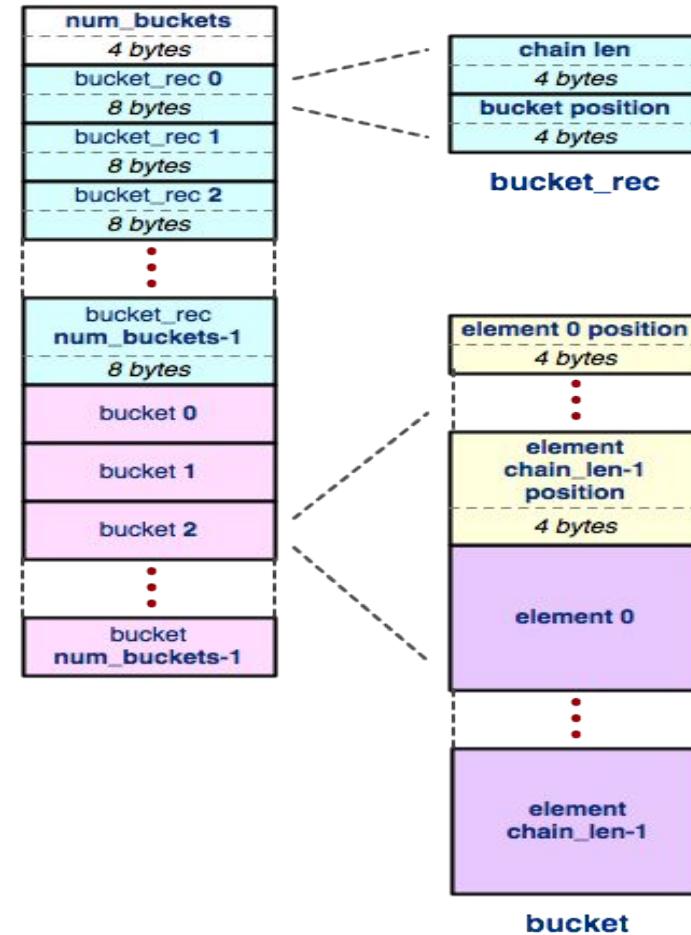
HashTable

- HashTable can have varying amount of buckets, so start with num_buckets.
- Buckets can be of varying lengths. To know the offset, we store some bucket records.



Buckets

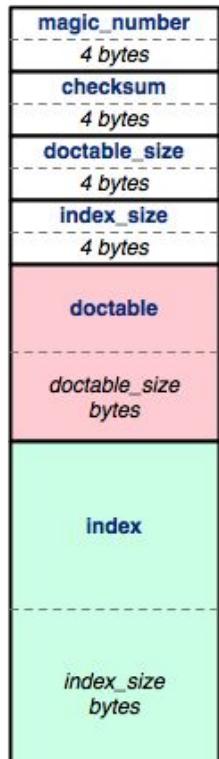
- A bucket is a list that contains elements in the table. Offset to a bucket is found in a bucket record.
- Elements can be of various sizes, so we need to store element positions to know where each element is.



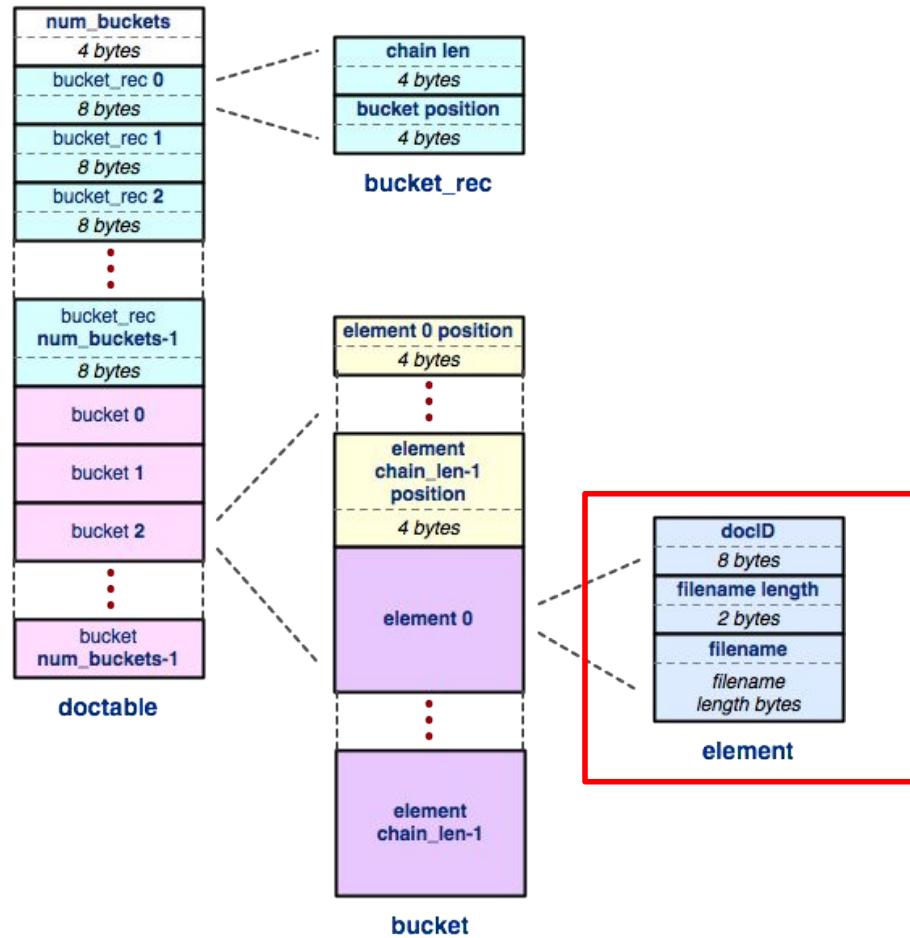
DocTable & MemIndex

- At their core, both DocTable & MemIndex are HashTables.
- The difference between DocTable and MemIndex is entirely what type of element is stored in them.

doctable

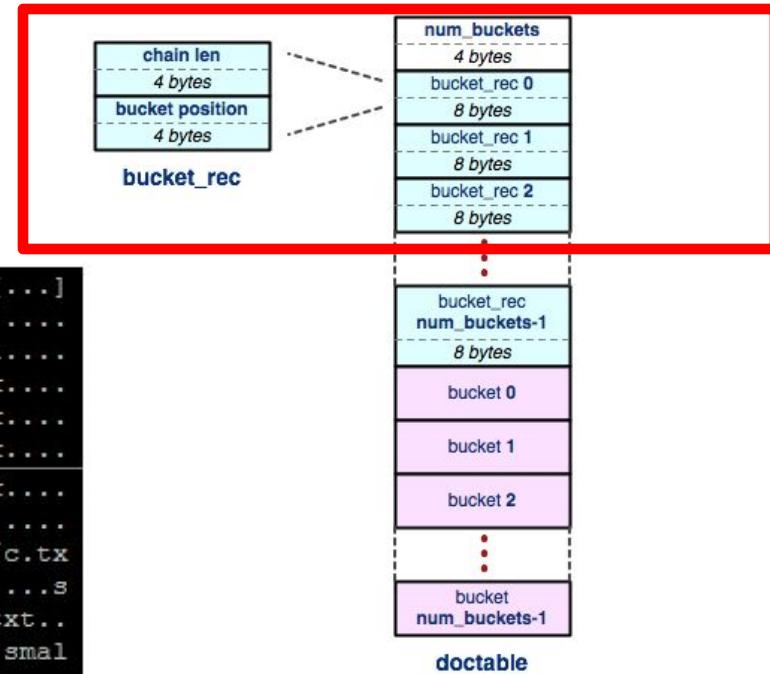


index file



DocTable (Hex)

0000000:	cafe	f00d	1c42	4620	0000	205b	0000	075dBF	..	[...]
0000010:	0000	0400	0000	0000	0000	2014	0000	0001
0000020:	0000	2014	0000	0001	0000	2031	0000	0001	1....
0000030:	0000	204e	0000	0000	0000	206b	0000	0000	.. N.....	k....
0000040:	0000	206b	0000	0000	0000	206b	0000	0000	.. k.....	k....
0000050:	0000	206b	0000	0000	0000	206b	0000	0000	.. k.....	k....
0002000:	0000	206b	0000	0000	0000	206b	0000	0000	.. k.....	k....
0002010:	0000	206b	0000	2018	0000	0000	0000	0001	.. k.....
0002020:	000f	736d	616c	6c5f	6469	722f	632e	7478	.. small_dir/c.tx
0002030:	7400	0020	3500	0000	0000	0000	0200	0f73	t.. 5.....	s
0002040:	6d61	6c6c	5f64	6972	2f62	2e74	7874	0000	mall_dir/b.txt..
0002050:	2052	0000	0000	0000	0003	000f	736d	616c	R.....	smal
0002060:	6c5f	6469	722f	612e	7478	7400	0000	8000	l_dir/a.txt.....
0002070:	0000	0000	0024	6f00	0000	0000	0024	6f00\$o.....\$o.



The header

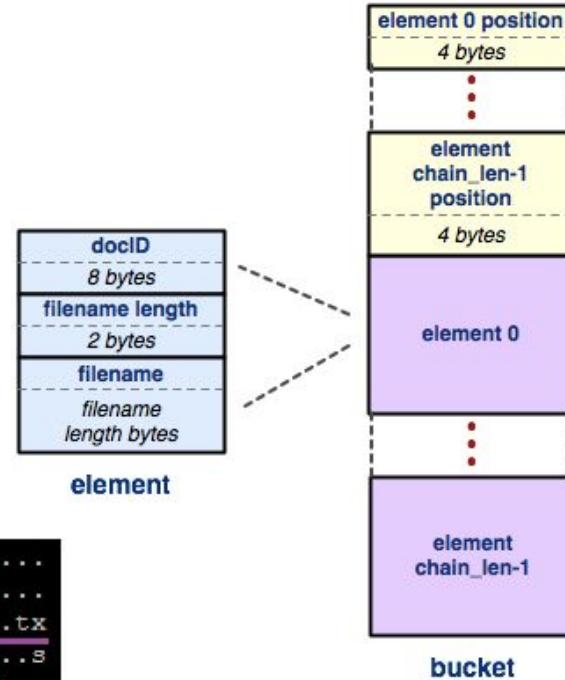
Num buckets (Chain len Bucket offset)*

doctable

```

0002000: 0000 206b 0000 0000 0000 206b 0000 0000 .. k..... k....
0002010: 0000 206b 0000 2018 0000 0000 0000 0001 .. k.. .....
0002020: 000f 736d 616c 6c5f 6469 722f 632e 7478 ..small_dir/c.tx
0002030: 7400 0020 3500 0000 0000 0000 0200 0f73 t.. 5.....s
0002040: 6d61 6c6c 5f64 6972 2f62 2e74 7874 0000 mall_dir/b.txt..

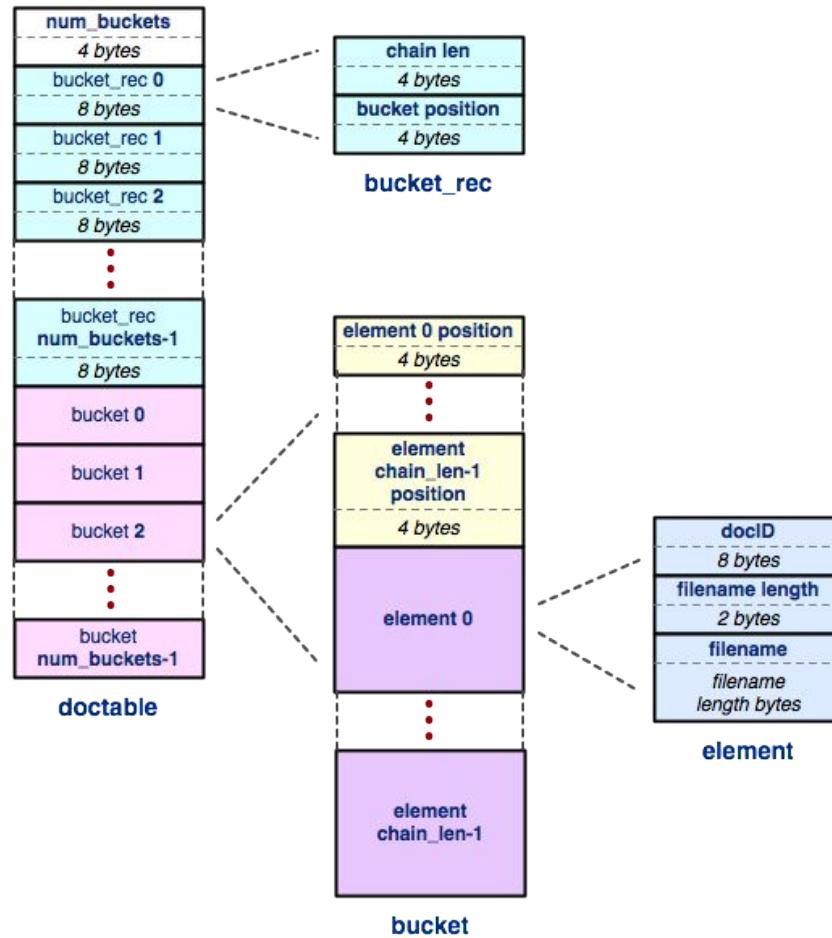
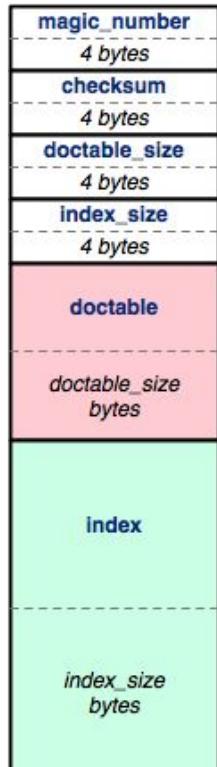
```



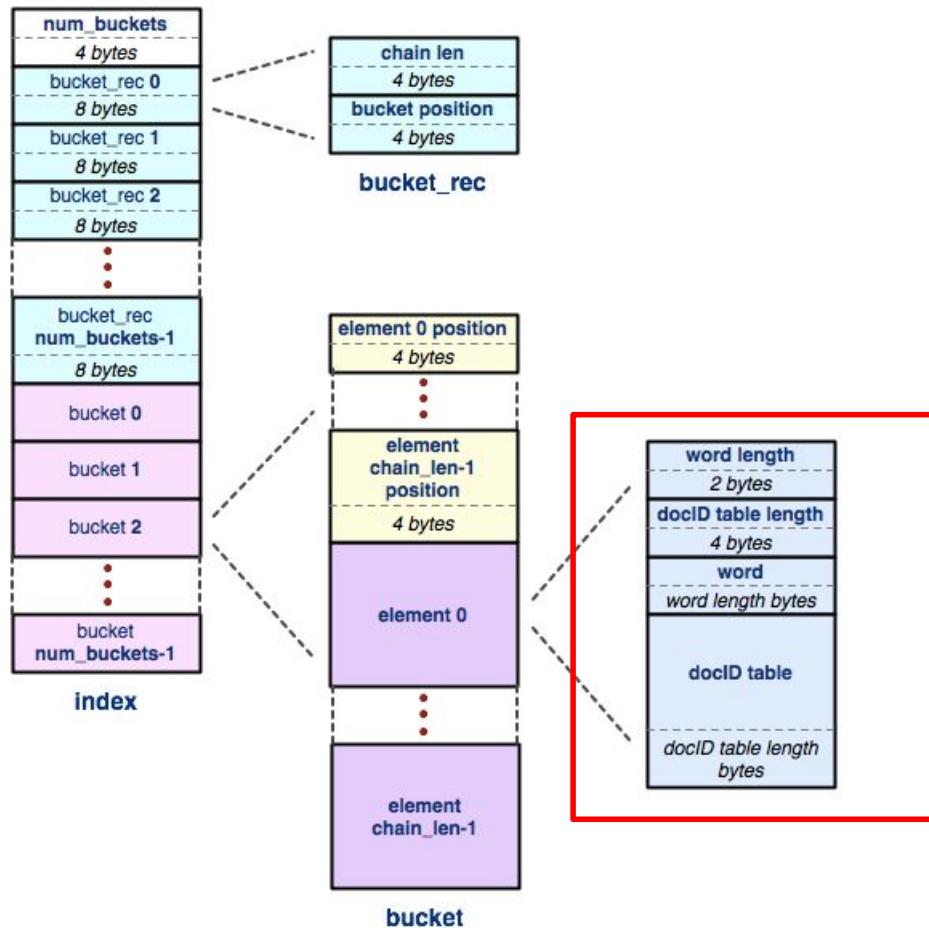
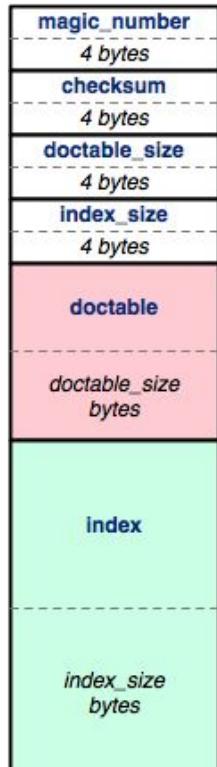
The buckets:

((Element offset)ⁿ (DocID Filename len Filename)ⁿ)*

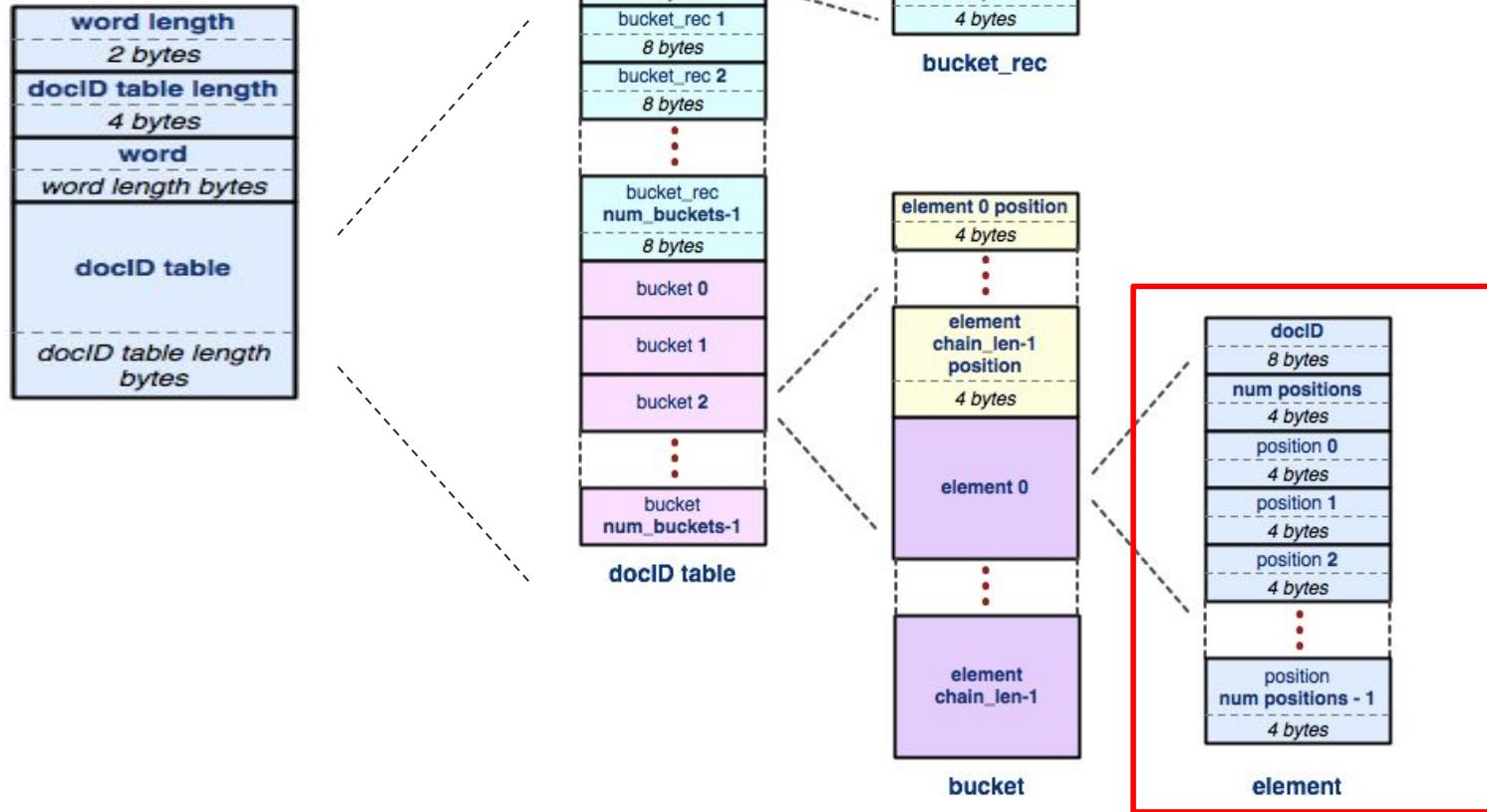
doctable



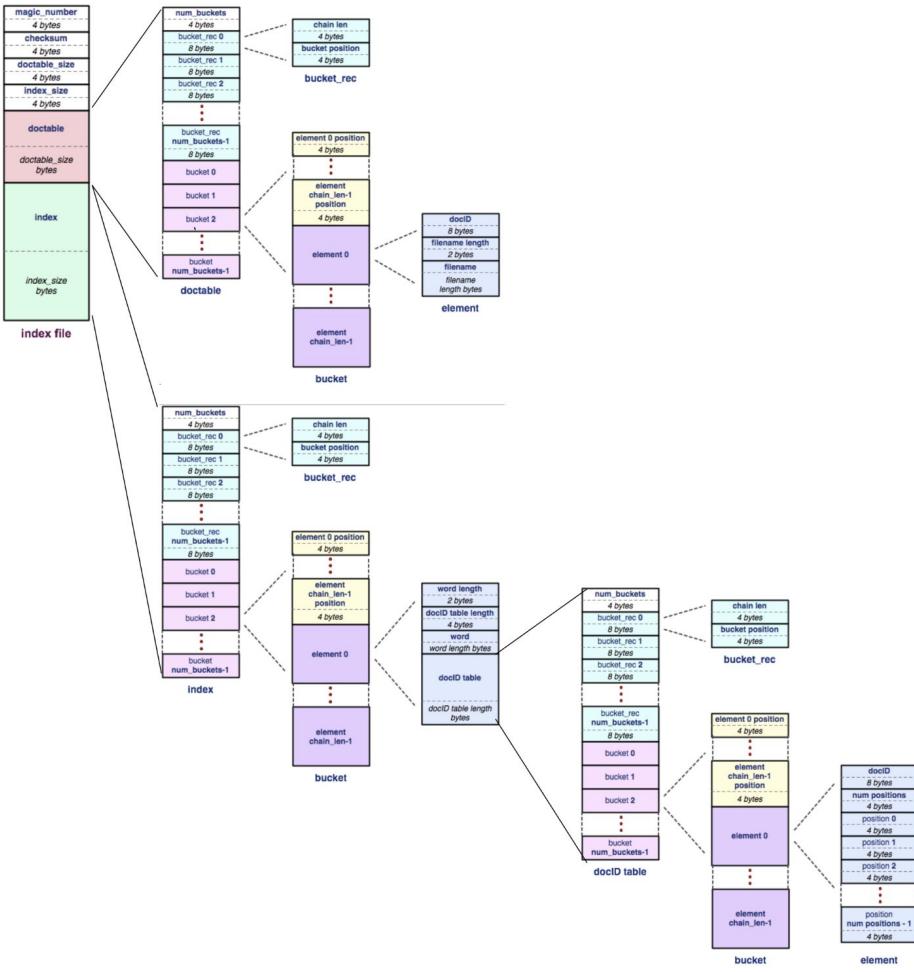
index



docID table



The Full Picture



HW Tips

- When Writing, you should (almost) always:
 1. `.toDiskFormat()`
 2. `fseek()`
 3. `fwrite()`
- When Reading, you should (almost) always:
 1. `fseek()`
 2. `fread()`
 3. `.toHostFormat()`
- The most common bugs in the hw involve forgetting to change byte ordering, or forgetting to `fseek()`.

Exercise 3!

Exercise 3 Solution

```
#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;
    }
}

...
```

Exercise 3 Solution

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

template <typename T>
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;
    }
}

...
```

Exercise 3 Solution

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

template <typename T>
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 T n) {
        Link *p = new Link(n, head_);
        head_ = p;
    }
}

...
```

Exercise 3 Solution

```
...
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;
}
```

Exercise 3 Solution

```
...
private:
    struct Link { // nodes for the linked list
        int T val;
        Link * next;
        Link(int T 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;
}
```

Exercise 3 Solution

```
...
private:
    struct Link { // nodes for the linked list
        int T val;
        Link * next;
        Link(int T 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<int> nums;
    nums.add(1);
    nums.add(2);
    return EXIT_SUCCESS;
}
```

C++ standard lib is built around templates

- Containers
 - Store data
 - Define iterators to go over that data
- Iterators
 - Different flavors (random access, bidirectional, etc)
 - Common interface to containers
- Algorithms
 - Use the common interface of iterators to do things
 - Different algorithms require different ‘complexities’ of iterators

Common STL Data Structures

- `map<Key, Value, Order=std::Less<Key>>`
 - Store key -> value pairs where we can use a key to get the value (TreeMap)
- `set<Item, Order=std::Less<Item>>`
 - An unindexed collection of items
 - When you care most about “do I have this”
- `vector<Item>`
 - Resizable array (ArrayList, in Java)
- `unordered_map<Key, Value, Hash=std::Hash<Key>>`
 - HashMap → use hash of key to order. Usually faster than map
- Assorted others (queue, linkedlist, etc.)

Now what's that 'std::less'? //Out of scope

```
std::less<T>(const T& lhs, const T& rhs) {  
    return lhs < rhs;  
}
```

- Much like in Java, some structures require ordering elements
 - E.g. set is implemented as a binary tree
- Want to let users store custom types.
 - Java uses Comparable, C++ uses operator< (in std::less)
- However, maybe you want to use a different ordering
 - Ordering is templated function so you can substitute
 - E.g. set<int, std::greater<int>> or set<int, myIntCompare>

Exercise 4!

Exercise 4!

Exercises:

2) 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) {
}

}
```

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) {
    vector<string> result;
    for (auto &word : words) {
        if (subs.find(word) != subs.end()) {
            result.push_back(subs[word]);
        } else {
            result.push_back(word);
        }
    }
    return result;
}
```

Exercise 5!

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 {                                int main() {  
public:                                         Thing t(17);  
    Thing(int n): n_(n) { }                      vector<Thing> v;  
    int getThing() const { return n_; }           v.push_back(t);  
    void setThing(int n) { n_ = n; }                }  
private:  
    int n_;  
};
```

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.)

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 {                                int main() {  
public:                                         Thing t(17);  
    Thing(int n): n_(n) { }                      vector<Thing> v;  
    int getThing() const { return n_; }           v.push_back(t);  
    void setThing(int n) { n_ = n; }                }  
private:  
    int n_;  
};
```

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.)

STL has to compare them using operator<. Add an appropriate operator< as either a member function in Thing, or as a free-standing function that compares two Thing& parameters.

Hex View Exercise

- Split up into break out rooms.
- Take a look at

<https://courses.cs.washington.edu/courses/cse333/20sp/sections/sec06.idx>

- Log into attu, use wget to download the file, then look into it.
- Try to figure out:
 - How many documents are in this index?
 - Which words are in each document?

Hex View Exercise

- Split up into break out rooms.
- Take a look at

<https://courses.cs.washington.edu/courses/cse333/20sp/sections/sec06.idx>

- Log into attu, use wget to download the file, then look into it.
- Try to figure out:
 - How many documents are in this index?
 - Which words are in each document?
- **Answer: This index file was built off of test_tree/tiny**