Outline

1 Memory Model
2 Data structures
3 Floating Point
4 Wrap-up
If you were registered in 6.179, I probably don’t have your registration.
I’m looking into this, but you might need to submit an add form.

Make sure to get an account on the grader; email me if you are not on the roster
You need an MIT certificate, and use the exact link here:

Use Piazza if you get strange compiler errors!
Suggestion: google the error, see if you can understand it- if’s not immediately obvious, post the error + relevant code on Piazza (plus the explanation if you figure it out).
Memory model review

1. Stack memory (local variables, function arguments/calls, return address, etc)
2. Heap memory (malloc)
Memory model review

1. Stack memory (local variables, function arguments/calls, return address, etc)
2. Heap memory (malloc)

Where is each located? Based on architecture..

1. x86/x86_64: Stack grows down
2. ARM: selectable
3. SPARC: circular stack?!
Memory model review

1. Stack memory (local variables, function arguments/calls, return address, etc)
2. Heap memory (malloc)

Where is each located? Based on architecture..

1. x86/x86_64: Stack grows down
2. ARM: selectable
3. SPARC: circular stack?!

In general, the stack will be growing down from upper memory addresses while the heap grows up.

Stack var: 0x7ffffa4c77170
Heap var: 0x000001ede010
Stack Diagram: (From Stack Overflow - http://goo.gl/t2PQo)

- **sp**: stack pointer (data)
- **used**: unavailable stack
- **args**: function arguments
- **ret**: return address (code)
- **locals**: local variables
- **free**: available stack
malloc and the Heap

Statically allocated

```c
int array[10];
int array2[] = { 1, 2, 3, 4, 5 };
char str[] = "Static string";
```

Dynamically allocated

```c
#include <stdlib.h>
int *array = malloc( 10 * sizeof( int ) );
// do stuff
array[5] = 5;
//when done
free( array );
```
What’s happening with the file IO?

You’re allocating resources for a file handle (stored in a FILE pointer), which you must then

```c
#include <stdio.h>

int main(void) {
    // open file for writing
    FILE *output = fopen( "prog.out", "w" );
    // do stuff; then close and free resources
    fclose(output);
    return 0;
}
```

Every time you `malloc`, you must remember to `free` the memory when you are done! **C does not do this for you** and it will otherwise result in a resource leak.
Let’s go over that again...

“I promise to always free each chunk of memory that I allocate.”
Let’s go over that again...

“I promise to always free each chunk of memory that I allocate.”

Don’t be the cause of memory leaks!
It’s a bad practice.
Array Indexing: Syntactic sugar

C doesn’t know what an array is, really. 
T array[] and T *array = malloc(...) are both pointers to contiguous blocks of memory.

```
int array[10];
// Initialize
for( int i = 0; i < 10; ++i ) {
    array[i] = i;
}
// Does the exact same thing as above
for( int i = 0; i < 10; ++i ) {
    *( array + i ) = i;
}
```
If C only knows about memory, how do we get it to understand a data structure?

```c
struct IntPair_s {
    int first;
    int second;
};

// in code:
struct IntPair_s pair;
pair.first = 1;
pair.second = 2;
struct IntPair_s *pairPtr = &pair;
// use pairPtr->first and pairPtr->second
// to access elements
```
If C only knows about memory, how do we get it to understand a data structure?

```c
typedef struct IntPair_s {
    int first;
    int second;
} IntPair;

// in code:
IntPair pair;
pair.first = 1;
pair.second = 2;
IntPair *pairPtr = &pair;
// use pairPtr->first and pairPtr->second
// to access elements
```
Floating Point

Real numbers have to be represented in memory in some finite way:

A floating point number `float x` with sign bit 'sign', exponent `e`, and mantissa bits `m_0, m_1, ..., m_{22}` can be written\(^1\)

\[
x = (-1)^{\text{sign}} \cdot \left(1.m_{22}m_{21}m_{20} \ldots m_0\right) \cdot 2^{e-\text{bias}}
\]

where bias is, in our case, 127.

---

\(^1\)Unless it's denormal, which we'll cover shortly.
Let's see some pictures...

**float (32 bits)**

- **sign exponent (8-bit)**
- **fraction (23-bit)**

```
0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

31 23 0

= 0.15625

**double (64 bits)**

- **sign (11 bit)**
- **exponent (11 bit)**
- **fraction (52 bit)**

```
```

63 52 0
Subtleties

Rounding and precision

Denormals

Long doubles
Examples

Time for some examples...
First assignment is posted: four problems total 1000 points

- floating (300)
- matrix (200)
- matrix2 (300)
- loop (200)
Wrap-up & Monday

Class on Monday is back in 54-100 at 2pm.

- Two shorter guest lectures:
  - Daniel Kang presenting x86 Assembly
  - Lef Loannidis presenting Secure C

Questions?

- I’m available after class or on Piazza.