# Scribe Notes 10/21

**Reading a Cumulative Distribution Function Graph:**

In this graph: The highest grade is around 78, the lowest is around 26, & the mean is around 55.
Percent of exams <= x

```
100%  |                 /’’’’’’’’
      |                /
      |               /
      |           ,~~’
50%   |          /
      |      ,~~’
      |     /
      |    /
0%    |~~~~’_______________
       0   25   50   75   100
            Grade Number
```

**Registers:**

%eax, %ebx, %ecx, %edx, %esi, %edi, %eip, %esp, %ebp
All separate, but the same hardware-style
They're fundamentally addressed, can't index into them

Which ones have important values at the start of a function and which are garbage?
Callee-saved registers are important, caller-saved registers are garbage.

Callee-saved registers:
- Callee must restore original value
- Value will be the same at the end of the function as it was at the beginning
- ebp, ebx, esi, edi, esp, ebp

Caller-saved registers:
- Callee can modify without restoring
- Value may have changed by the end of the function
- eax, ecx, edx

%eax contains return value.
%esp contains return address; it tells the processor where to go after the instructions are complete.

**Stack & Heap:**

The stack is like a stack of papers; the top (item 0) is the item that can be accessed the fastest.

The running function is at the top of the stack. Everything else is asleep and has no access to the processor.

The stack grows downward from the top and the heap grows upwards from the bottom.
Why?
If they both grew in the same direction, the memory would easily become fragmented, as the max space of each would be half of the total space. Instead, they grow towards each other, so they share the same space and have a shared max of the total space.

**Aligning**:

Calling convention: Compilers ensure that the stack pointer (%esp) is always aligned to 16.

```
subl    $12, %esp
call    g
addl    $12, %esp
ret
```

The subtraction to the stack pointer sets the pointer for the next function to 12 bytes after the current function's pointer. 12 bytes because the pointer itself is 4 bytes and the next one must be aligned to 16 bytes (12 + 4).

**More About Registers:**

Positive offsets off of %ebp are the current function’s arguments. Negative offsets off of %ebp are local variables. Positive offsets off of %esp are the next function’s (the callee's) arguments.

When a function is complicated (has local variables or arguments), it'll usually start with:

```
        pushl   %ebp
```

But this call adds 4 bytes to the stack, so the X in the future "subl X, %esp" instruction will be 4 less bytes to keep alignment (see: Aligning).

**More Assembly Info:**

"leave" pops %ebp.
“testl” is a bitwise AND.
Compilers like to convert while loops into an initial test and then a do-while loop