This is CS50.
Week 5

• Data Structures
• Linked Lists
• Trees
• Hash Tables
• Tries
What questions do you have?
Today

Linked Lists
Hash Tables, Trees, and Tries
Lab
PART ONE

Linked Lists
Arrays
Arrays

• Fixed size
• Contiguous in memory
Linked Lists

• Any size
• Not contiguous in memory
typedef struct node
{
    int number;
    struct node *next;
} node;
Linked List

node *list = malloc(sizeof(node));
list->number = 10;
list->next = NULL;
Linked List

list

0x500

next

0x600

number

10

next

0x700

number

0x600

20

next

0x700

30

next

NULL
Insertion list

- number 10
- number 20
- number 30
- number 40
Insertion

list

number
10
next

number
20
next

number
30
next

number
40
next

number
25
next
Insertion

list

number
10
next

number
20
next

number
30
next

number
40
next

number
25
next
Insertion

list

10
next

20
next

30
next

40
next

25
next
Insertion

list

number
10
next

number
20
next

number
30
next

number
40
next

number
25
next
Insertion

list

number 10
next

number 20
next

number 30
next

number 40
next

number 25
next
Insertion

list

number
next

number
next

number
next

number
next

number
next
node *red
node *blue = malloc(sizeof(node));
red->next = blue;
node *green = malloc(sizeof(node));
red->next = green;
green->next = blue
red->next = green
free(red);
free(green);
free(blue);
Insertion

list

number 10
next

number 20
next

number 30
next

number 40
next
Insertion

list

number

next

number

next

number

next

number

next

number

next

number

next

number

next
Insertion

list

Number 10
next

Number 20
next

Number 30
next

Number 40
next

Number 5
next
Insertion

- list
- number 5
- number 10
- number 20
- number 30
- number 40
node *list = NULL;
node *list = NULL;
node *list = NULL;
node *list = NULL;
node *n = malloc(sizeof(node));
node *list = NULL;
node *n = malloc(sizeof(node));
node *list = NULL;
node *n = malloc(sizeof(node));
node *list = NULL;
node *n = malloc(sizeof(node));
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;

n = malloc(sizeof(node));
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;
n = malloc(sizeof(node));
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;

n = malloc(sizeof(node));
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;

n = malloc(sizeof(node));
n->number = 50;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;

n = malloc(sizeof(node));
n->number = 50;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
n->next = list;
node *list = NULL;
node *n = malloc(sizeof(node));
  n->number = 28;
  n->next = NULL;
list = n;

n = malloc(sizeof(node));
  n->number = 50;
  n->next = list;
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
n->next = list;

list = n;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;
n = malloc(sizeof(node));
n->number = 50;
n->next = list;
list = n;
node *list = NULL;
node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
list = n;

n = malloc(sizeof(node));
n->number = 50;
n->next = list;
list = n;
Exercise

Download distribution code at https://cs50.brianyu.me/list.c

Update list.c to:

1) Create a new node and store the number inside.
2) Add the node to the list.
3) When all nodes are added, print the value in each node.
4) Free all nodes.
PART TWO
Hash Tables, Trees, and Tries
Hash Tables
Hash Table

- Array of linked lists
- Use a **hash function** to take an input, and pick a corresponding linked list
int hash(char *s)
{
    return s[0] - 'A';
}
Hash Function

• Deterministic: always maps same input to the same output
• Minimize collisions: fewer collisions means shorter linked lists
Linked List

node *list;
Hash Table

node *table[50];
Trees
Binary Trees
typedef struct node
{
    int number;
    struct node *left;
    struct node *right;
}
node;
Binary Search Trees
Tries
typedef struct node {
    bool word;
    struct node *children[26];
} node;
PART THREE

Lab
Problem Set 5
Problem Set 5

- Speller
## Big Board speller

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Time</th>
<th>Load</th>
<th>Check</th>
<th>Size</th>
<th>Unload</th>
<th>Memory</th>
<th>Heap</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CS50 Staff Solution</td>
<td>7.445s</td>
<td>0.825s</td>
<td>6.165s</td>
<td>0.000s</td>
<td>0.455s</td>
<td>8.0 MB</td>
<td>8.0 MB</td>
<td>2.9 kB</td>
</tr>
</tbody>
</table>

*Time* is a sum of the times required to spell-check *texts/*.txt using *dictionaries/large*. *Memory* is a measure of maximal heap and stack utilization when spell-checking *texts/holmes.txt* using *dictionaries/large*.
This is CS50.