This is Week 7

Jason Hirschhorn

Fall, 2011

Agenda

- Announcements
- Review
  - Problem Set 5
  - GDB
  - Valgrind
- Basic Data Structures
  - Stacks
  - Queues
- Linked Lists
  - Inserting
  - Finding
  - Deleting
- Advanced Data Structures
  - Hash Tables + hashtable.c
  - Binary Search Trees + bst.c
  - Tries
- Problem Set 6
  - Resources
  - // TODO

Announcements

- Problem Set 6 Walkthrough (Sun, 7pm, NW B103) – https://www.cs50.net/psets/
- Office Hours – https://www.cs50.net/ohs/
    - NOT @ Harvard innovation lab this week
- Lecture videos, slides, source code, scribe notes – https://www.cs50.net/lectures/
- Bulletin Board – http://help.cs50.net
- Problem Set 5's Scavenger Hunt
    - Ends 10/31
    - Section pride!
- Problem Set 6's BIG BOARD
    - More section pride!

Review

pset5 – Correctness

```
// allocate space for block from the file
BYTE *buffer = malloc(sizeof(BYTE) * BLOCK);

// check for successful malloc call
if (buffer == NULL)
{
     printf(“Could not allocate the memory.\n”);
     return 1;
}

// free the buffer
free(buffer)
```

pset5 – Design

```
// ensure second argument is an integer
for (int i = 0, n = strlen(argv[1]); i < n; i++)
      if (!isdigit(argv[1][i]))
            return 1;

// save resize factor
int factor = atoi(argv[1]);

// ensure valid resize factor
if (factor < 1 || factor > 100)
      return 2;
```

pset5 – GDB

```
jharvard@appliance (~/pset5/bmp): gdb resize

(gdb) break main
(gdb) run 4 smiley.bmp bigsmiley.bmp

(gdb) next
...

(gdb) print bi
$1 = {...biSizeImage = 3072...}

(gdb) continue
```

pset5 – Valgrind

- Analyzes your code for memory mismanagement

```
valgrind ./resize 4 smiley.bmp bigsmiley.bmp
```

- Good

```
HEAP SUMMARY:
   in use at exit: 0 bytes in 0 blocks
 total heap usage: 3 allocs, 3 frees, 800 bytes
```

pset5 – Valgrind

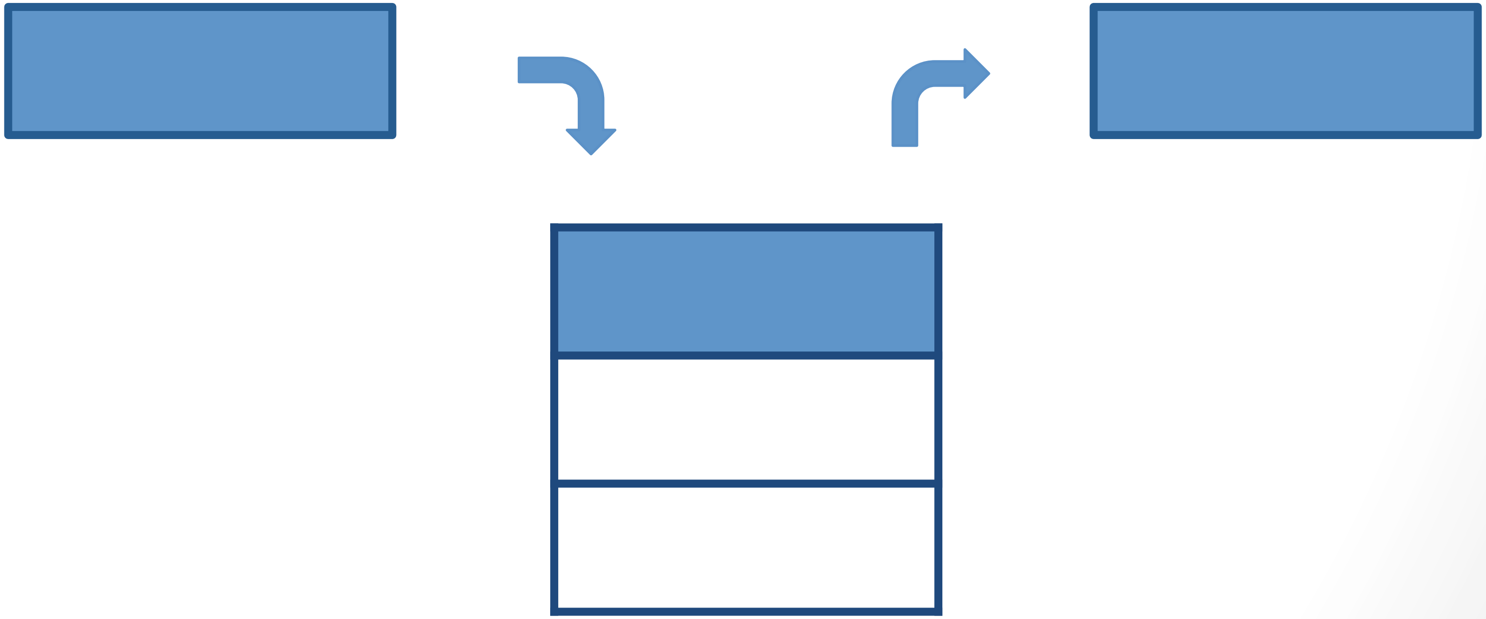
- Bad

```
HEAP SUMMARY:
   in use at exit: 96 bytes in 1 blocks
 total heap usage: 3 allocs, 2 frees, 800 bytes

LEAK SUMMARY
  definitely lost: 96 bytes in 1 blocks

Rerun with –leak-check=full to see details
```

Basic
Data Structures

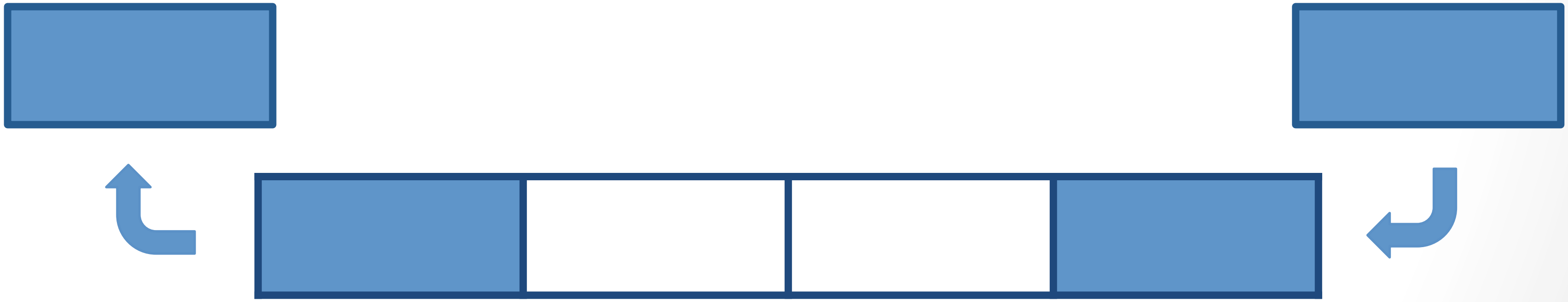
Stacks

- LIFO
  - Last in, first out
- Insert objects on the top (“push”)
- Remove objects from the top (“pop”)



Queues

- ~~British version of a stack~~
- FIFO
  - First in, first out
- Insert objects at the end
- Remove objects from the beginning



Jenga®

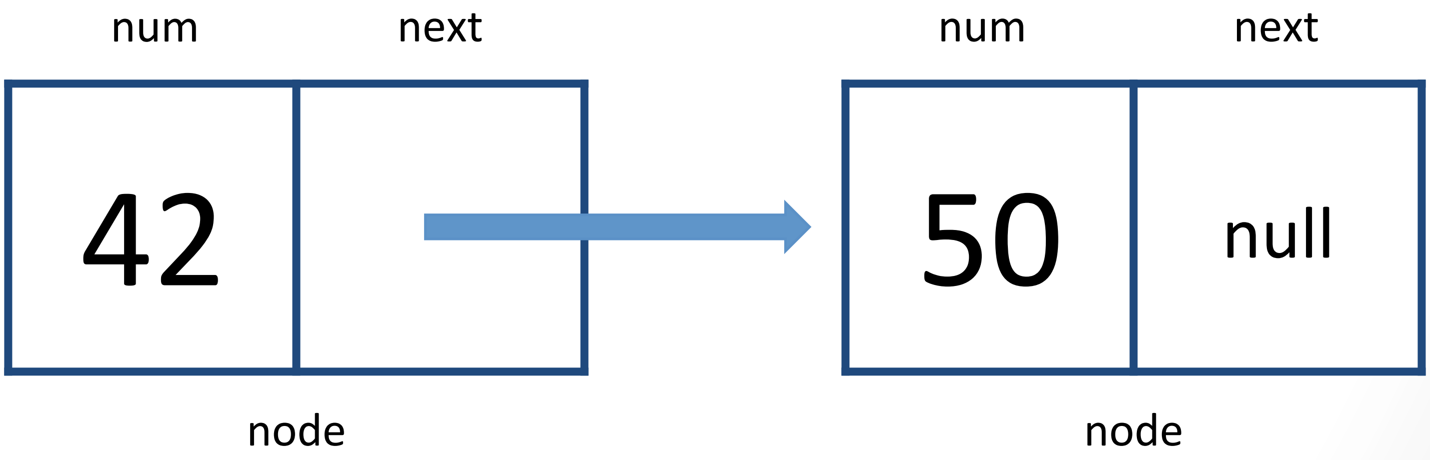
Linked Lists

Resources from http://www.cs.grinnell.edu/~walker/courses/153.sp09/readings/reading-lists-c.shtml

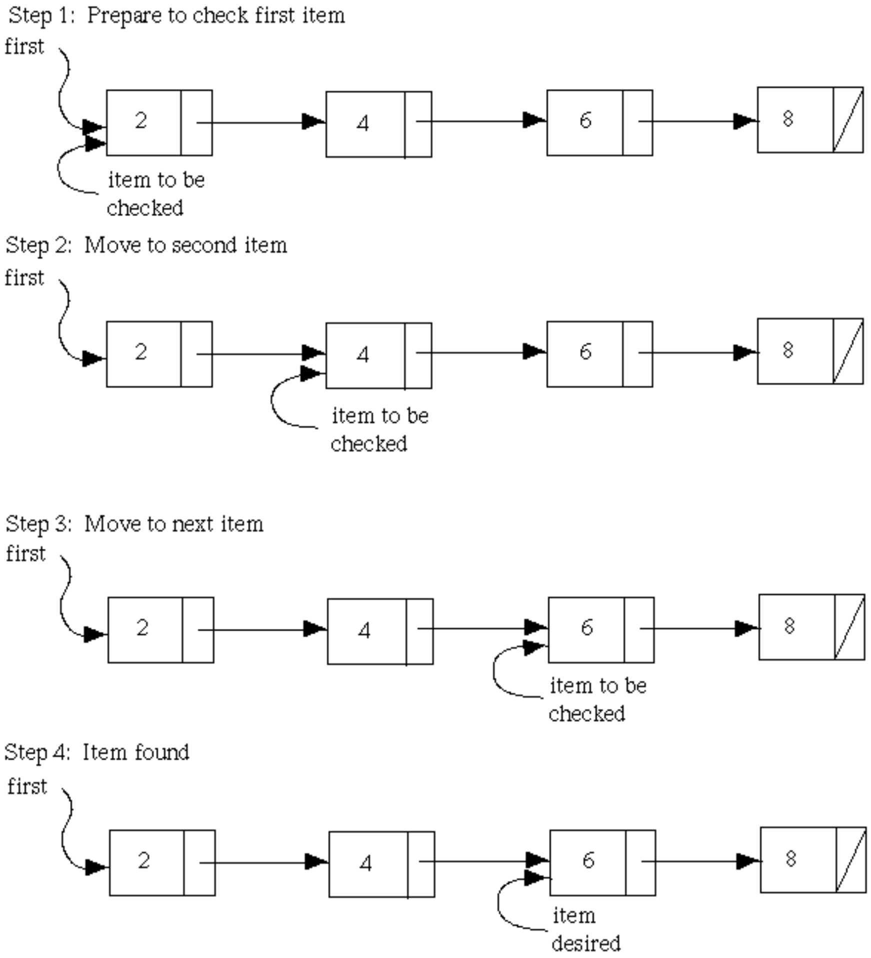
Linked List

- A list of structs
  - “Nodes”

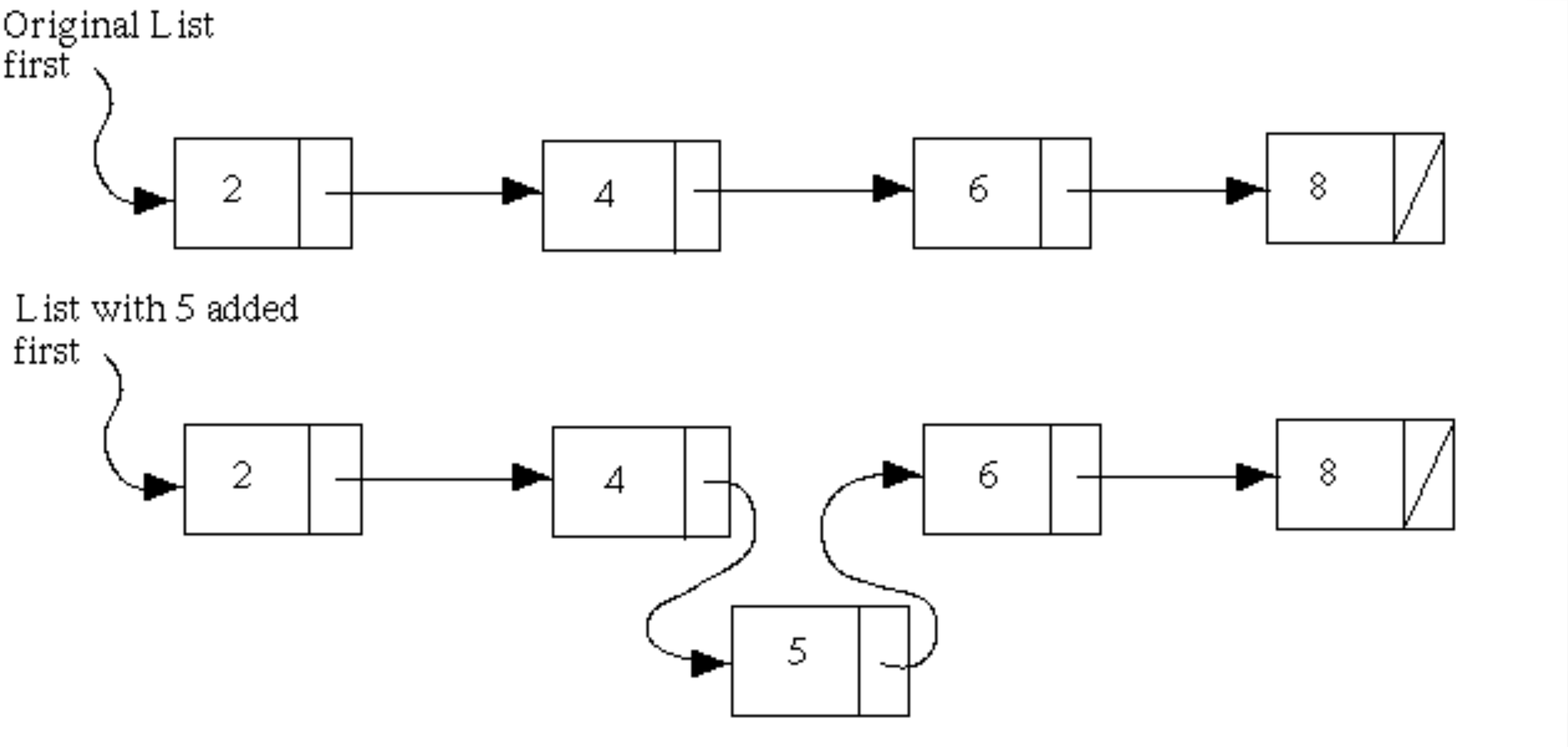
```
typedef struct node {
      int num;
      struct node *next;
} node;
```



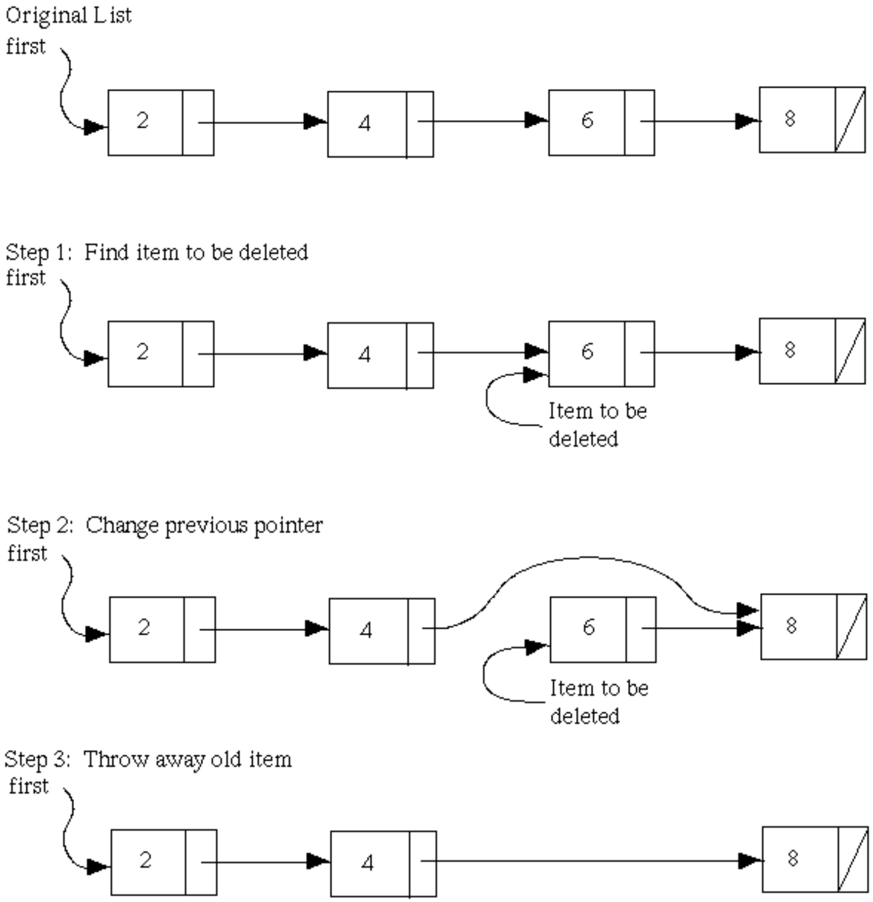
Finding an Object



Inserting an Object



Deleting an Object


Original List
first
2
4
6
8
Step 1: Find item to be deleted
first
2
4
6
8
Item to be deleted
Step 2: Change previous pointer
first
2
4
6
8
Item to be deleted
Step 3: Throw away old item
first
2
4
8

Advanced Data Structures

Hash Tables

- Array + a hash function

Step 1

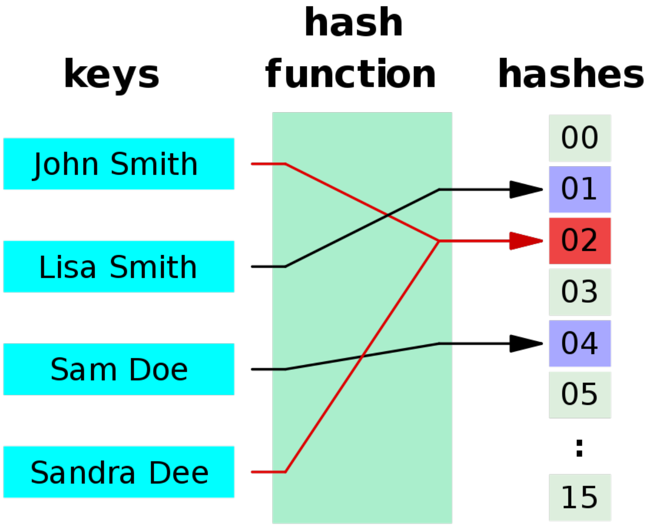
- Key

Step 2

- Value = hash_function(Key)

Step 3

- Array[Value] = Key



Hash Functions

- Good hash functions are
  - Deterministic = it behaves predictably
  - Well distributed = uniformly distributed

Problems

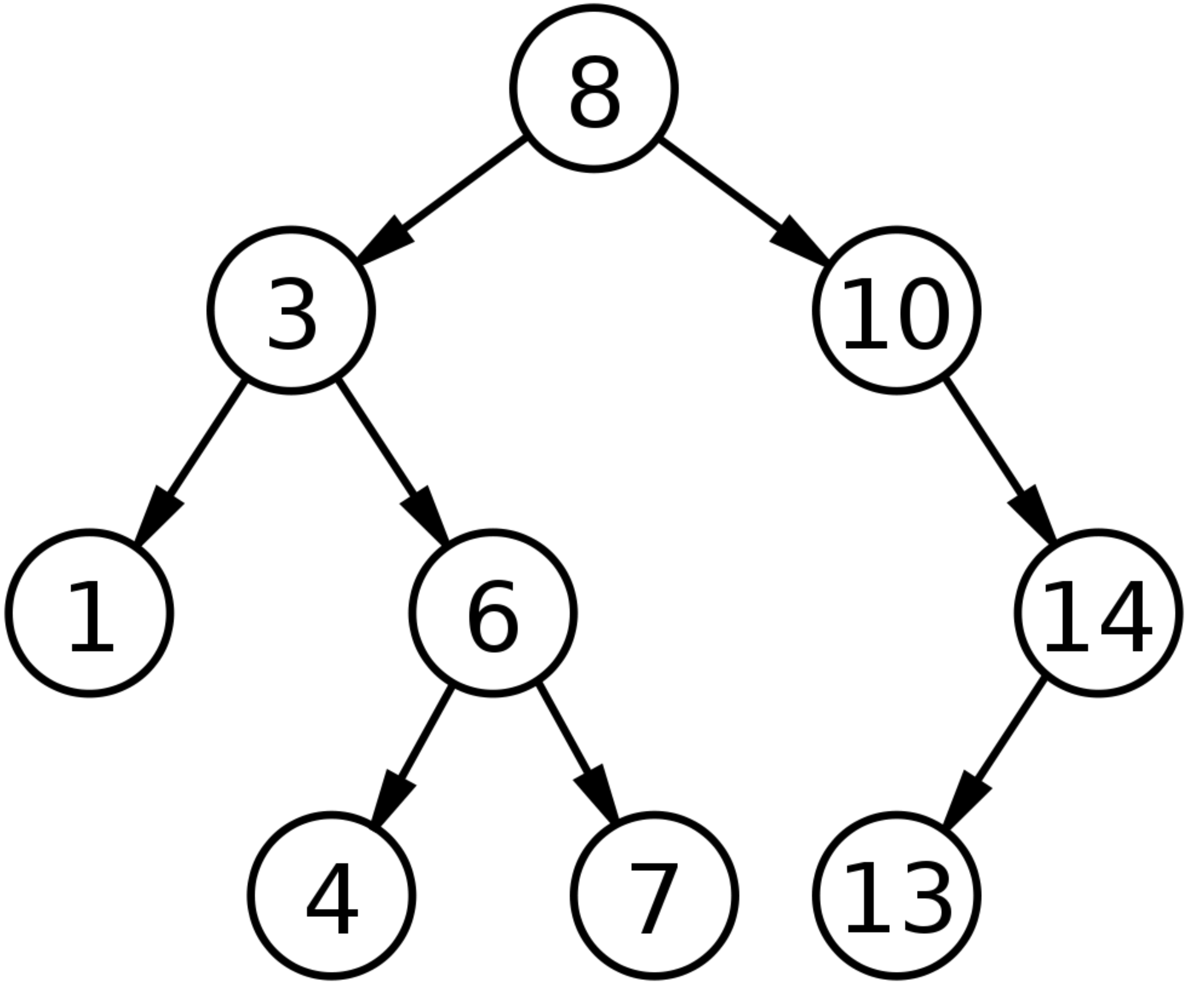
- What if a key maps to a value larger than our hash table?
  - %
- What if two keys map to the same value?
  - Probing = find the next open spot
  - Separate chaining = linked list from that spot

Binary Search Trees

- Like a linked list, but nodes are arranged in a “tree” shape
- Each node has <= 2 child nodes
  - Left child node < parent node
  - Right child node > parent node

```
typedef struct  node {
     int value;
     struct node *left
     struct node *right;
} node;
```

Binary Search Trees



Tries

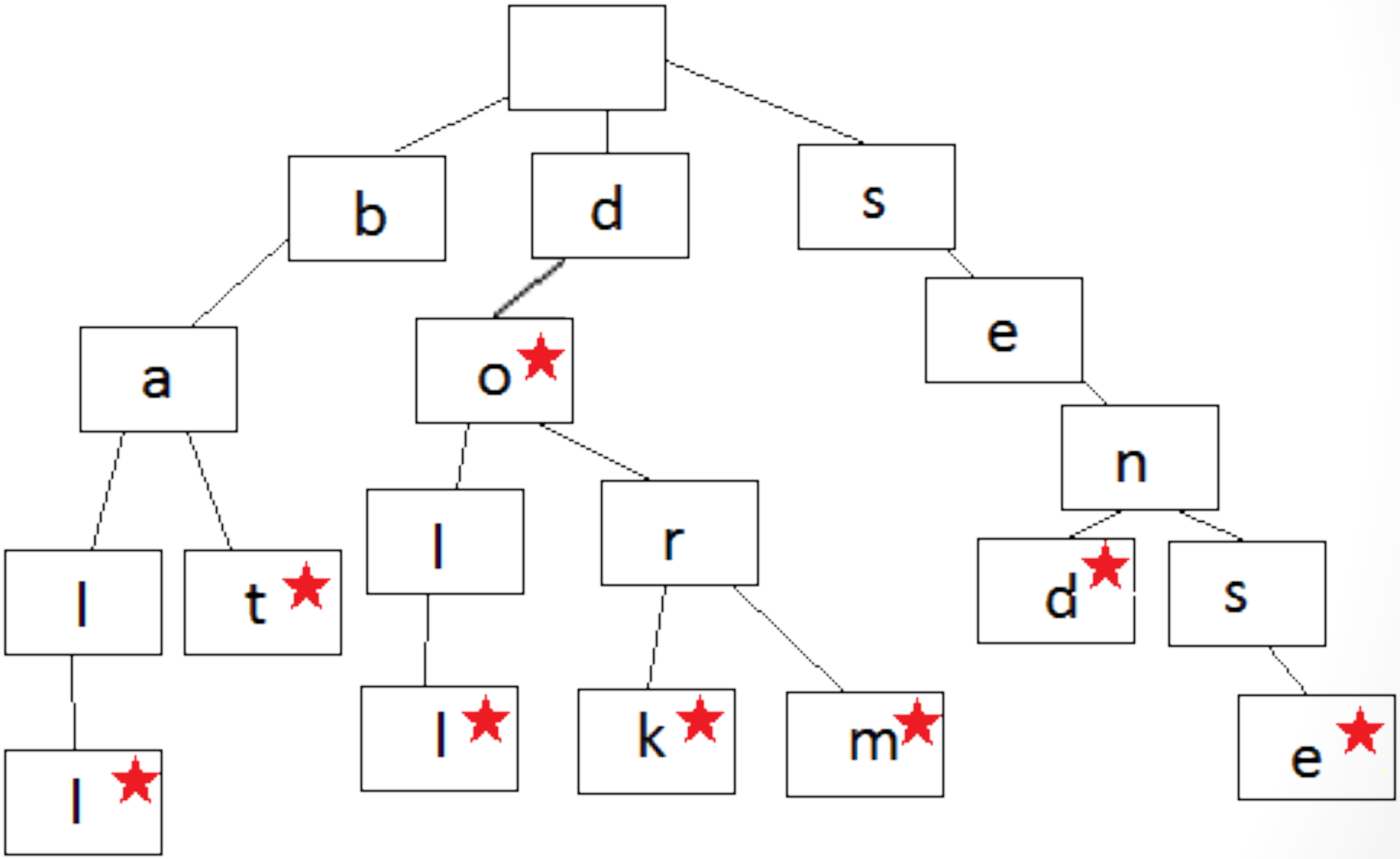
- Like a tree, but each node can have more than 2 children

Example

- A trie that stores words
  - Each child node represents the next letter in some word
  - Each node has <= 26 child nodes

```
typedef struct  node {
      bool is_word;
      struct node *children[27];
} node;
```

Tries



Problem Set 6

Resources

- Google
    - https://www.google.com/
- C Reference Guide
    - https://www.cs50.net/resources/cppreference.com/
- stackoverflow
    - http://stackoverflow.com/
- Google
    - https://www.google.com/

Google

// TODO

- load
    - Put a text file in the dictionary
- check
    - Is the word in the dictionary?
- size
    - How big is the dictionary?
- unload
    - Bye, bye dictionary

Initial Questions

- What type of dictionary (i.e. data structure) do we want to create?
- Since we want to access our dictionary across multiple functions, where should we put it in memory?

That was Week 7

http://www.youtube.com/watch?v=C7hTAp6KrGY