CS 241 | Lecture Handout #4

Q1 Thinking about pointers...

```
int ****** ptr;
```

Q2. Using `read()`:

```
ssize_t read(int fd, void *buf, size_t count);
```

...what type of call is read?

...how would we use it?

```
1:
2:
3:
```

Q3. Using `scanf()`:

```
int scanf(const char * format, ...);
```

In **scanf**, the format string is the same as **printf** except that every type must be passed by reference to be written into by **scanf**:

Specifier:	d i	u o x	f	c s	p
Type:					

scanf return value (and why is it useful)?

Example:

```
int num;  char c;
int result = scanf("%d %c", &num, &c);
printf("Values: %d %c\n", num, c);
printf("Return value: %d\n", result);
```

...what is the return value for the input: **7 hello**

...what is the return value for the input: **6** *(...followed by an EOF)*

Q4 fscanf, scanf, sscanf ?

```
10,23
20,25
30,37
...
```

How can I read and process my data?

```
1: FILE *file = fopen("mydata.csv","r");
2:
3:
```

Q5. Using `getline()`:

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

The C-string passed by reference as **lineptr** will store the line; the size of the memory allocated in **lineptr** must be stored in **n** (to avoid overflow). Additionally:

> If ***lineptr** is set to **NULL** and ***n** is set **0** before the call, then **getline()** will allocate a buffer for storing the line. This buffer should be freed by the user program even if **getline()** failed.

...found in ***man getline***

Example usage:

```
  1:  char *s = NULL;
  2:  int n = 0;
  3:  getline(&s, &n, stdin);
  4:  getline(&s, &n, stdin);
  5:
...:  ...
  n:  free(s);
```

Q6. Processes: What are they? Can I have new one?
A process is the base computation container on Linux; multiple processes allow for multiple separate (and parallel) execution.

Is there a system call to make a new process?

Q7. Environmental Variables
A process-specific dictionary that stores information about the execution environment:

- Command line:

- C programming: