Thinking about pointers...

```
1: int ****** ptr;
```

Puzzle #1: Print out the arguments to a process

```
 1: int main(int argc, char**argv){
 2:
 3:
 4:
 5:
 6:
 7:
 8:
 9:
10:
```

Using `read()`:

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

...what type of call is read?

...how would we use it?

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

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:					

Return value?

Example:

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

...what is the return value of the input: **`7 hello`**

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

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);
```

Processes:
A process is the base computation container on Linux; multiple processes allow for multiple separate (and parallel) execution.

Q: System call to make a new process?

Environmental Variables
Process-specific dictionary that stores information about the execution environment:

- Command line:

- C programming:

Meta Example: *"Let is snow, let it snow!"*
`snowflake.c` attempts to create a snowstorm where every snowflake is a process *(found in /_shared/ in the CS 241 svn)*. Screen cursor logic is provided, simple API is:

- **`int rows`**: contains the number of rows of the terminal/console
- **`int cols`**: contains the number of columns of the terminal/console
- **`gotoxy(x, y)`**: moves cursor to a given **`x`**, **`y`** position

The key function, **`snowflake()`**:

```
 1: void snowflake() {
 2:   srand((unsigned)time(NULL));
 3:   int col = rand() % cols;
 4:   int row = 0;
 5:
 6:   while (row < rows) {
 7:     gotoxy(row, col);
 8:     fprintf(stderr, "*");
 9:     usleep(200000);
10:     gotoxy(row, col);
11:     fprintf(stderr, " ");
12:     row++;
13:   }
14: }
```