Welcome to CS50 section! This is Week 5.

Please open your CS50 IDE and run this in your console:

```
cd ~/workspace/cs50-section

git reset --hard

git pull
```

If new to this section, visiting, or want to “start over”, run this in your console:

```
rm -r -f ~/workspace/cs50-section/
cd ~/workspace
git clone https://github.com/bw/cs50-section.git
```

The next pset is pset 5, after which you will have a take-home midterm. Afterwards, one week off to take a break (or participate in the coding contest!)
Know before attempting pset 5:

- **Structs**
  - Defining structs
  - Static v dynamic creation
  - Accessing fields

- **Linked lists**
- **Hash tables**
  - Tries

- **Stacks and queues**
Structs (aka structures)

- Encapsulate data of different types together
- Think “object oriented programming”

```c
typedef struct {
    char name[40];
    char github[20];
    int year;
} student;
```
Structs (aka structures)

- What's the datatype of this struct?

typedef struct {
  char name[40];
  char github[20];
  int year;
}
student;
What's the datatype of this struct?

```
struct student
{
    char name[40];
    char github[20];
    int year;
}
```

`student;`
**Structs (aka structures)**

- Created within the global scope
- To create a variable of type “struct student”:
  ```c
  struct student brandon;
  ```
- To assign fields using the dot operator:
  ```c
  strcpy(brandon.name, "Brandon Wang");
  strcpy(brandon.github, "bw");
  brandon.year = 2019;
  ```
Structs (aka structures)

Before moving forward, be comfortable with--

- Conceptual understanding of a struct
  - Good use cases for structs?
- How and where to define a struct
- How to add/modify fields of a struct
- How to modify string fields of a struct
Linked lists

Before pset 5, review--

● Creating a linked list
● Inserting into a linked list
  ○ At the head
  ○ At the tail
  ○ In the middle
● Deleting from a linked list
● Deleting an entire linked list
● Iterating over a linked list
Linked lists

Conceptually, what’s a linked list?
Linked lists

Conceptually, what’s a linked list?

Programmatically, what’s a linked list?
Linked lists

Conceptually, what’s a linked list?

Programmatically, what’s a linked list?

typedef struct node {
    // just some form of data; could be a char* or whatever
    int i;

    // pointer to next node; have to include `struct` since this is a
    // recursive definition
    struct node *next;
} node;
Linked lists

Conceptually, what’s a linked list?

Programmatically, what’s a linked list?

typedef struct node {
    // just some form of data; could be a char* or whatever
    int i;

    // pointer to next node; have to include `struct` since this is a
    // recursive definition
    struct node *next;
} node;
Tricky things about linked lists

- How do we iterate over a linked list?
- How do we insert/delete a node?
  - Be careful of node orphaning
Tackling pset 5

Overarching decision you must make:

What data structure do I use?

- Hash table
- Trie

“you should not encourage any student to pursue any implementation”
Hash tables

- Associative array
- Position of each element determined by a “hash function”

Hash functions:

- Take an input and generate a reproducible output
- Ideally: constant time output and few collisions
Hash tables

Best of both worlds approach:
Combines “random access ability” of an array with the “dynamism” of a linked list

Assuming we define our hash table well:

- Insertions tend towards $O(1)$
- Deletions tend towards $O(1)$
- Lookups tend towards $O(1)$
What defines a good hash function? Ideally--

- **Be deterministic**
- Use *only* the data being hashed
- Use *all* of the data being hashed
- Uniformly distribute data

For some hash function applications:
- Generate very different results for very similar (but different) data
Example hash function

```c
unsigned int hash(char* str) {
    int sum = 0;

    for (int j = 0; str[j] != '\0'; j++) {
        sum += str[j];
    }

    return sum % HASH_MAX;
}
```
Hash tables → Collisions

Collisions occur when two pieces of data yield the same code.

If storing data, we want both pieces of data.
So we need to get both elements in the hash table.
Hash tables → Collisions

- Handling collisions
  - Linear probing
    - Clustering
  - Chaining via linked lists
Tries
Tries

Tries: roadmaps for hash tables.

- If you can follow the map from beginning to end, the data exists.
- If you can’t, it doesn’t exist.
- No collisions possible/allowed
Tries

See CS50-standard slides for example scenario.
Stacks and queues

We will talk about these conceptually today.
Code snippets available after section (brandon.wang/cs50)
Stacks and queues

We will talk about these conceptually today. Code snippets available after section (brandon.wang/cs50)

What do stacks and queues do?

- Maintains data in an organized way
- Optimized for a specific type of data access
- Really bad for other forms of data access

- Usually we implement as an array or linked list
Stacks

Stacks are **last in, first out** (LIFO).

Allowed operations:

- **Push**
  - Add an element to the top of the stack

- **Pop**
  - Grab the most recently added element from the top.
Stacks

typedef struct stack {
    VALUE array[CAPACITY];
    int top;
} stack;
Stacks

typedef struct stack {
    VALUE array[CAPACITY];
    int top;
} stack;

How would we push/pop elements?
Queues

Queues are **first in, first out** (FIFO).

Allowed operations:

- **Enqueue**
  - Add an element to the end of the queue
- **Dequeue**
  - Remove the (oldest) element from the front of the queue
Queues

Best implemented as a linked list.

typedef struct queue {
    VALUE val;
    struct queue *prev;
    struct queue *next;
} queue;

Maintain pointers to head AND tail of the list.
Data structures summary
Data structures

Four primary ways we’ve looked at in CS50:

- Arrays
- Linked lists
- Hash tables
- Tries
Arrays

- Insertion
- Deletion
- Lookup
- Ease of sorting
- Size
Linked lists

- Insertion
- Deletion
- Lookup
- Ease of sorting
- Size
Hash tables

- Insertion
- Deletion
- Lookup
- Ease of sorting
- Size
Tries

- Insertion
- Deletion
- Lookup
- Ease of sorting
- Size
That’s all for today!