CSE 333 Section 7

HW3, C++, and Inheritance

Ever have a moment like this when programming?

UNIVERSITY of WASHINGTON

Logistics

- **Exercise 14** due tomorrow (!!)

  Friday (5/16) @ 10 am

- **HW3** due in over a week **Thursday**

  (5/22) @ 11:59 PM

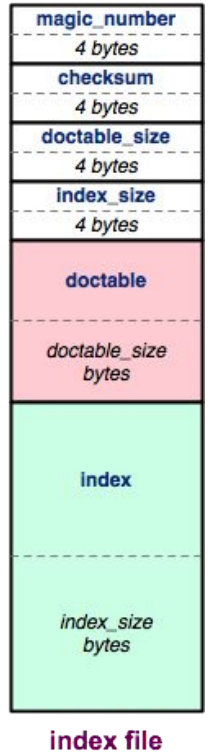
- How was the midterm?
  - Reminder: with the dropped question, max score was only 88, so interpret scores accordingly

HW 3 Overview

Index File

Crawling the whole file tree takes a long time!

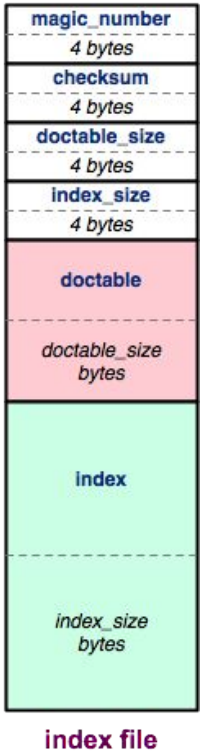
To save time we'll write the completed `DocTable` and `MemIndex` into a file!



Byte Ordering and Endianness

- Network (Disk) Byte Order (Big Endian)
  - The most significant byte is stored in the lowest 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 netlong);  // network to host`
- Pro-tip:
  The structs in HW3 have `toDiskFormat()` and `toHostFormat()` functions that will convert endianness for you.

Index File Components

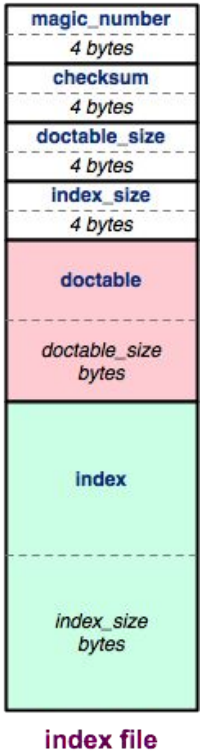


Header (metadata)

DocTable

MemIndex

Index File Header



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

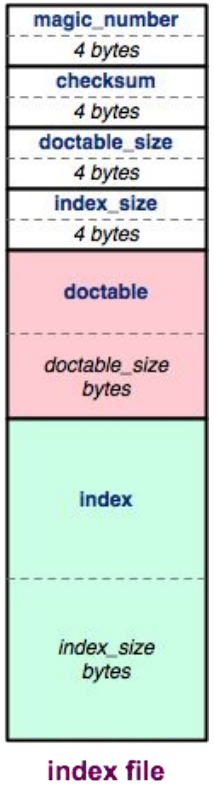
Index File Header - HEX

1. Find a hex editor/viewer of your choice
   - `xxd <indexfile>`
   - `hexdump -vC <indexfile>`
   - Pipe the output into a file or into `less` to view

```
0000000: cafe f00d 1c42 4620 0000 205b 0000 075d  .....BF .. [...]
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....
```

The header:

Magic word Checksum Doctable size Index size



Hex View

- emacs – “M-x hexl-mode”

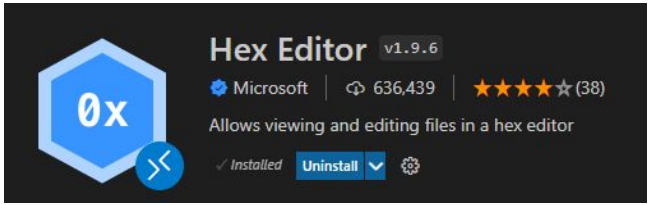
```
File Edit Options Buffers Tools Hexl Help
87654321  0011 2233 4455 6677 8899 aabb ccdd eeff  0123456789abcdef
00000000: cafe f00d ff48 a0a1 0000 006a 0000 024e  .....H.....j...N
00000010: 0000 0001 0000 0002 0000 001c 0000 0024  ...............$
00000020: 0000 0054 0000 0000 0000 0002 0026 2e2f  ...T.........&./
00000030: 7465 7374 5f74 7265 652f 7469 6e79 2f68  test_tree/tiny/h
00000040: 6f6d 652d 6f6e 2d74 6865 2d72 616e 6765  ome-on-the-range
00000050: 2e74 7874 0000 0000 0000 0001 001c 2e2f  .txt...........
```

- vim – “:%!xxd”

```
00000000: cafe f00d ff48 a0a1 0000 006a 0000 024e  .....H.....j...N
00000010: 0000 0001 0000 0002 0000 001c 0000 0024  ...............$
00000020: 0000 0054 0000 0000 0000 0002 0026 2e2f  ...T.........&./
00000030: 7465 7374 5f74 7265 652f 7469 6e79 2f68  test_tree/tiny/h
00000040: 6f6d 652d 6f6e 2d74 6865 2d72 616e 6765  ome-on-the-range
00000050: 2e74 7874 0000 0000 0000 0001 001c 2e2f  .txt...........
```

Hex View

For those working in VSCode…



- 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 ff48 a0a1 0000 006a 0000 024e  .....H.....j...N
00000010: 0000 0001 0000 0002 0000 001c 0000 0024  ...............$
00000020: 0000 0054 0000 0000 0000 0002 0026 2e2f  ...T.........&./
00000030: 7465 7374 5f74 7265 652f 7469 6e79 2f68  test_tree/tiny/h
00000040: 6f6d 652d 6f6e 2d74 6865 2d72 616e 6765  ome-on-the-range
00000050: 2e74 7874 0000 0000 0000 0001 001c 2e2f  .txt.........../
```

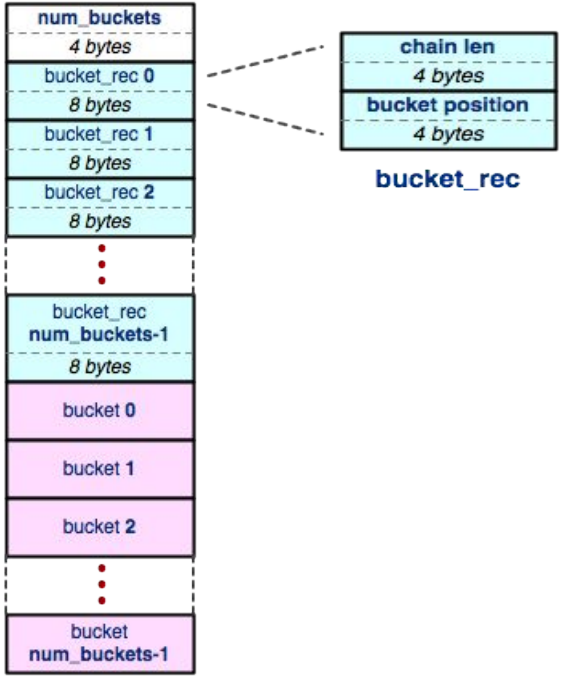
- vim – “:%!xxd”

```
00000000: cafe f00d ff48 a0a1 0000 006a 0000 024e  .....H.....j...N
00000010: 0000 0001 0000 0002 0000 001c 0000 0024  ...............$
00000020: 0000 0054 0000 0000 0000 0002 0026 2e2f  ...T.........&./
00000030: 7465 7374 5f74 7265 652f 7469 6e79 2f68  test_tree/tiny/h
00000040: 6f6d 652d 6f6e 2d74 6865 2d72 616e 6765  ome-on-the-range
00000050: 2e74 7874 0000 0000 0000 0001 001c 2e2f  .txt.........../
```

HashTable

- HashTable can have varying amount of buckets, so start with `num_buckets`.

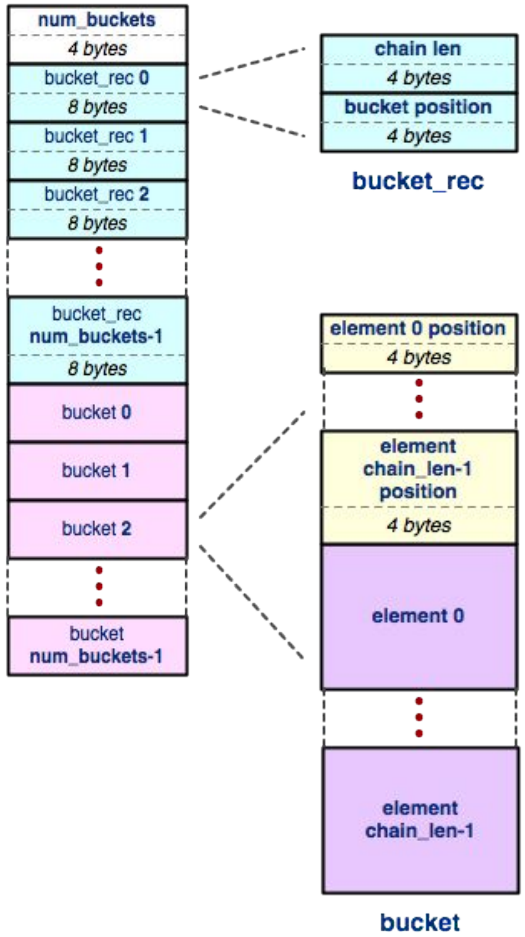
- Buckets can be of varying lengths. To know the offset, we store some bucket records.



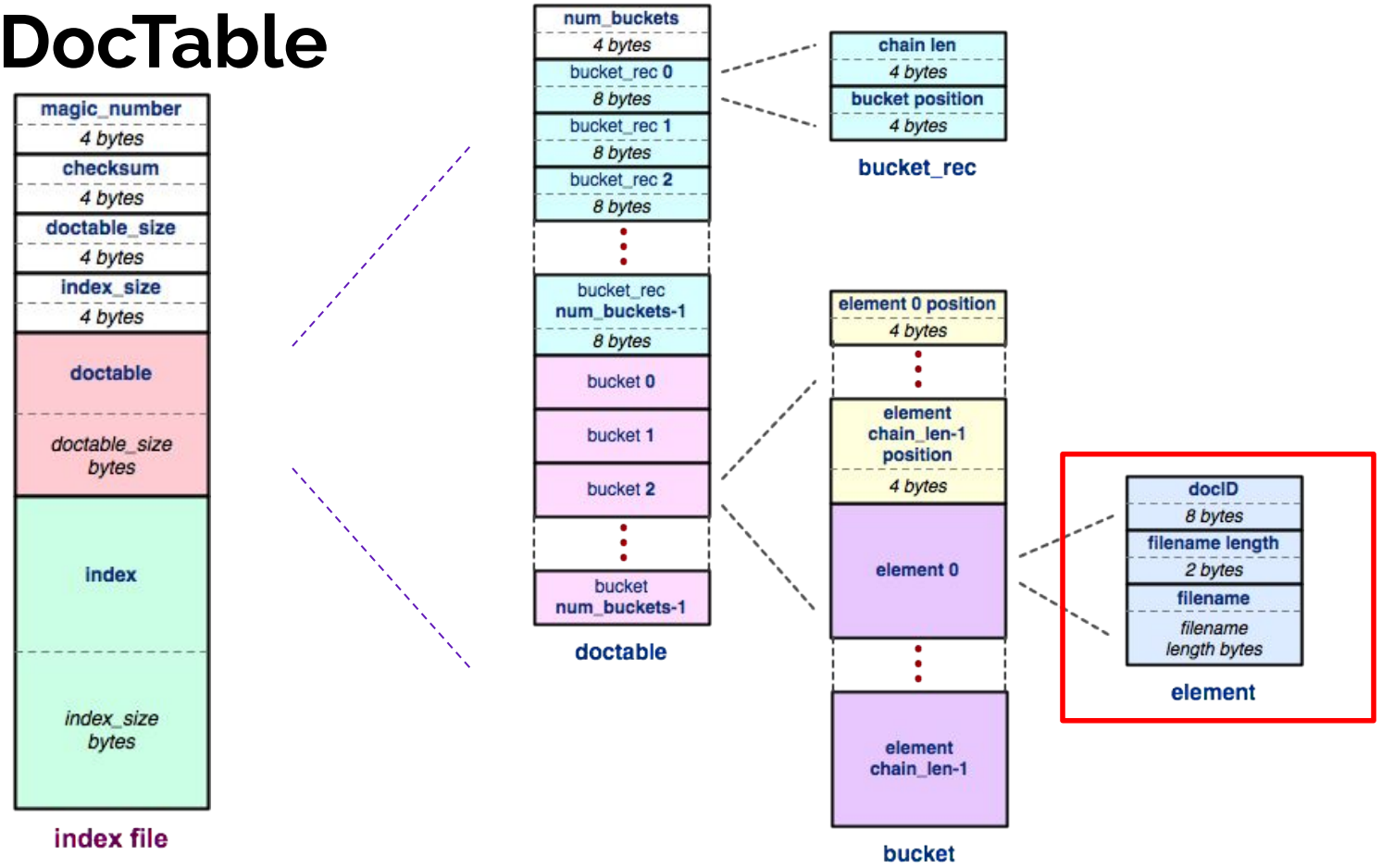
Buckets

- A bucket is a list that contains elements in the table. Offset to a bucket is found in a bucket record.

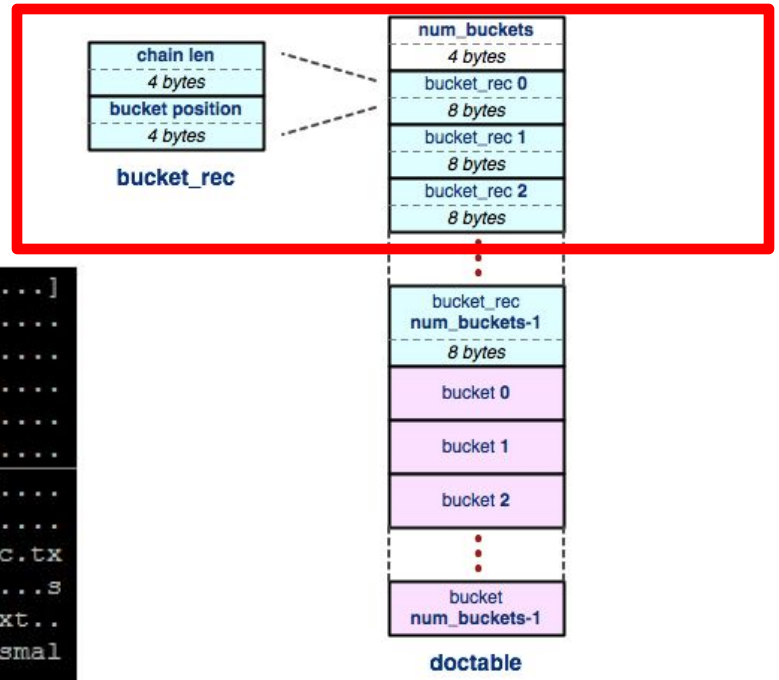
- Elements can be of various sizes, so we need to store element positions to know where each element is.



DocTable


magic_number
4 bytes
checksum
4 bytes
doctable_size
4 bytes
index_size
4 bytes
doctable
doctable_size bytes
index
index_size bytes
index file
num_buckets
4 bytes
bucket_rec 0
8 bytes
bucket_rec 1
8 bytes
bucket_rec 2
8 bytes
bucket_rec num_buckets-1
8 bytes
bucket 0
bucket 1
bucket 2
bucket num_buckets-1
doctable
chain len
4 bytes
bucket position
4 bytes
bucket_rec
element 0 position
4 bytes
element chain_len-1 position
4 bytes
element 0
element chain_len-1
bucket
docID
8 bytes
filename length
2 bytes
filename
filename length bytes
element

DocTable (Hex)



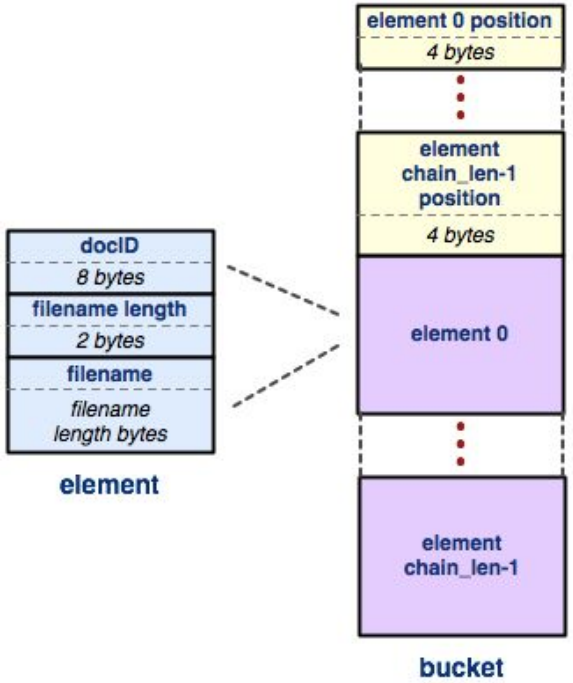

```
0000000: cafe f00d 1c42 4620 0000 205b 0000 075d  .....BF .. [...]
0000010: 0000 0400 0000 0000 0000 2014 0000 0001  .......... .....
0000020: 0000 2014 0000 0001 0000 2031 0000 0001  .. ....... 1....
0000030: 0000 204e 0000 0000 0000 206b 0000 0000  .. N...... k....
0000040: 0000 206b 0000 0000 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

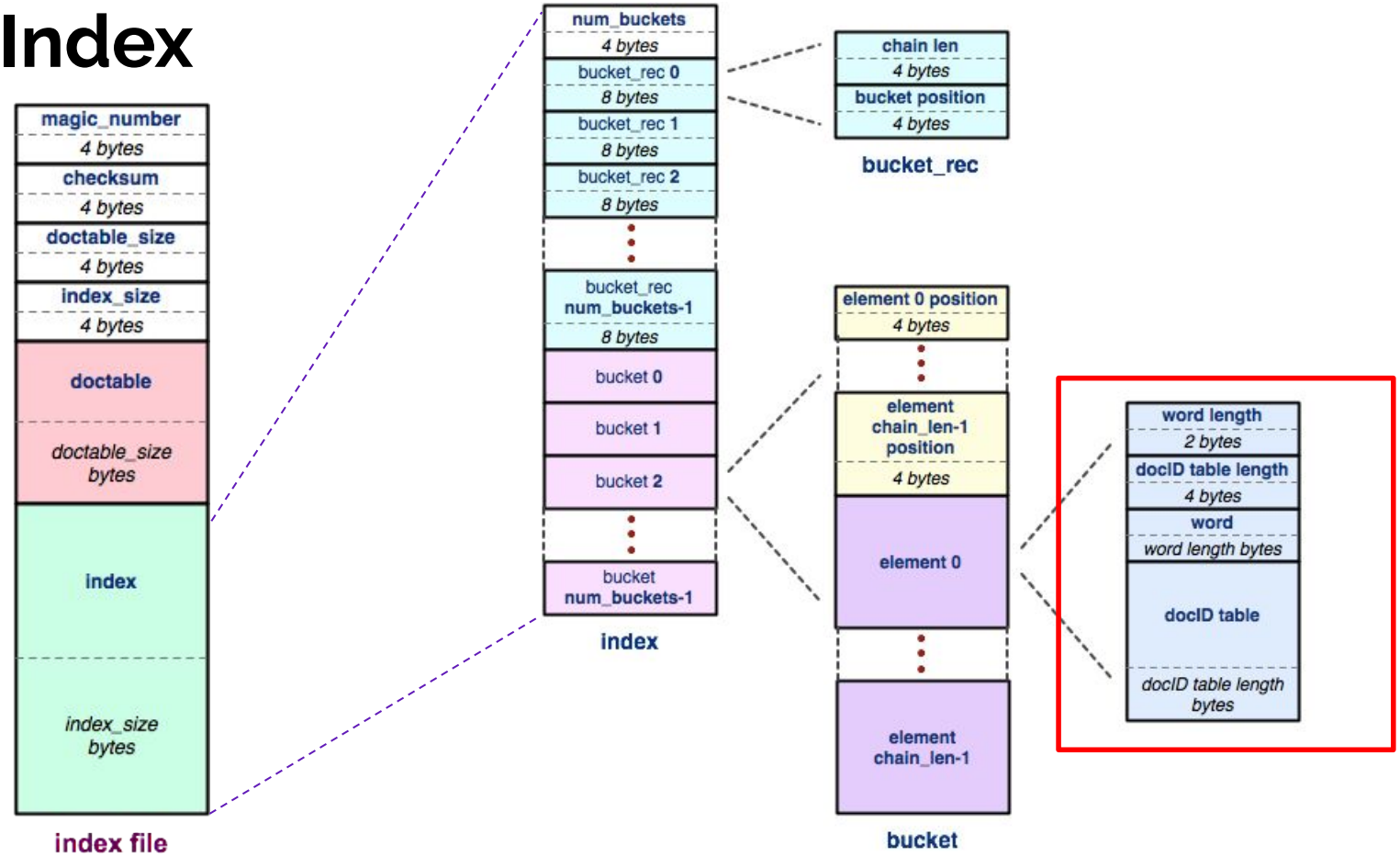
```
0002000: 0000 206b 0000 0000 0000 206b 0000 0000  .. k...... k....
0002010: 0000 206b 0000 2018 0000 0000 0000 0001  .. k.. ........
0002020: 000f 736d 616c 6c5f 6469 722f 632e 7478  ..small_dir/c.tx
0002030: 7400 0020 3500 0000 0000 0000 0200 0f73  t.. 5..........s
0002040: 6d61 6c6c 5f64 6972 2f62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



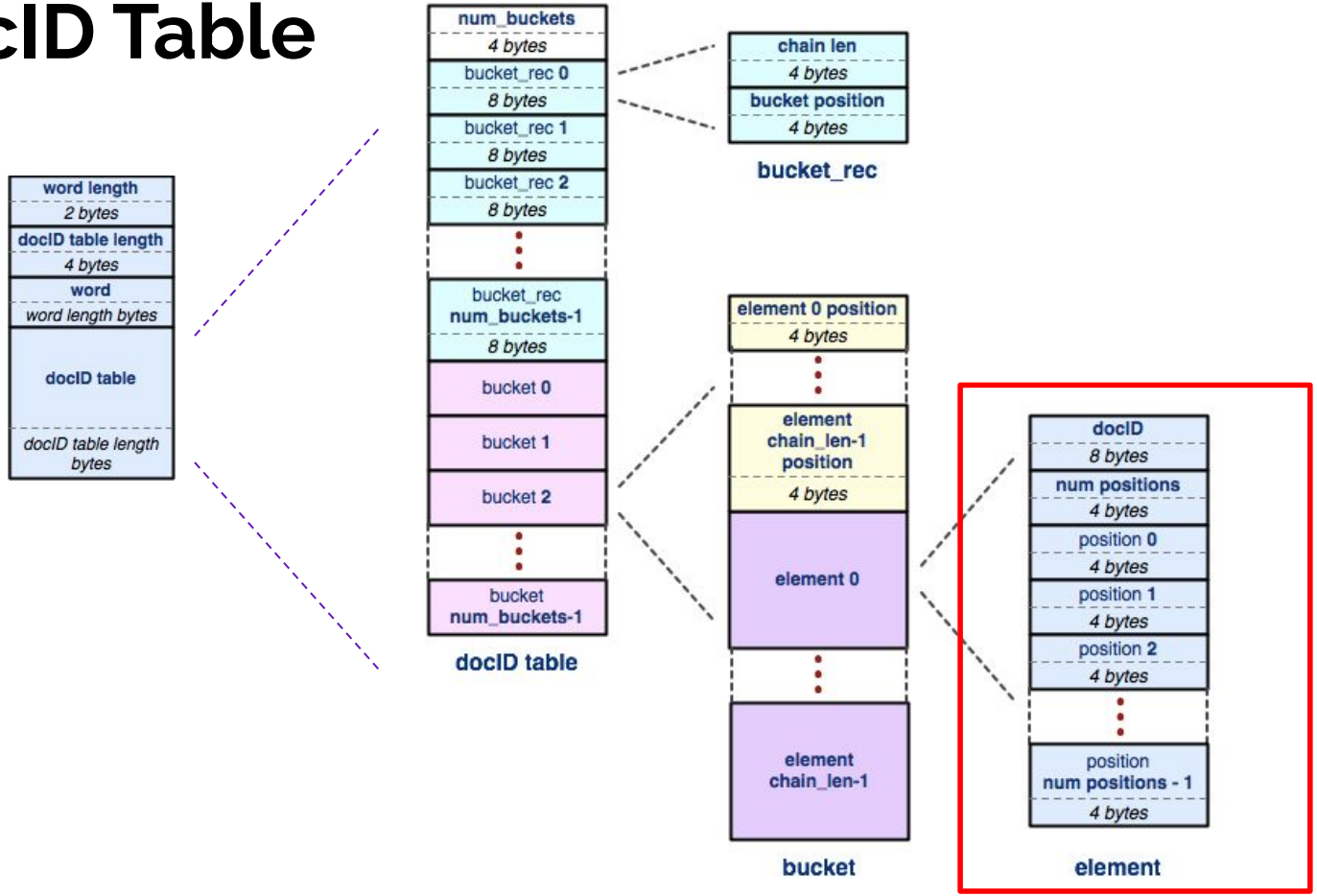
The buckets: where n is equal to the number of elements

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

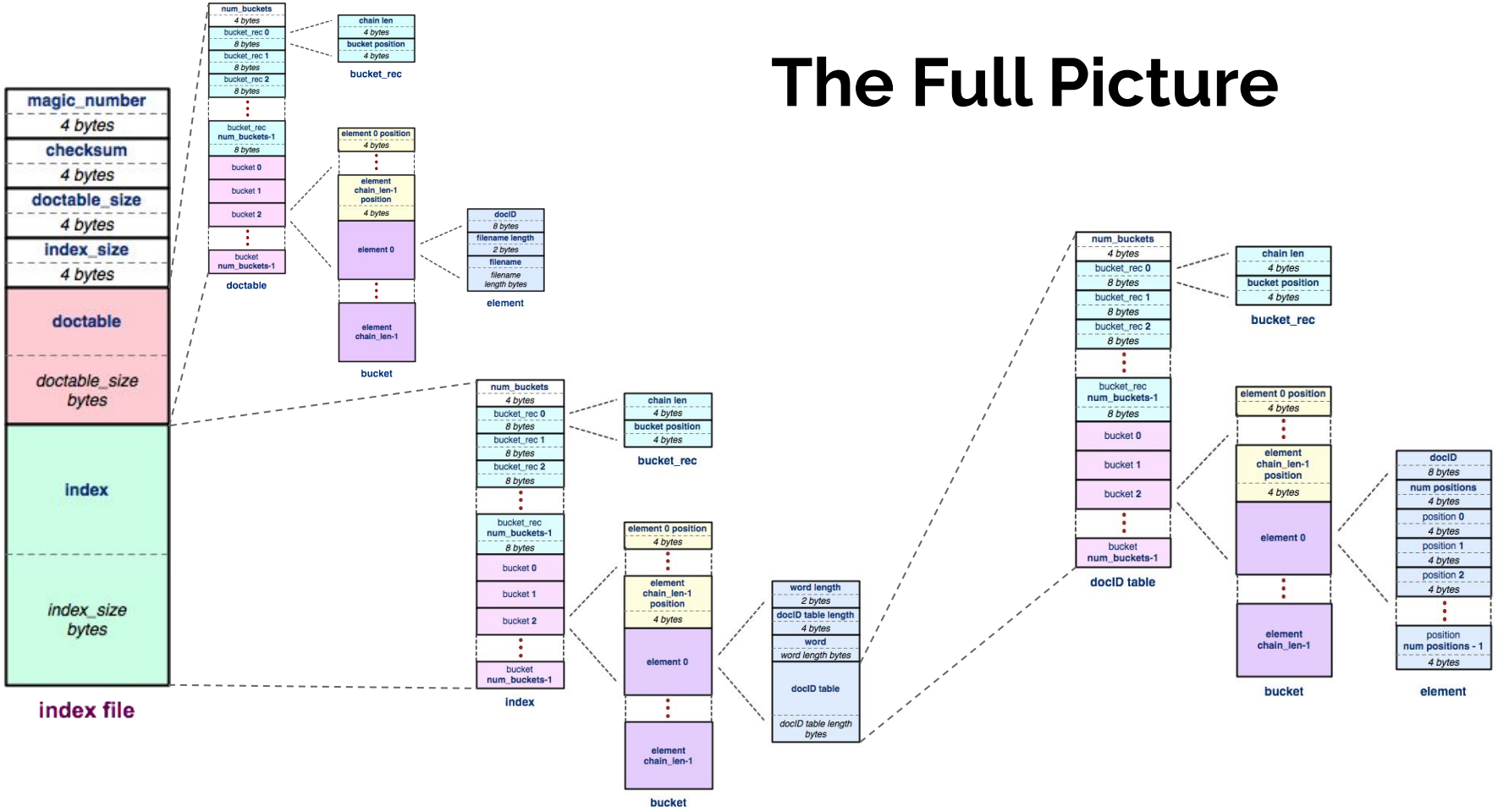
Index


magic_number
4 bytes
checksum
4 bytes
doctable_size
4 bytes
index_size
4 bytes
doctable
doctable_size bytes
index
index_size bytes
index file
num_buckets
4 bytes
bucket_rec 0
8 bytes
bucket_rec 1
8 bytes
bucket_rec 2
8 bytes
bucket_rec num_buckets-1
8 bytes
bucket 0
bucket 1
bucket 2
bucket num_buckets-1
index
chain len
4 bytes
bucket position
4 bytes
bucket_rec
element 0 position
4 bytes
element chain_len-1 position
4 bytes
element 0
element chain_len-1
bucket
word length
2 bytes
docID table length
4 bytes
word
word length bytes
docID table
docID table length bytes

DocID Table


word length
2 bytes
docID table length
4 bytes
word
word length bytes
docID table
docID table length bytes
num_buckets
4 bytes
bucket_rec 0
8 bytes
bucket_rec 1
8 bytes
bucket_rec 2
8 bytes
bucket_rec num_buckets-1
8 bytes
bucket 0
bucket 1
bucket 2
bucket num_buckets-1
docID table
chain len
4 bytes
bucket position
4 bytes
bucket_rec
element 0 position
4 bytes
element chain_len-1 position
4 bytes
element 0
element chain_len-1
bucket
docID
8 bytes
num positions
4 bytes
position 0
4 bytes
position 1
4 bytes
position 2
4 bytes
position num positions - 1
4 bytes
element

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()`.

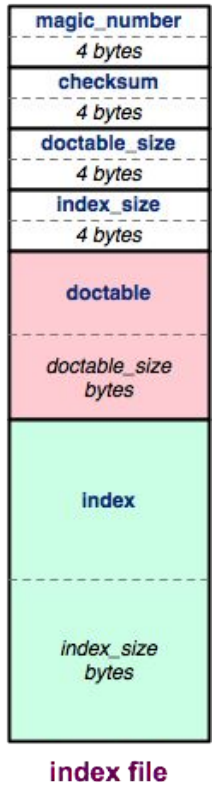
HW Tips: Index Checker (hw3fsck)

- Hw3fsck checks fields inside the file for reasonableness. Prints out a helpful message if it spots some kind of problem.

- More rigorous check on your index file you've produced
  - Run `./hw3fsck index_filename`

- Run after finishing WriteIndex.cc

- Can be found in hw3/hw3fsck directory (and compiled version in solution_binaries also)



Hex View Exercise

- Take a look at https://courses.cs.washington.edu/courses/cse333/25sp/sections/sec07.idx
  - Download the file, then look into it using your viewer of choice.
- Try to figure out:
  - How many documents are in this index?
  - Which words are in each document?

Hex View Exercise

- Take a look at https://courses.cs.washington.edu/courses/cse333/25sp/sections/sec07.idx
  - Download the file, then look into it using your viewer of choice.
- 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 so 2 documents, and 9 words.

Smart Pointers!

Review: Smart Pointers

- **`std::shared_ptr`** (Documentation) – Uses reference counting to determine when to delete a managed raw pointer
  - **`std::weak_ptr`** (Documentation) – Used in conjunction with `shared_ptr` but does **not** contribute to reference count

- **`std::unique_ptr`** (Documentation) – Uniquely manages a raw pointer
  - Used when you want to declare unique ownership of a pointer
  - Disabled `cctor` and `op=`

Using Smart Pointers

- Treat a smart pointer like a **normal (raw) pointer**, except now you **won't** have to use **`delete`** to deallocate memory!
  - You can use `*`, `->`, `[]` as you would with a raw pointer!

- **Initialize** a smart pointer by passing in a pointer to **heap memory**:

```
unique_ptr<int[]> u_ptr(new int[3]);
```

  - For **`shared_ptr`** and **`weak_ptr`**, you can use cctor and op= to get a copy

```
shared_ptr<int[]> s_ptr(another_shared_ptr);
```

Using Smart Pointers cont.

- Want to transfer ownership from one `unique_ptr` to another ?

```
unique_ptr<T> V = std::move(unique_ptr<T> U);
```

- Want to convert your `weak_ptr` to a `shared_ptr`?

```
std::shared_ptr s = w.lock();
```

- Want to get the reference count of a `shared_ptr`?

```
int count = s.use_count();
```

Casting

Motivations

- C++ casting provides **explicit casting** when converting from one type to another
  - Creates **less ambiguity** of an operation
  - Allows for more **visibility** for casting (even though they are longer to write)
  - See: https://google.github.io/styleguide/cppguide.html#Casting

Different Flavors of Casting

- `static_cast<type_to>(expression);`
  Casting between related types
- `dynamic_cast<type_to>(expression);`
  Casting pointers of similar types (only used with inheritance)
- `const_cast<type_to>(expression);`
  Adding or removing **const**-ness of a type
- `reinterpret_cast<type_to>(expression);`
  Casting between incompatible types of the **same size** (doesn't do float conversion)

Tips with Casting

- Style: Use C++ style casting in C++
  - Tradeoff: A little extra programming overhead and typing, but provides **clarity** to your programs
  - Be **explicit as possible** with your casting! This means if you notice multiple operations in an implicit cast, you should explicitly write out each cast!

- Read documentation of casting on which casting to use
  - Documentation: https://www.cplusplus.com/articles/iG3hAqkS/
  - The purpose of C++ casting is to be less ambiguous with what the casts you're using are actually doing

Exercise 2

```
class Base {
 public:
  int x;
};
```

```
class Derived : public Base {
 public:
  int y;
}
```

```
int64_t x = 0x7ffffffe870;
char* str = ______________________________ (x);

void foo(Base* b) {
  Derived* d = _________________________ (b);
  // additional code omitted
}

Derived* d = new Derived;
Base* b = _________________________________ (d);

double x = 64.382;
int64_t y = ______________________________ (x);
```

```
class Base {
 public:
  int x;
};
```

```
class Derived : public Base {
 public:
  int y;
}
```

```
int64_t x = 0x7ffffffffe870;
char* str = _____________________________ (x);

void foo(Base* b) {
  Derived* d = ________________________ (b);
  // additional code omitted
}

Derived* d = new Derived;
Base* b = ____________________________ (d);

double x = 64.382;
int64_t y = ____________________________ (x);
```

reinterpret_cast<char*>

dynamic_cast<Derived*>

static_cast<Base*>

static_cast<int64_t>

Inheritance

Inheritance

- Motivation: Better modularize our code for similar classes!

- The public interface of a derived class inherits all **non-private** member variables and functions (**except** for `ctor`, `cctor`, `dtor`, `op=`) from its base class
  - *Similar to*: A subclass inherits from a superclass

- Aside: We will be only using **public, single** inheritance in CSE 333

HW3 Data Structures

- **DocTable**: A HashTable that maps document ID's to document names
- **Index**: A HashTable of words to DocID Tables (which will contain…)
- **DocIDTable**: (Like `WordPositions` struct) A HashTable that maps documents to offsets in a file (where a specific word is)

That's a lot of HashTables!

HW3 Data Structures: Lots of HashTables

Key	Value
5	"test_tree/README.TXT"
1	"test_tree/books/ulysses.txt"
4	"test_tree/bash-4.2/trap.c"
2	"test_tree/enron_email/2."
3	"test_tree/example.txt"

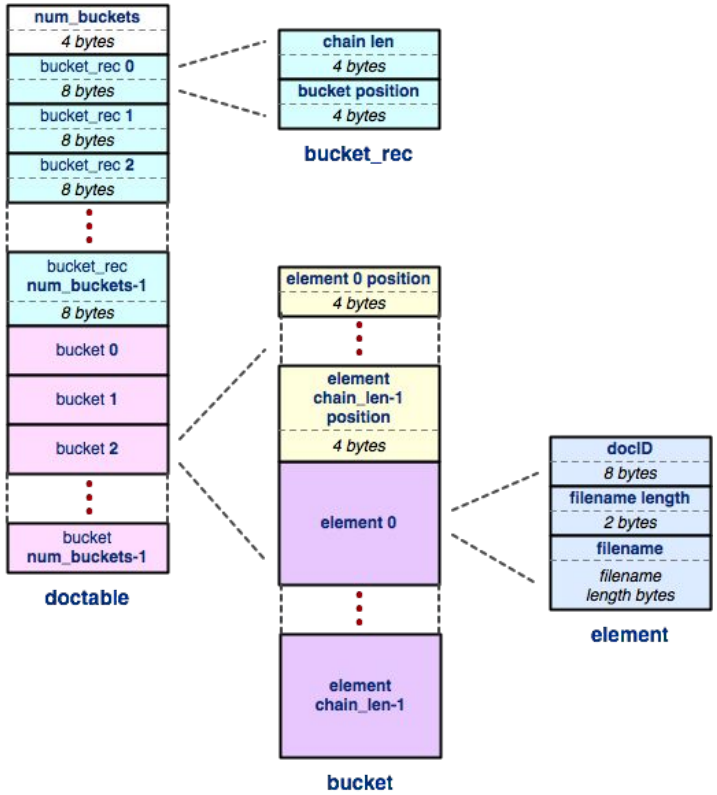
docid_to_docname

Key	Value
FNV64("test_tree/README.TXT")	(DocID_t) 5
FNV64("test_tree/example.txt")	(DocID_t) 3
FNV64("test_tree/enron_email/2.")	(DocID_t) 2
FNV64("test_tree/bash-4.2/trap.c")	(DocID_t) 4
FNV64("test_tree/books/ulysses.txt")	(DocID_t) 1

docname_to_docid

Write File (Part A)

Read File (Part B)



Motivation: HW3 Index Reading (Part B)

- Base Class:
  - `HashTableReader`
  - Example `protected` members:
    - `LookupElementPositions(HTKey_t hash_val) const;`
    - `FILE* file_;`
    - There are more! Take a look at the .h file for `HashTableReader`!
- Derived Classes (can use the fields of the base class):
  - `IndexTableReader` – Reads the index table
  - `DocIDTableReader` – Reads the DocID Table
  - `DocTableReader` – Reads the DocTable
  - `FileIndexReader` – Reads the Index from the file

Polymorphism: Dynamic Dispatch

- **Polymorphism** allows for you to access objects of related types (base and derived classes) – Allows interface usage instead of class implementation

- **Dynamic dispatch**: Implementation is determined *at* **runtime** via lookup
  - Allows you to call the **most-derived** version of the actual type of an object
  - Generally want to use this when you have a derived class

- `virtual` replaces the class's default **static dispatch** with **dynamic dispatch**

Dynamic Dispatch: Style Considerations

- Defining Dynamic Dispatch in your code base
  - Use `virtual` **only once** when first defined in the base class
    - (although in older code bases you may see it repeated on functions in subclasses)
  - All derived classes of a base class should use `override` to get the compiler to check that a function overrides a virtual function from a base class

- Use `virtual` for destructors of a base class – Guarantees all derived classes will use dynamic dispatch to ensure use of appropriate destructors

Example Base Class: Abstract Classes

- Pure `virtual` Functions – Functions without any implementation
  - Declaration Example: `virtual int foo() = 0;`
  - Used for creating an interface of a function

- **Abstract Classes** are those with one or more `virtual` functions
  - Creates an interface for the client to use without knowing its details
  - **Requires** a derived class to implement its functionality (cannot itself be instantiated)

- Often used like an interface!

  Usage Example: `AbstractClass* a = new DerivedClass(params);`

Example Abstract Class/Derived Class

```
using std::string;

class Fruit {
 public:
  Fruit() = default;
  virtual ~Fruit() {}

  // A fun fact
  virtual string FunFact() = 0;

};
```

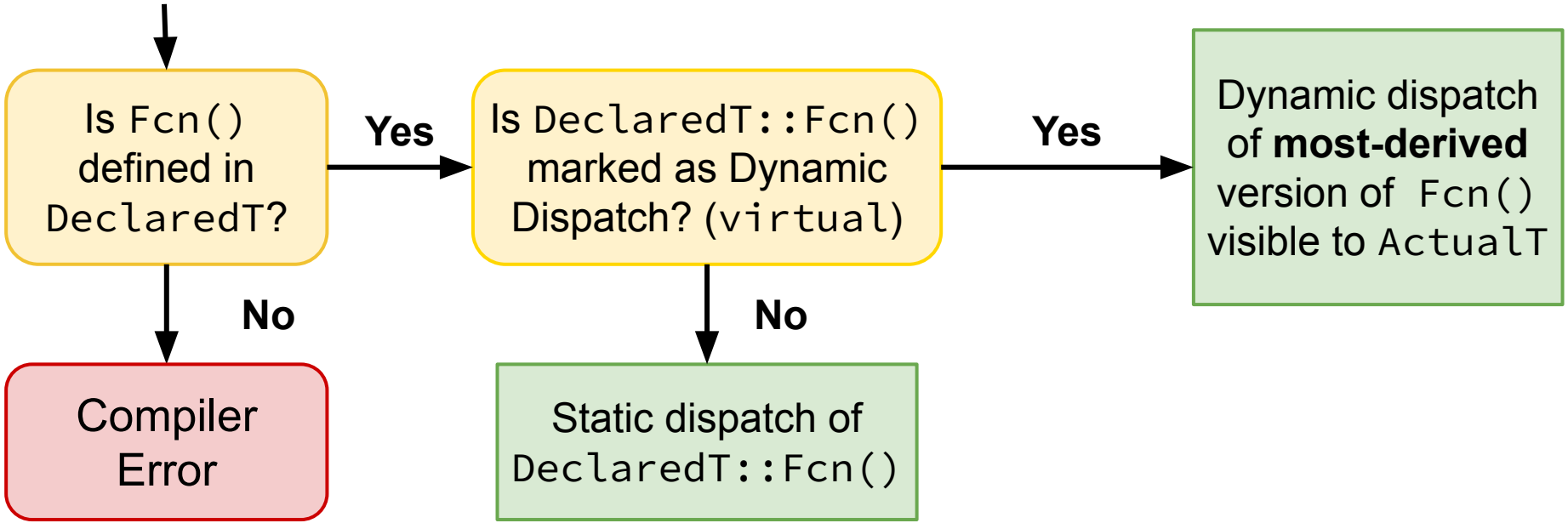
```
using std::string;

class Banana : public Fruit {
 public:
  Banana() = default;
  virtual ~Banana() = default;

  string FunFact() override {
    return “It’s a berry”;
  }
};
```

Dispatch Decision Tree

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



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();     //Dynamic Dispatch (for more derived)
  void Bar() override;    //Compiler Error!!
  void Baz();             //Dynamic Dispatch
};
```

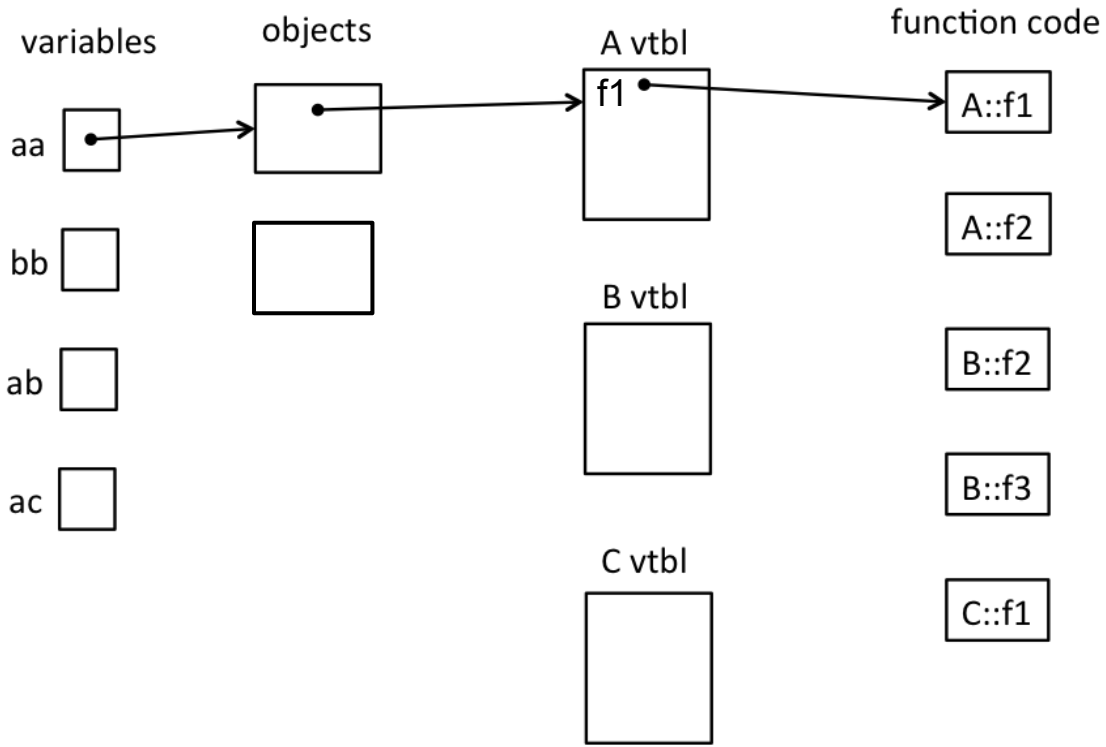
Fixes: static, dynamic, or error?

```
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();             // static dispatch
  void Baz() override;    // still dynamic (sticky!)
};
```

Exercise 1

Exercise 1 (Drawing vtable diagram)



Exercise 1 Solution (pointers)

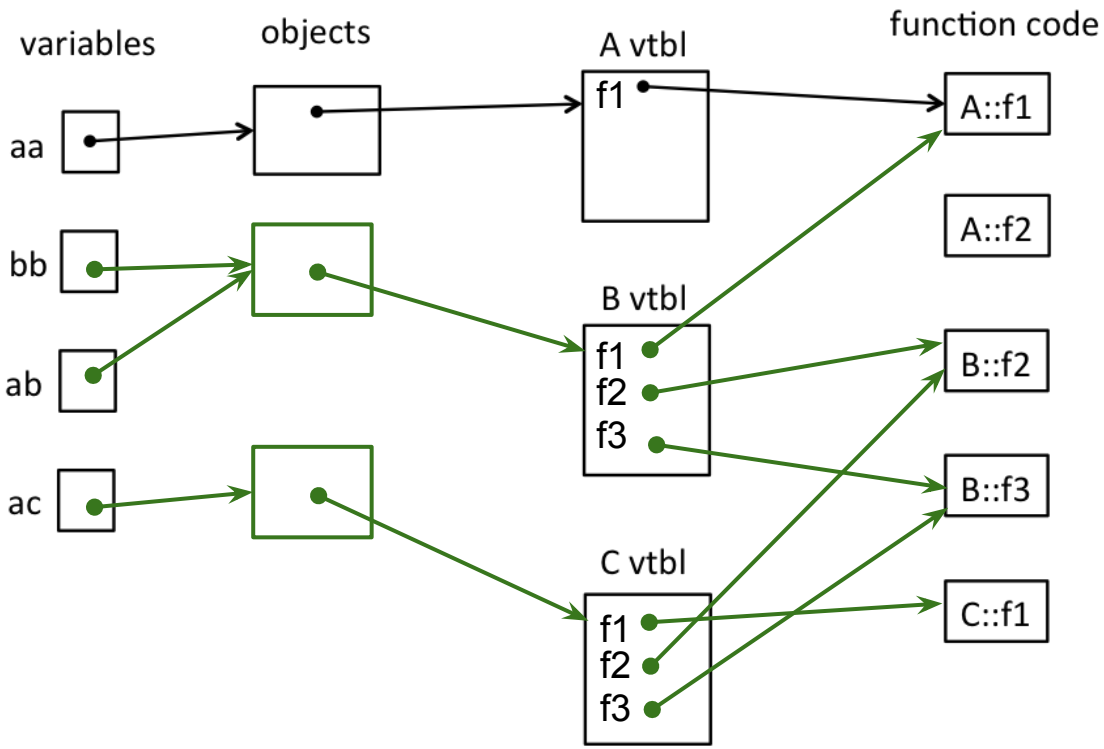
```
#include <iostream>
using namespace std;

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();
```



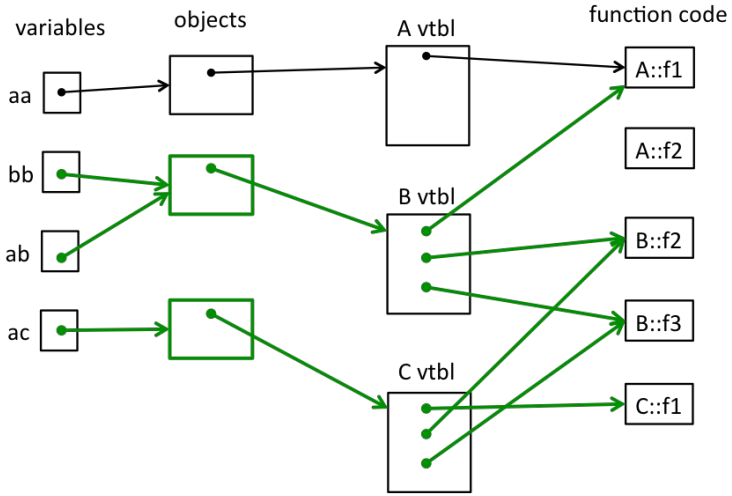
Exercise 1 Solution (output)

```
#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	B	C	D
B::f2 A::f1	A::f2 C::f1	A::f2 A::f1	B::f2 C::f1

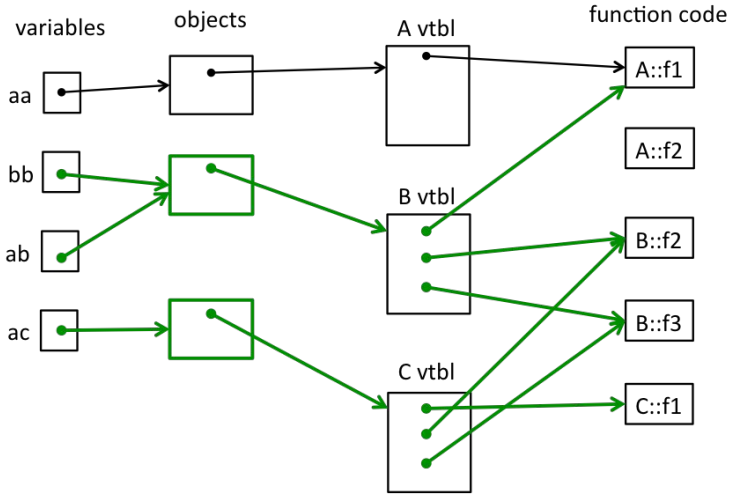
Exercise 1 Solution (output)

```
#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	B	C	D
B::f2 A::f1	A::f2 C::f1	A::f2 A::f1	B::f2 C::f1

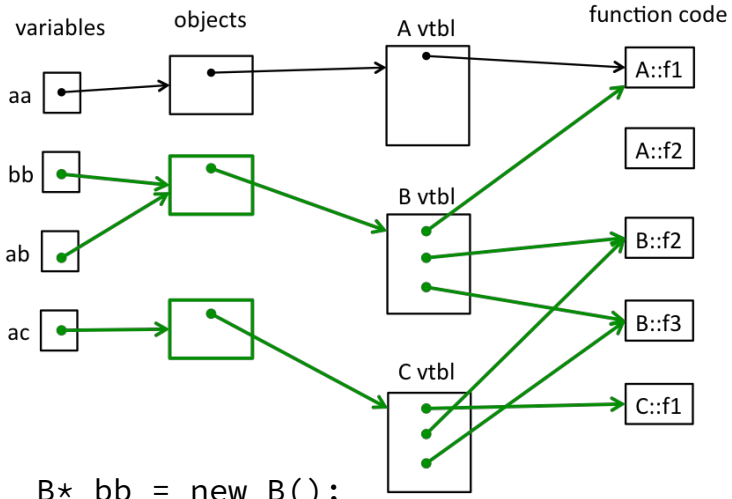
Exercise 1 Solution (output)

```
#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();
```

A	B	C	D
B::f2 ---- B::f2	A::f2 ---- B::f2	B::f2 ---- A::f2	A::f2 ---- A::f2

Exercise 1 Extension

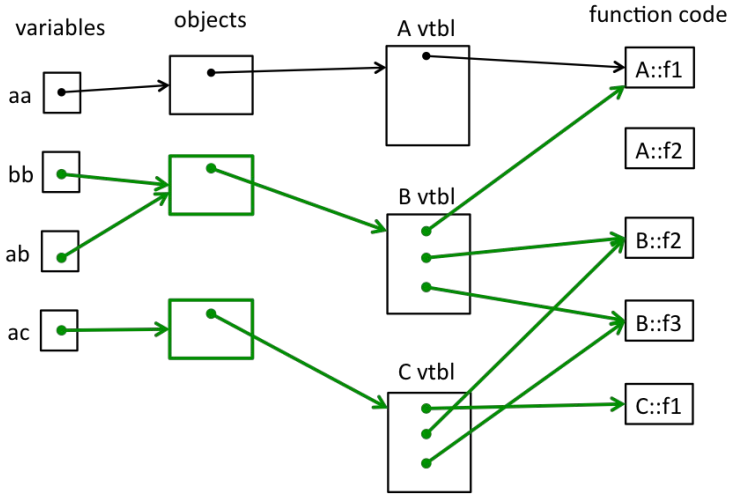
Exercise 2 Solution (output)

```
#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	B	C	D
B::f2 A::f1 B::f3	A::f2 A::f1 B::f3	A::f2 C::f1 B::f3	B::f2 C::f1 B::f3

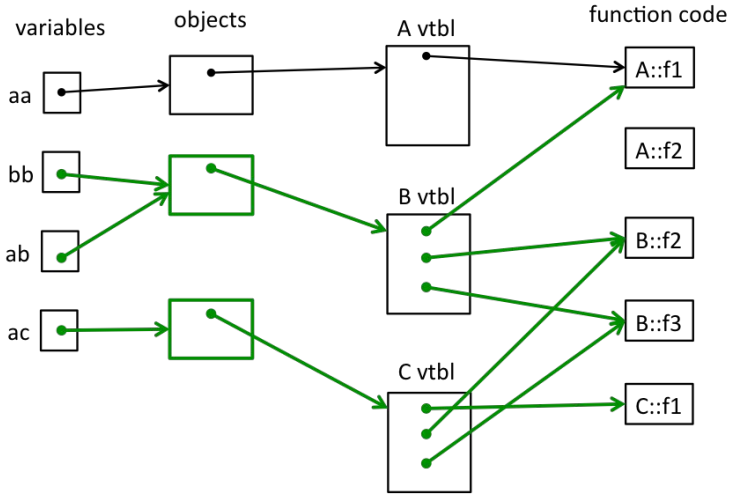
Exercise 2 Solution (output)

```
#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();
```

A	B	C	D
B::f2 A::f1	A::f2 C::f1	A::f2 A::f1	B::f2 C::f1

Bonus Exercise!

Bonus

Change the following code to use smart pointers.

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

struct IntNode {
  IntNode(int* val, IntNode* node): value(val), next(node) {}

  ~IntNode() { delete val; }

  int* value;
  IntNode* next;
};
```

Bonus

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

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;
};
```

Bonus

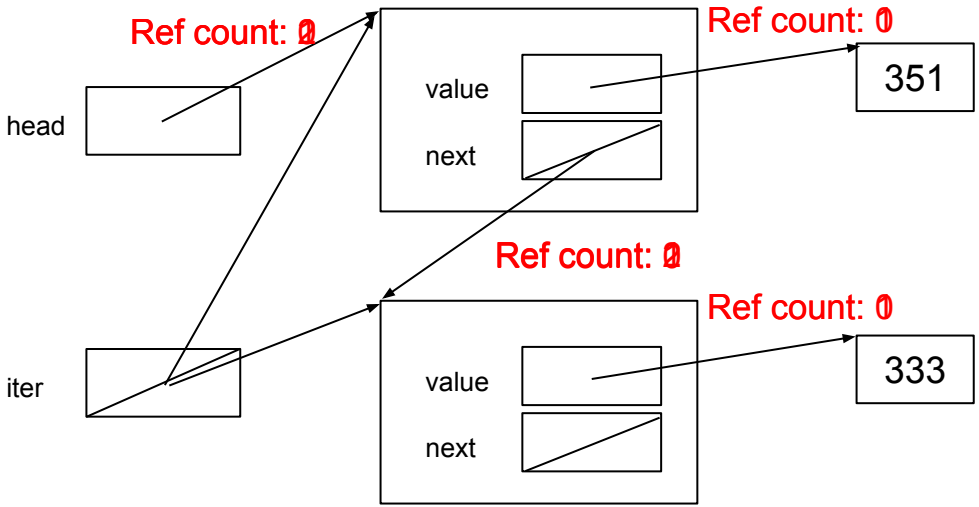
```
#include <memory>
using std::shared_ptr;

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;
};
```

Bonus : Client Code




```
#include <iostream>

using std::cout;
using std::endl;

int main() {
  shared_ptr<IntNode> head(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;
  }
}
```

Bonus: Client Code

Nothing left on the heap!

```
#include <iostream>

using std::cout;
using std::endl;

int main() {
  shared_ptr<IntNode> head(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;
  }
}
```

Thanks for coming to section!