CSE 333 Section 4

C++ Intro; Makefiles; HW2

w/ Julia & Kenzie!

Logistics

- **Exercise 5**
  - due **Friday** (January 28)
- **Exercise 6**
  - due **Wednesday** (February 2)
- **Homework 2**
  - due next **Thursday** (February 3)

Pointers, References, & Const

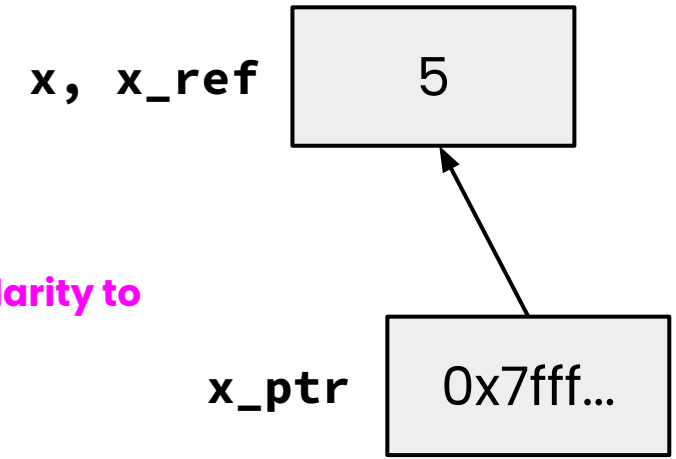
Example

Consider the following code:

```
int x = 5;
int& x_ref = x;
int* x_ptr = &x;
```

Note syntactic similarity to pointer declaration

Still the address-of operator!



What are some tradeoffs to using pointers vs references?

Pointers vs. References

Pointers

- Can move to different data via reassignment/pointer arithmetic
- Can be initialized to **`NULL`**
- Useful for output parameters: `MyClass* output`

References

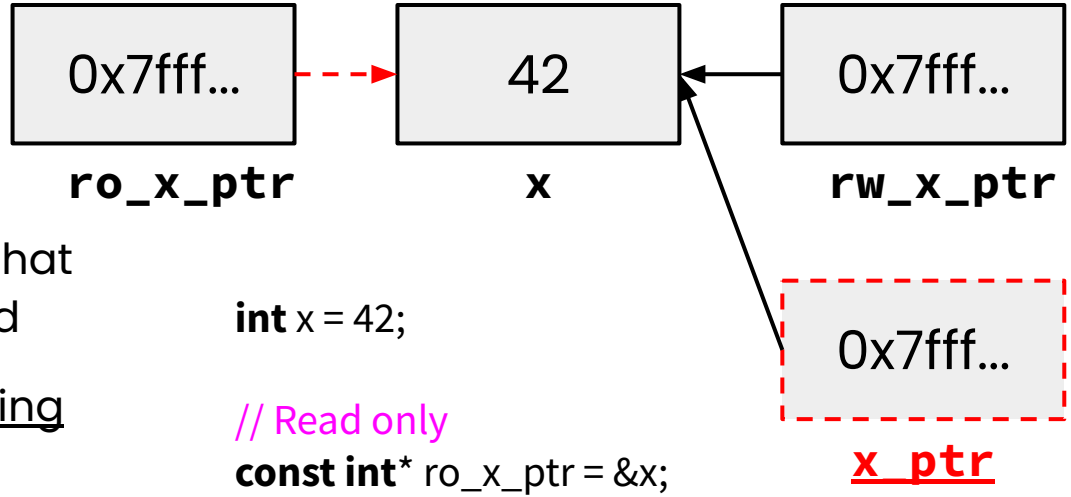
- References the same data for its entire lifetime - *can't reassign*
- No sensible "default reference," must be an alias
- Useful for input parameters: **`const`** `MyClass &input`

Pointers, References, Parameters

- **`void func(int& arg)`** vs. **`void func(int* arg)`**
- Use references when you don't want to deal with pointer semantics
  - Allows real pass-by-reference
  - Can make intentions clearer in some cases
- **STYLE TIP:** use references for input parameters and pointers for output parameters, with the output parameters declared last
  - Note: A reference can't be NULL

Const

- Mark a variable with const to make a compile time check that a variable is never reassigned
- Does not change the underlying write-permissions for this variable

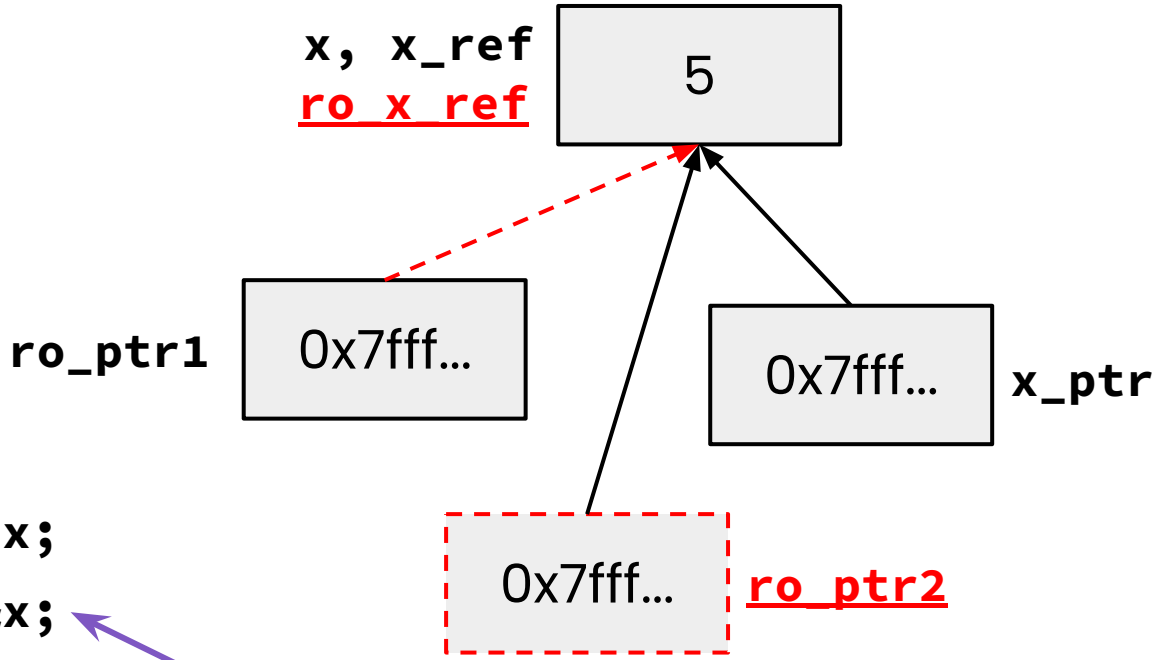



```
int x = 42;

// Read only
const int* ro_x_ptr = &x;

// Can still modify x with rw_ptr!
int* rw_x_ptr = &x;

// Only ever points to x
int* const x_ptr = &x;
```

Legend
Red = can't change box it's next to
Black = read and write

Exercise 1

```
int x = 5;
int& x_ref = x;
int* x_ptr = &x;
const int& ro_x_ref = x;
const int* ro_ptr1 = &x;
int* const ro_ptr2 = &x;
```



"Const pointer to an int"

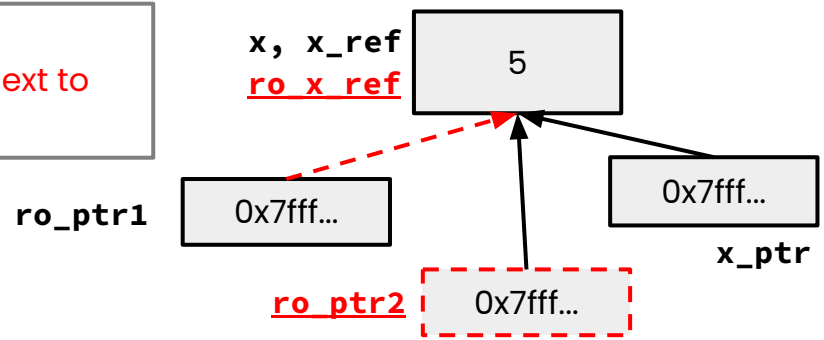
"Pointer to a const int"

Tip: Read the declaration "right-to-left"

Legend
Red = can't change box it's next to
Black = read and write

Exercise 1

Legend
Red = can't change box it's next to
Black = "read and write"




```
void foo(const int& arg);
void bar(int& arg);

int x = 5;
int& x_ref = x;
int* x_ptr = &x;
const int& ro_x_ref = x;
const int* ro_ptr1 = &x;
int* const ro_ptr2 = &x;
```

Which lines result in a compiler error?

✔ OK ✘ ERROR

✔ `bar(x_ref);`
✘ `bar(ro_x_ref);` ro_x_ref is const
✔ `foo(x_ref);`
✔ `ro_ptr1 = (int*) 0xDEADBEEF;`
✘ `x_ptr = &ro_x_ref;` ro_x_ref is const
✘ `ro_ptr2 = ro_ptr2 + 2;` ro_ptr2 is const
✘ `*ro_ptr1 = *ro_ptr1 + 1;` (*ro_ptr1) is const

Objects and Const Methods

```
#ifndef POINT_H_
#define POINT_H_

class Point {
 public:
  Point(const int x, const int y);
  int get_x() const { return x_; }
  int get_y() const { return y_; }
  double Distance(const Point& p) const;
  void SetLocation(const int& x, const int& y);

 private:
  int x_;
  int y_;
};  // class Point

#endif // POINT_H_
```

Cannot mutate the object it's called on.

Trying to change x_ or y_ inside will throw a compiler error!

A **const** class object can only call member functions that have been declared as **const**

Exercise 3

Exercise 3

Which *lines* of the snippets of code below would cause compiler errors?

✔️ OK ❌ ERROR

✔️ **int** z = 5;
✔️ **const int*** x = &z;
✔️ **int*** y = &z;
✔️ x = y;
❌ *x = *y;

✔️ **int** z = 5;
✔️ **int* const** w = &z;
✔️ **const int* const** v = &z;
❌ *v = *w;
✔️ *w = *v;

Exercise 3

Which *lines* of the snippets of code below would cause compiler errors?

✔ OK ❌ ERROR

```
class MultChoice {
 public:
  MultChoice(int q, char resp) : q_(q), resp_(resp) { }  // 2-arg ctor
  int get_q() const { return q_; }
  char get_resp() { return resp_; }
  bool Compare(MultChoice &mc) const;  // do these MultChoice's match?

 private:
  int  q_;     // question number
  char resp_;  // response: 'A','B','C','D', or 'E'
};  // class MultChoice
```

✔ **const MultChoice** m1(1,'A');
✔ **MultChoice** m2(2,'B');
❌ cout << m1.get_resp();
✔ cout << m2.get_q();

✔ **const MultChoice** m1(1,'A');
✔ **MultChoice** m2(2,'B');
✔ m1.Compare(m2);
❌ m2.Compare(m1);

What would you change about the class declaration to make it better?

```
class MultChoice {
 public:
  MultChoice(int q, char resp) : q_(q), resp_(resp) { }  // 2-arg ctor
  int get_q() const { return q_; }
  char get_resp() { return resp_; }
  bool Compare(MultChoice &mc) const;  // do these MultChoice's match?

 private:
  int  q_;     // question number
  char resp_;  // response: 'A','B','C','D', or 'E'
};  // class MultChoice
```

Makefiles

```
target: src1 src2 … srcN
    command/commands
```

Makefiles are used to manage project recompilation. Project structure / dependencies can be represented as a DAG, which a Makefile encodes to recursively build the minimum number of files for a target.

Exercise 5

Point.h

```
class Point { … };
```

UsePoint.cc

```
#include "Point.h"
#include "Thing.h"
int main( … ) { … }
```

UseThing.cc

```
#include "Thing.h"
int main( … ) { … }
```

Point.cc

```
#include "Point.h"
// defs of methods
```

Thing.h

```
struct Thing { … };
// full struct def here
```

Alone.cc

```
int main( … ) { … }
```

1. Draw out Point's DAG
    - The direction of the arrows is not important, but be consistent
2. Write a suitable Makefile for this structure

https://courses.cs.washington.edu/courses/cse333/22wi/lectures/08/08-make-22wi.pdf#page=21

DAG

Point.h

```
class Point { … };
```

UsePoint.cc

```
#include "Point.h"
#include "Thing.h"
int main( … ) { … }
```

UseThing.cc

```
#include "Thing.h"
int main( … ) { … }
```

Point.cc

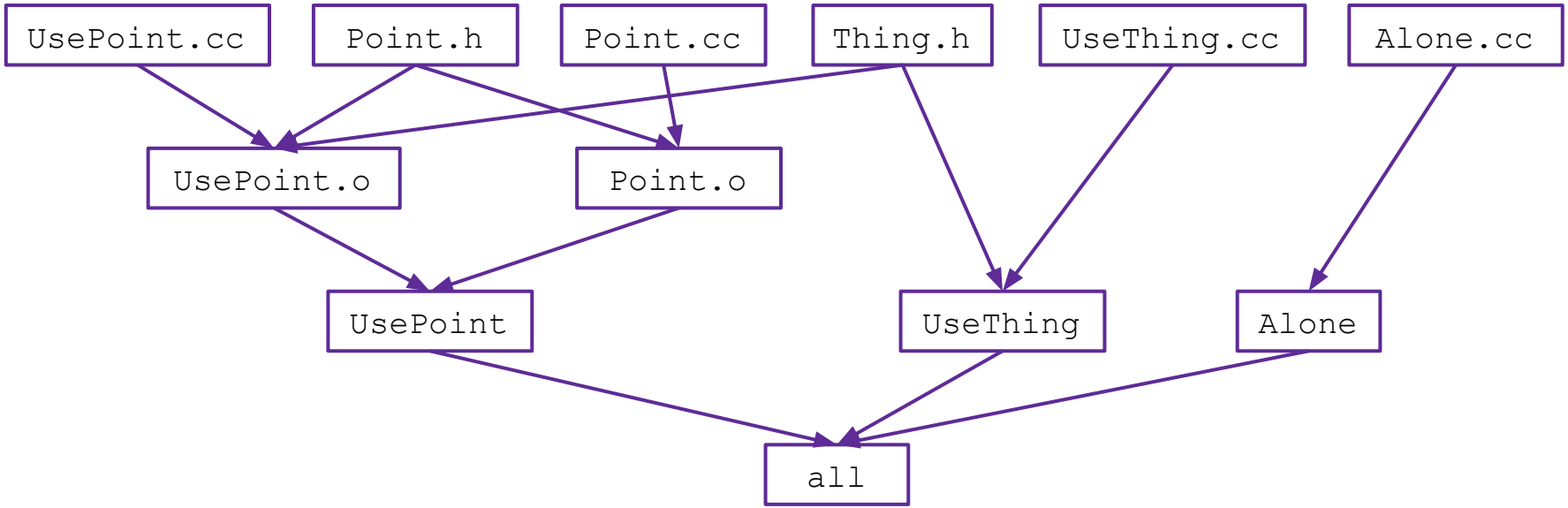
```
#include "Point.h"
// defs of methods
```

Thing.h

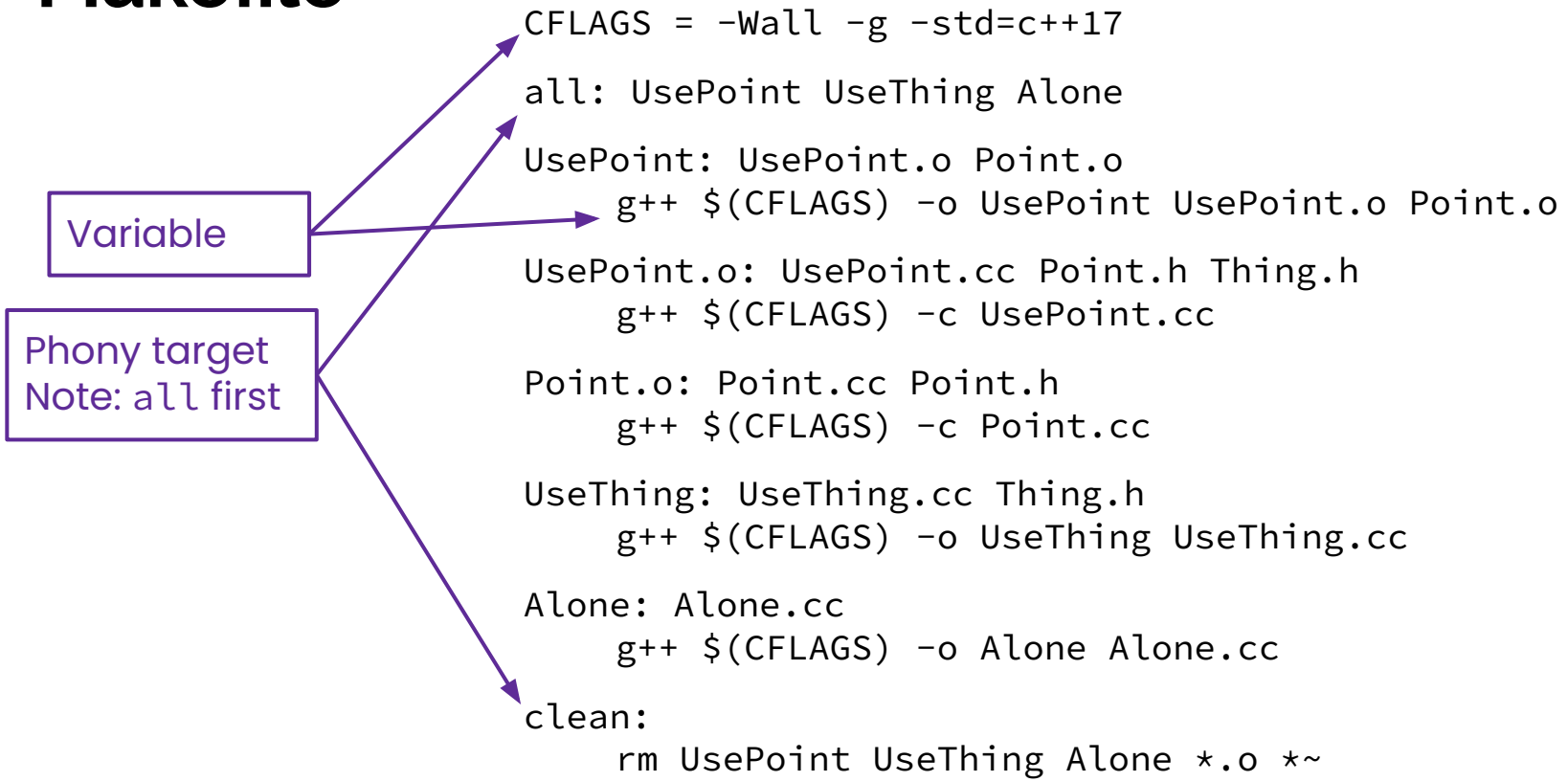
```
struct Thing { … };
// full struct def here
```

Alone.cc

```
int main( … ) { … }
```

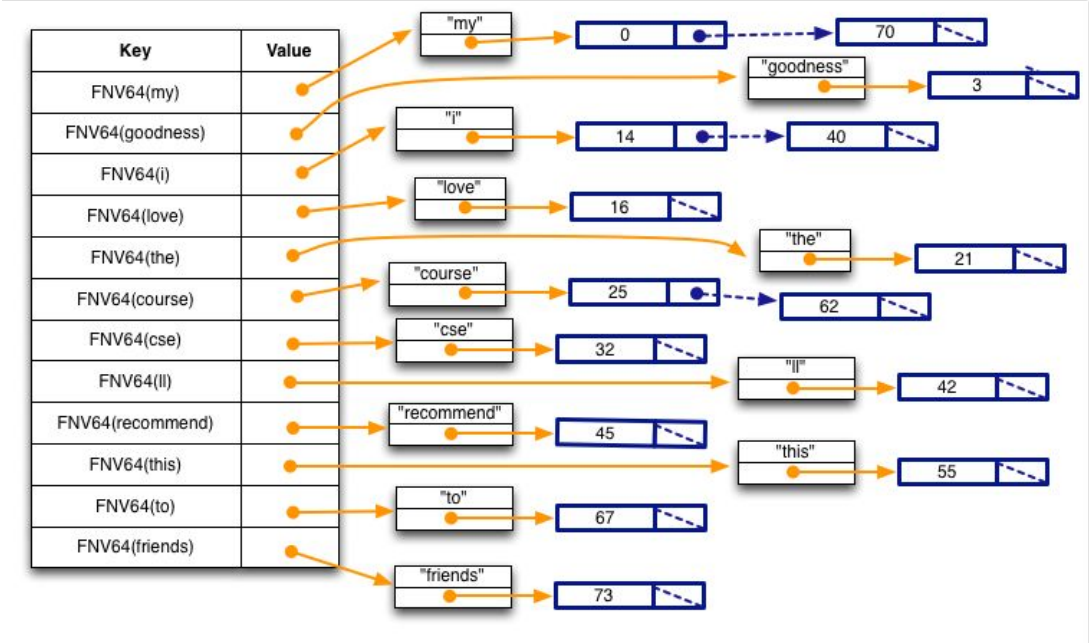


Makefile


Variable
Phony target
Note: all first
CFLAGS = -Wall -g -std=c++17
all: UsePoint UseThing Alone
UsePoint: UsePoint.o Point.o
g++ $(CFLAGS) -o UsePoint UsePoint.o Point.o
UsePoint.o: UsePoint.cc Point.h Thing.h
g++ $(CFLAGS) -c UsePoint.cc
Point.o: Point.cc Point.h
g++ $(CFLAGS) -c Point.cc
UseThing: UseThing.cc Thing.h
g++ $(CFLAGS) -o UseThing UseThing.cc
Alone: Alone.cc
g++ $(CFLAGS) -o Alone Alone.cc
clean:
rm UsePoint UseThing Alone *.o *~

Homework 2 Recap

Document Parsing



The parser produces these WordPosition structures for each file, which are stored in a hash table by word.

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

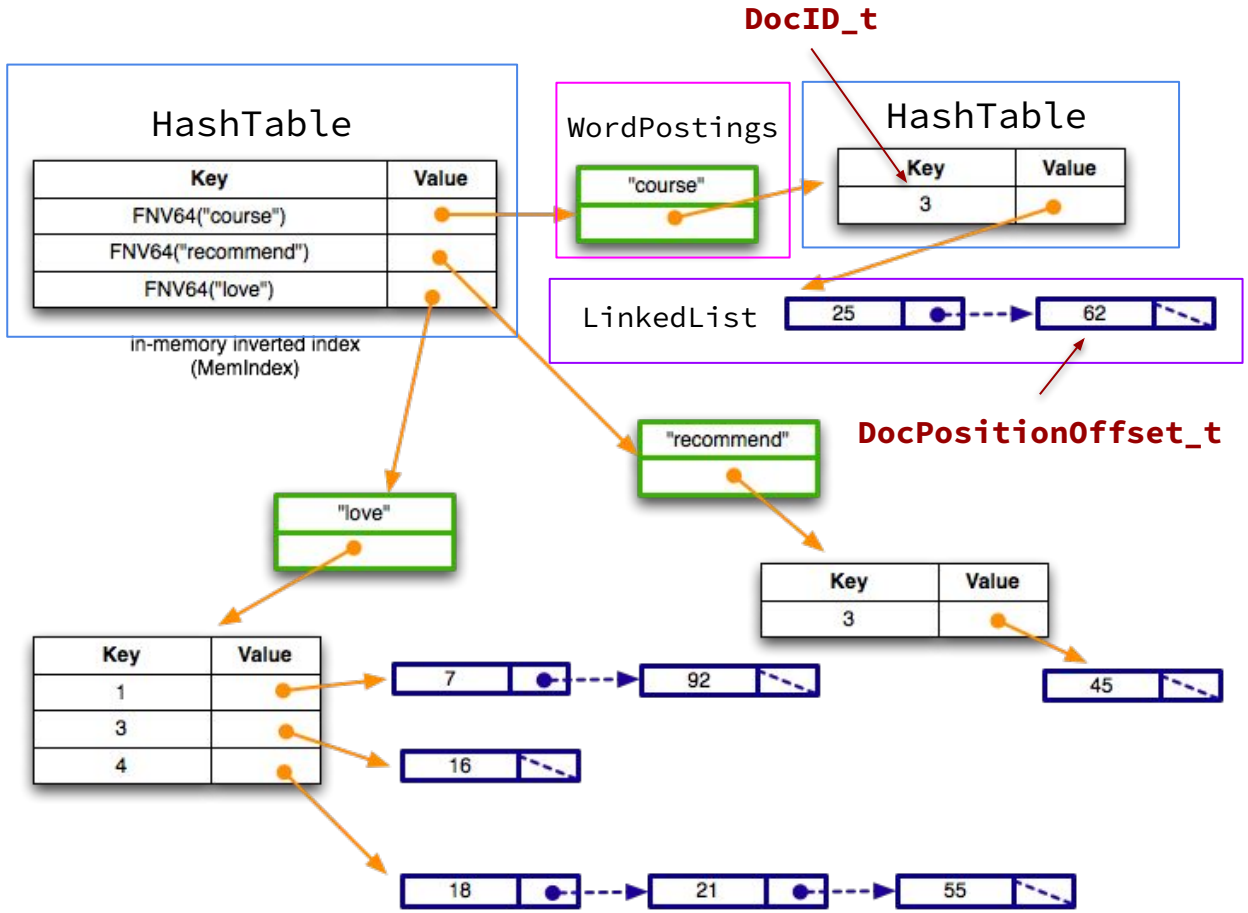
Map document names to IDs. FNV64 is a hash function.

MemIndex

The file crawler builds up a MemIndex structure, which is just a hash table of “word postings.”

A word posting is a pair of a word and a hash table with DocID keys and list of positions as a value.

Based on the figure, you can see that the word "course" appeared in a single document with docID 3, at byte offsets 25 and 62 from the start of file. Similarly, the word "love" appears in three documents: docID 1 at positions 7 and 92, docID 3 at position 16, and docID 4 at positions 18, 21, and 55.



Q&A :-)