# CSE 333
# Section 4

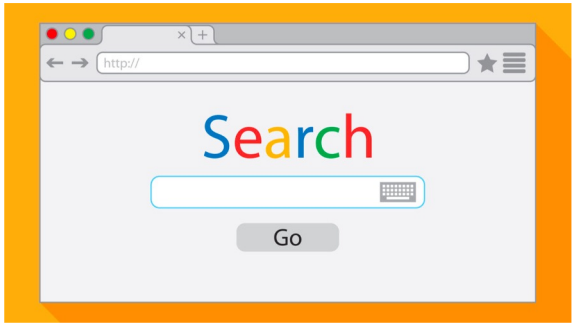
Makefiles, HW2 Overview, C++ Intro

# Logistics

- Homework 2
  - Due next **Thursday, 2/6 @ 11:59pm**
  - Indexing files to allow for searching
- Exercise 9
  - Write a Vector class in C++
  - Out tomorrow morning, due **Monday @ 10:00am**

# Homework 2 Overview

# Homework 2




- Main Idea: Build a search engine for a file system
  - It can **take in queries** and **output a list of files** in a directory that has that query
  - The query will be **ordered** based on the number of times the query is in that file
  - Should handle **multiple word queries** (*Note: all words in a query have to be in the file*)

- What does this mean?
  - Part A: **Parsing a file** and reading all of its contents into heap allocated memory
  - Part B: **Crawling a directory** (reading all regular files recursively in a directory) and building an index to query from
  - Part C: **Build a searchshell** (search engine) to query your index for results

**Note**: It will use the **LinkedList** and **HashTable** implementations from **HW1**!

# Part A: File Parsing

Read a file and generate a HashTable of WordPositions!

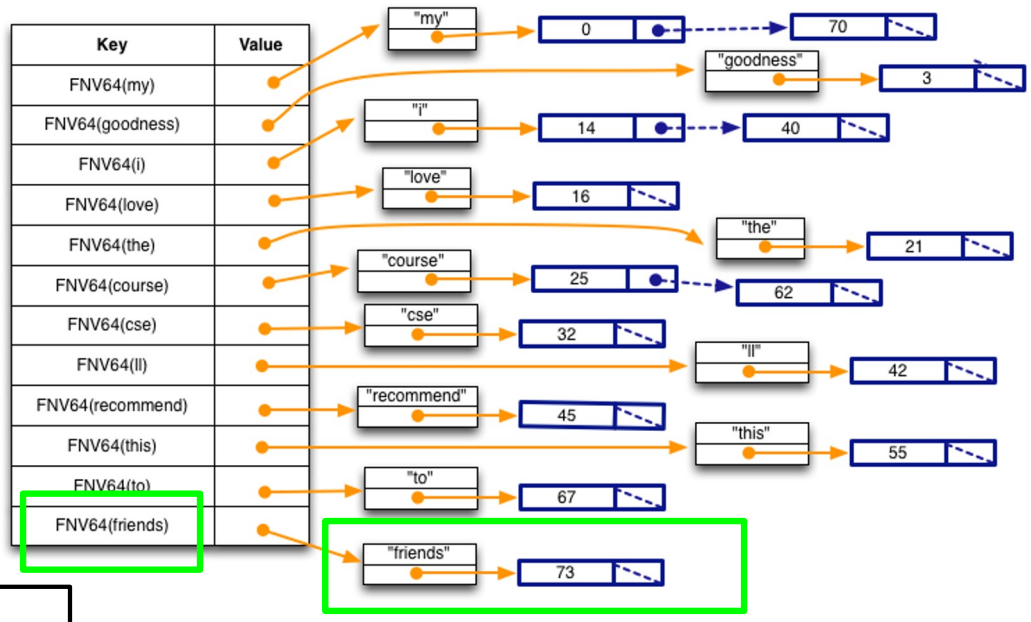
Word positions will include the word and LinkedList of its positions in a file.

```
typedef struct WordPositions {
 char        *word;      // normalized word.  Owned.
 LinkedList  *positions; // list of DocPositionOffset_t.
} WordPositions;
```

somefile.txt

```
My goodness!  I love the course CSE333.\n
I'll recommend this course to my friends.\n
```

ParseIntoWordPositionsTable(contents)




Note that the key is the hashed C-string of `WordPositions`

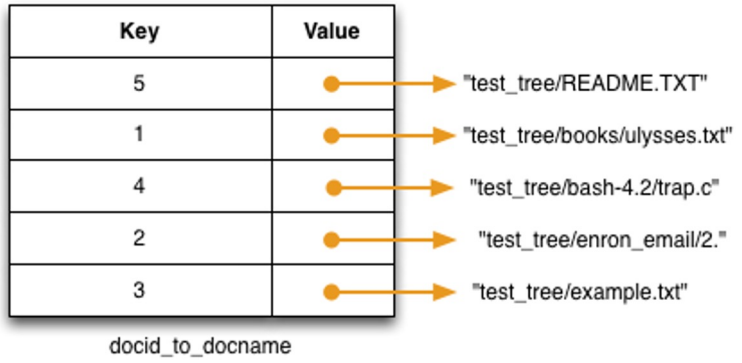
# Part B: Directory Crawling – DocTable

Read through a directory in CrawlFileTree.c

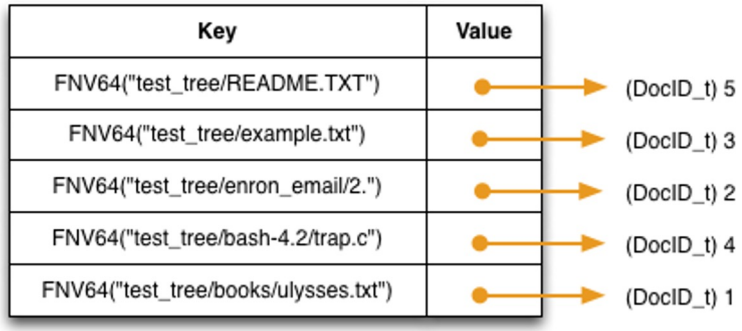
For each file visited, build your DocTable and MemIndex!

DocTable maps document names to IDs. FNV64 is a hash function.

```
struct doctable_st {
 HashTable *id_to_name;  // mapping doc id to doc name
 HashTable *name_to_id;  // mapping docname to doc id
 DocID_t    max_id;      // max docID allocated so far
};
DocID_t DocTable_Add(DocTable *table, char *doc_name);
```



docid_to_docname



docname_to_docid

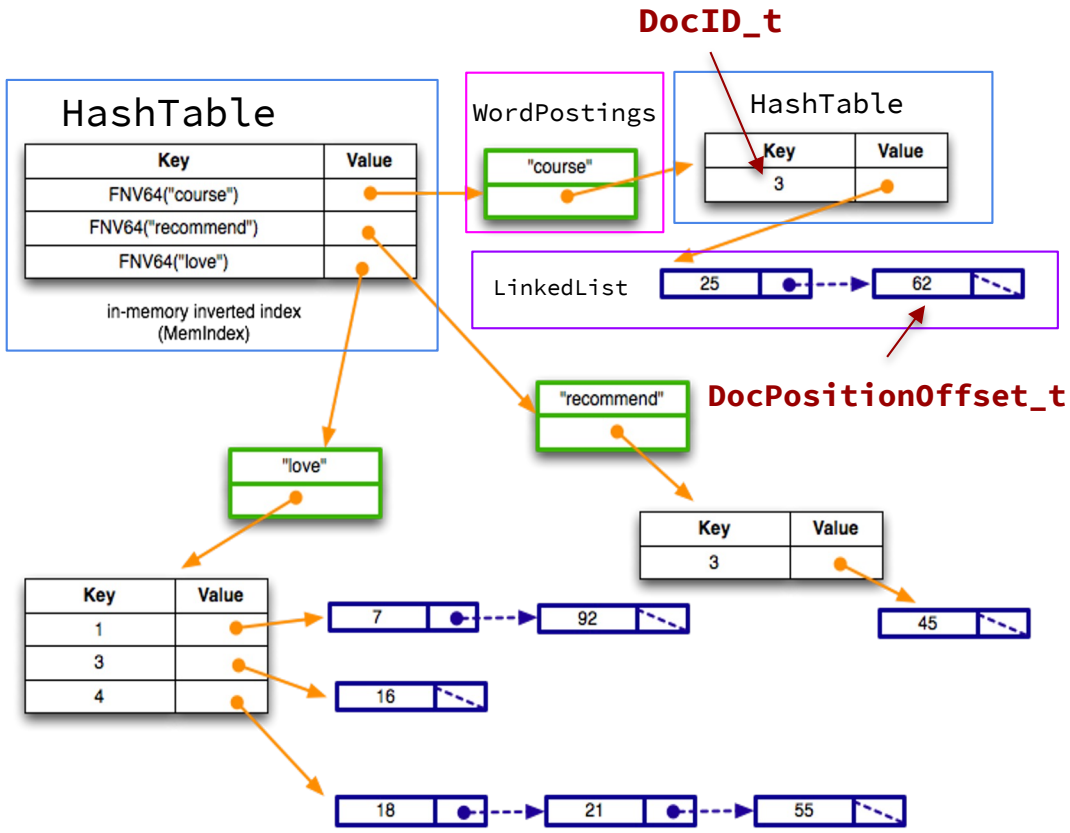
# Part B: Directory Crawling – MemIndex

MemIndex is an index to view files.
It's a HashTable of WordPostings.

```
typedef struct {
 char         *word;
 HashTable    *postings;
} WordPostings;
```

Let's try to find what contains "course":

- WordPostings' postings has an element with `key == 3` (Only DocID 3 has "course in its file")
- The value is the LinkedList of offsets the words are in DocID 3



# Part C: Searchshell

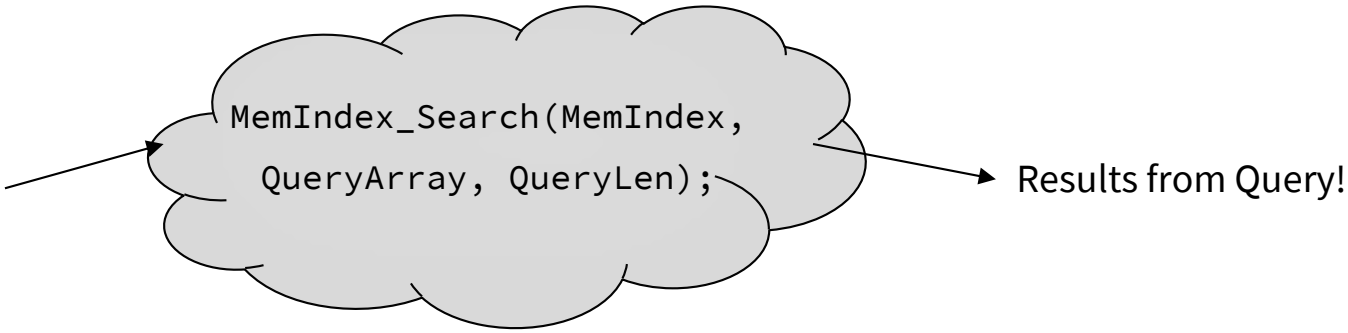
- Use queries to ask for a result!
  - Formatting should match example output
  - Exact implementation is up to you!

MemIndex.h

```
typedef struct SearchResult {
 uint64_t docid;  // a document that matches a search query
 uint32_t rank;   // an indicator of the quality of the match
} SearchResult, *SearchResultPtr;
```

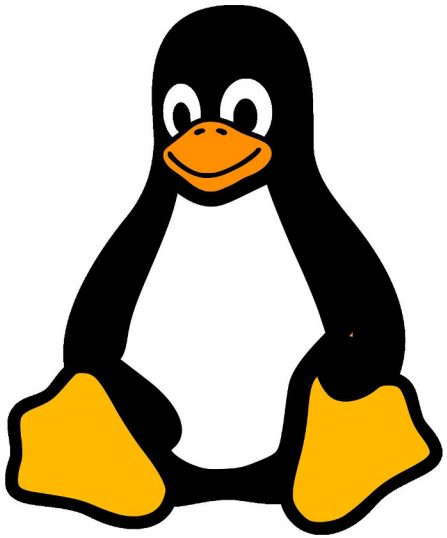
Query

```
course friends my
```



# Hints

- Read the `.h` files for documentation about functions!
- Understand the high level idea and data structures before getting started
- Follow the suggested implementation steps given in the CSE 333 HW2 spec



# Makefile Demo

# Pointers, References, & Const

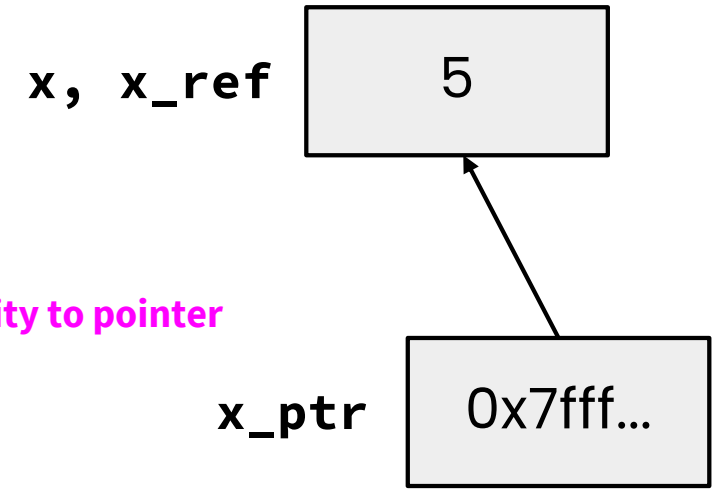
# Example

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**

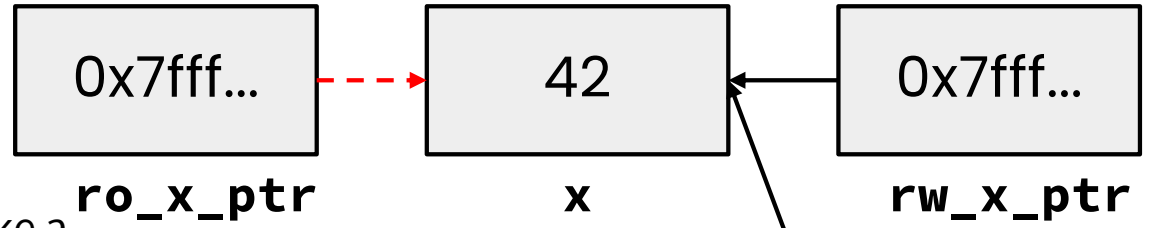
- References the same data for its entire lifetime - *can't reassign*
- No sensible "default reference," must be an alias
- Useful for input parameters:
  `const MyClass &input`

# 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_x_ptr!
int* rw_x_ptr = &x;

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

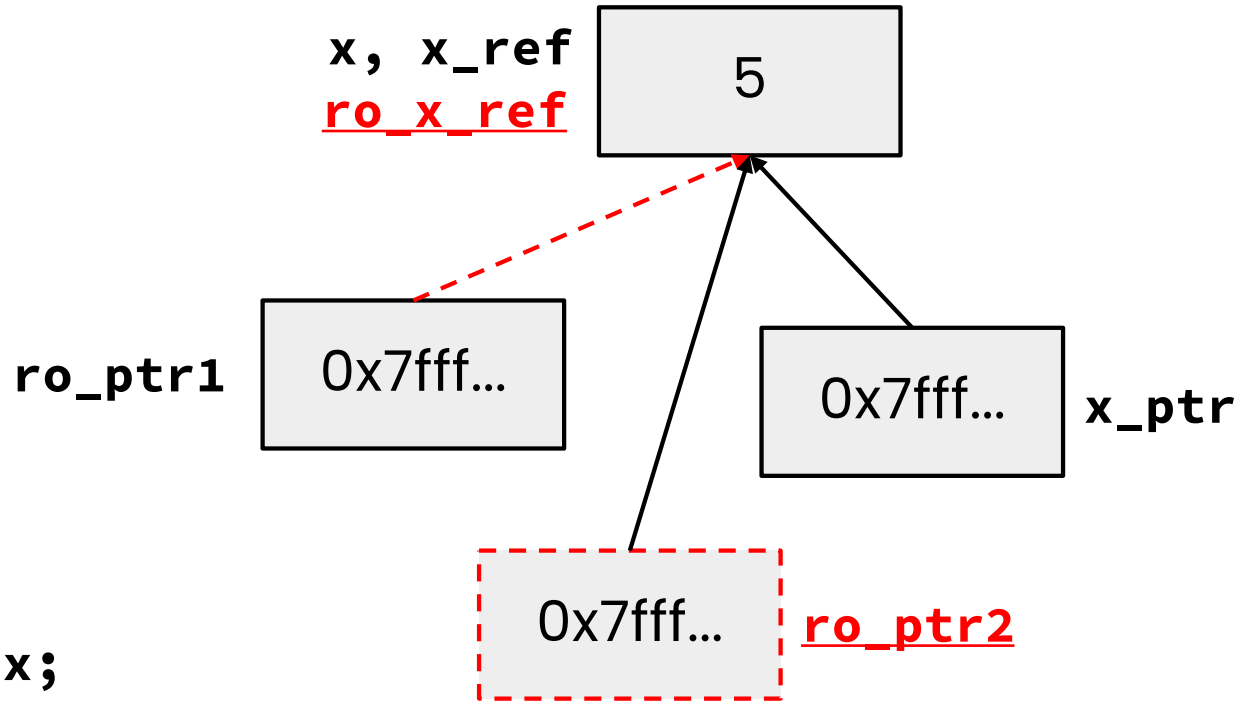
**Legend**
**Red** = can't change box it's next to
**Black** = read and write

# Exercise 1

# 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;
```




"Pointer to a const int"

"Const pointer to an int"

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

**Legend**

**Red** = can't change box it's next to

**Black** = read and write

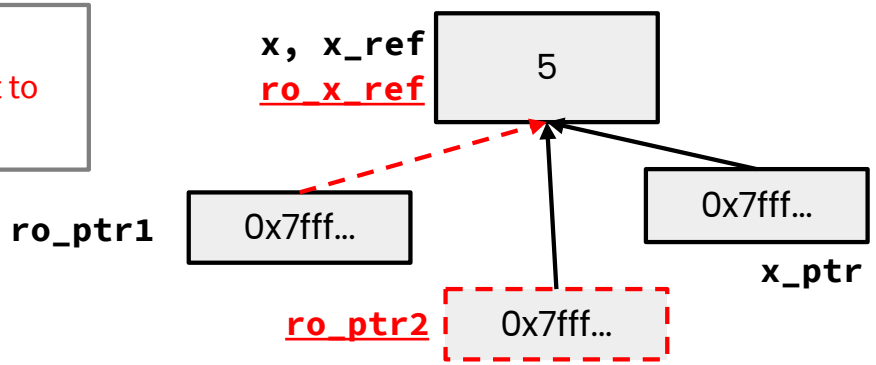
# Exercise 1

When would you prefer `void Func(int &arg);` to `void Func(int *arg);`? Expand on this distinction for other types besides int.

- When you don't want to deal with pointer semantics, use references
- When you don't want to copy stuff over (doesn't create a copy, especially for parameters and/or return values), use references
- Style wise, we want to use **references for input parameters** and **pointers for output parameters**, with the output parameters declared last

# 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 produce a compiler error!**

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

# Exercise 2

# Exercise 2

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
```

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

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

- Make get_resp() const
- Make the parameter to Compare() const
- Stylistically:
  - Add a setter method and default constructor
  - Disable copy constructor and assignment operator