6.S096 Lecture 2 – Subtleties of C

Data structures and Floating-point arithmetic

Andre Kessler

Outline

Memory model review

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

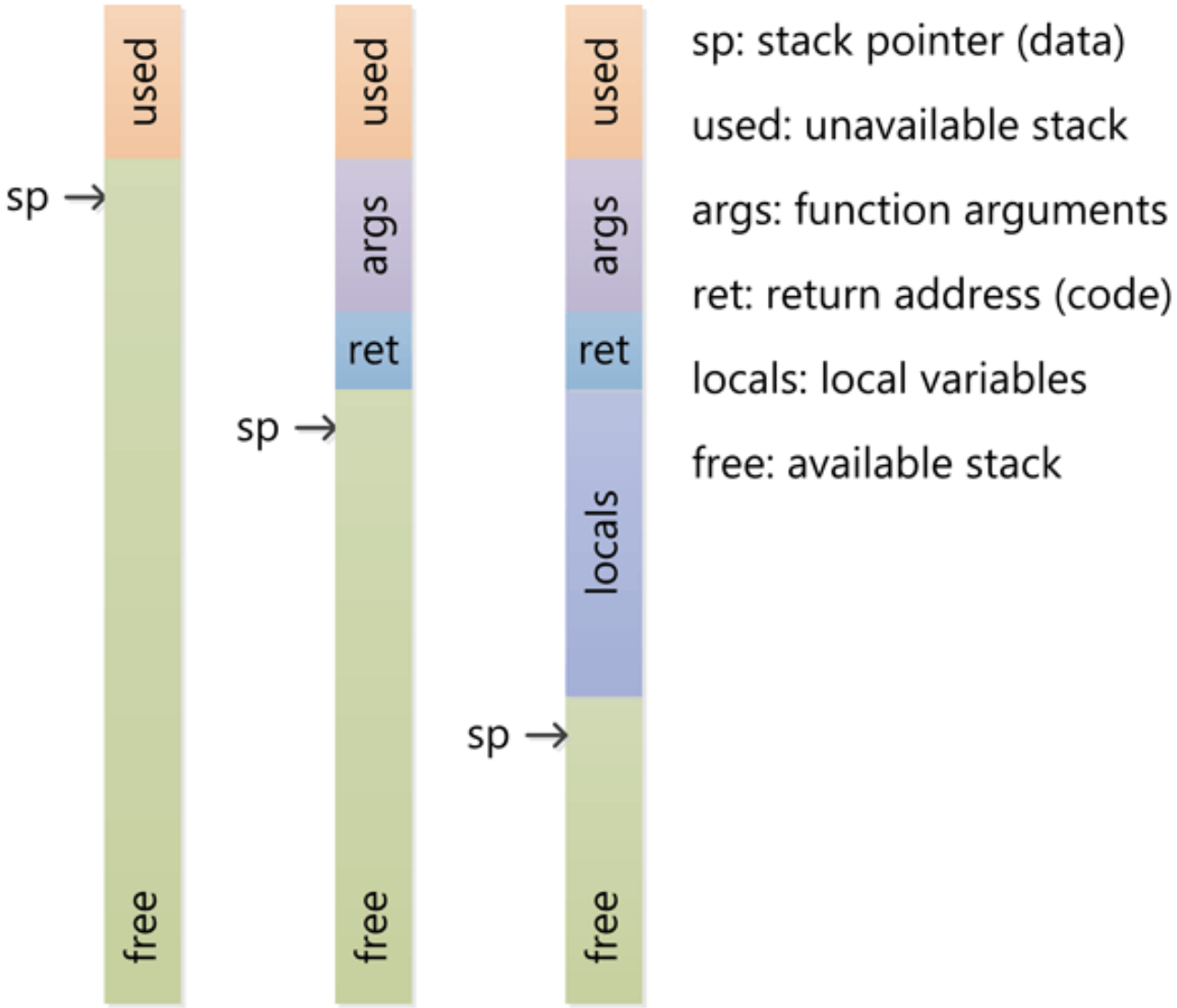
Where is each located? Based on architecture..

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:  0x7fffa4c77170
Heap var::  0x000001ede010
```

Stack Diagram: (From Stack Overflow - http://goo.gl/t2PQo)



malloc and the Heap

Statically allocated

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

Dynamically allocated

```
#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."

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;
}
```

Structs

If C only knows about memory, how do we get it to understand a data structure?

```
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
```

Structs and Typedef

If C only knows about memory, how do we get it to understand a data structure?

```
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 represented in memory in some finite way:

A floating point number `float` x with sign bit 'sign', exponent e, and mantissa bits $m_0, m_1, \ldots, m_{22}$ can be written[1]

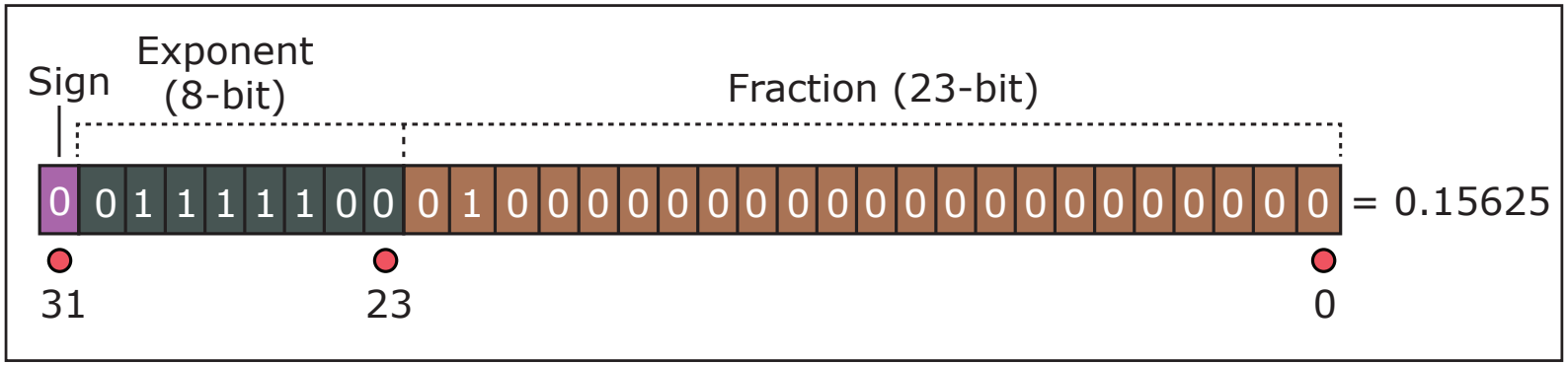
$$x = (-1)^{\text{sign}} \cdot (1.m_{22}m_{21}m_{20}\ldots m_0) \cdot 2^{e-\text{bias}}$$

where bias is, in our case, 127.

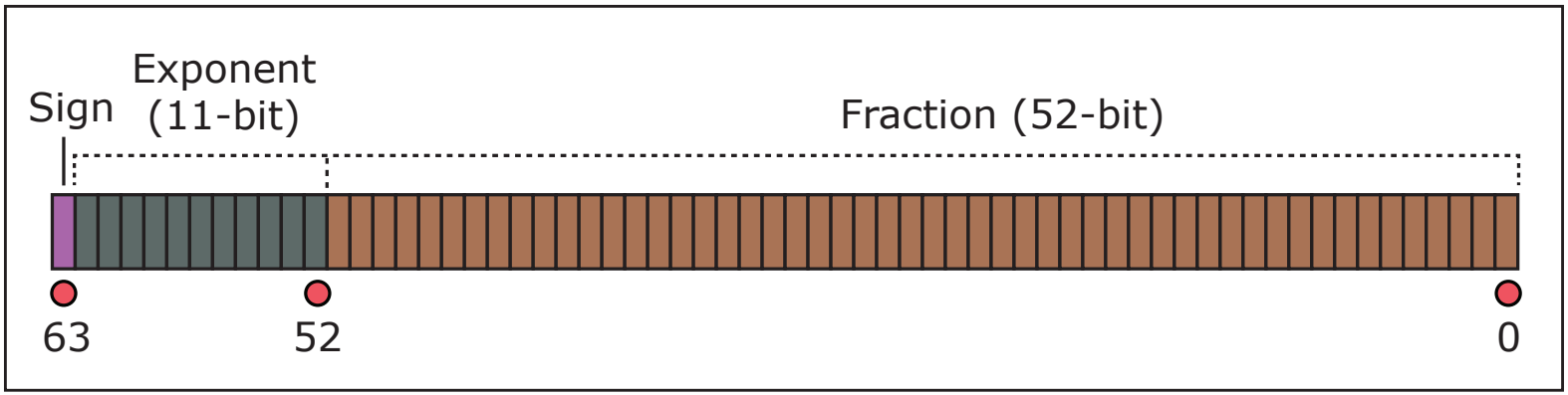
[1] Unless it's denormal, which we'll cover shortly.

Let's see some pictures...

float (32 bits)



double (64 bits)



Images by MIT OpenCourseWare.

Subtleties

Rounding and precision

Denormals

Long doubles

Examples

Time for some examples...

First Assignment

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

Two shorter guest lectures:

Daniel Kang presenting x86 Assembly

Lef Ioannidis presenting Secure C

Questions?

6.S096 Effective Programming in C and C++

IAP 2014