Announcements

• Midterm on Monday
• Midterm review at Friday’s lecture
• Homework assignment due today
• Read chapters 5-7 in book
Using Heap Allocated Structures

- Draw a picture of the data structure
- Rounded boxes represent the objects that compose the data structure
- Arrows represent pointers
- Two types of pointers in pictures
  - Pointers that originate from objects – these connect the objects together (and provide a path to access one object from another)
  - Pointers from variables – these give your code some way to access objects
Example

- Red arrow represents variable that is a pointer:
  ```c
  struct list * listhead;
  ```
- Yellow arrows are pointers that are stored in each of the objects
Specifying the Pieces

- Need to specify what the parts (ie. objects) that make up the data structure look like (what fields do they contain)
- Do this with a structure definition

```c
struct foo {
    int x;
    float y;
    struct foo * foo_ptr;
};
```
Data Structure Operations

- List the operations that the data structure will support
- For each operation, draw a sequence of pictures breaking down the steps
Drawing Pictures

1.

2.

3.

4.
Translating Pictures into Code

• Problem:
  How do we give objects in the picture names so that we can manipulate them

• Solution:
  Use variables

```
listhead
```

```
[ ] ➔ [ ] ➔ [ ] ➔ [ ]
```
Translating Pictures into Code

• Problem:
  How do we give objects in the picture names so that we can manipulate them

• Solution:
  Use variables

```c
struct list tmpptr=listhead->next;
```

![Diagram of list structure]
Translating Pictures into Code

- Problem: When do we use malloc?
- Answer: Only when we draw a new box in the picture.

2.
Translating Pictures into Code

- Problem: When do we use `free`?
- Answer: Only when we make a box in the picture disappear
- Caveats: After we call `free` on an object, we can’t touch it again or we’ll get a segmentation fault
- So make sure we’ve taken it out of the data structure first and that we don’t need any information in it!
Finding the place to insert

• Suppose we want to insert an object at a specific point...
• We first need to find the point – use a loop to search through the data structure
• Could even draw pictures for the search!
• After we find the spot, then do manipulation!
Allocating memory in the heap

• `malloc` (memory allocate) function
  • need to include `stdlib.h` to use
  • `int * ptr = (int *)malloc(sizeof(int));`
    • takes in a size (in bytes)
    • returns a pointer to a block of memory of the specified size
    • or NULL if there is an error
  • `struct list * list_ptr=(struct list *)malloc(sizeof(struct list));`
Freeing memory in the heap

• when you’re done with the memory use free to return it
  • free(ptr);
  • free takes in a pointer to beginning of memory block and returns that block back to the operating system