Puzzle 1: What is the value of `p[strlen(p)]` **, if** p **points to** `"Sys"` **?**

Hint: `strlen("") ==0 strlen("@")==1, strlen(NULL)==`#@?#WT?!

Q1: How do I find out how to use
____< **useful function or system call here>**___?

$	

Puzzle 2: How do I find out how to use <u>stat</u> **in C?**

Q2: What are the manual sections?
- Section 2:

- Section 3:

- Section 7:

Q3: How do I allocate and free heap memory in C?
- Allocate:

- Free:

Q4: Can I make a pointer *really free* **by freeing it twice?**

Q5: What do we call a pointer that has been free'd?

Best Practice: <u>Always</u> set free'd pointers to NULL.

```
1:  // ... code ...
2:  free(ptr);
3:  ptr = 0;
```

Puzzle 3: Fix a custom string concatentation function:

```
1:  void mystrcat(char *dest, const char *src) {
2:
3:
4:    while (*src) {
5:
6:
7:      dest = src;
8:
9:
10:     src++; dest++;
11:
12:
13:   }
14: }
```

Puzzle 4 - Walk Through

Type	Variable	Memory Addr.
const char *	src	0x1000
char *	dest	0x2000

Address	Memory Contents
1000	'!'
1001	'2'
1002	'B'
1003	'\0'
...	
2000	'2'
2001	'B'
2002	'|'
2003	'\0'
2004	
2005	
2006	
...	

⇨ **Line 3:** What does `(*src)` do?

⇨ **Line 4:** What does `(dest = src)` do?

⇨ **Line 3..9:** When does the loop exit?

Puzzle 5: Fix my custom string duplication function

```
1:  char *mystrdup(const char *src) {
2:
3:
4:    char *p = sizeof(src);
5:
6:
7:    strcpy(src, p);
8:
9:
10:   return p;
11: }
```

Q6: What is the purpose of a file stream, just files?

A "file stream" (or "file descriptor" in system calls) is the base interface to <u>EVERYTHING</u> external to RAM. This includes:

-

-

-

- Standard Streams:

    - **stdin:**

    - **stdout:**

    - **stderr:**

Q7: Writing to file streams: `fprintf`

What if the output of the following code snippet?

```
1:  fprintf(stderr, "CS 241: ");
2:  fprintf(stdout, "System ");
3:  fprintf(stderr, "Programming ");
4:  fprintf(stdout, "\n");
```

⇨ Result:

Q8: What is asprintf()?

```
int asprintf(char **strp, const char *fmt, ...)
```

⇨ **char **strp:**

⇨ **const char *fmt:**

Puzzle 6: Pointer Arithmetic

```
1:   // Count the number of elements in an int-array
2:   // before the number -1 appears in the array:
3:   int count_before(int *array) {
4:     int *ptr = array;
5:
6:
7:     while (*ptr != -1) { ptr++; }
8:
9:
10:    return (ptr – array) / _____;
11:  }
```

Debug Less: Use `assert` e.g. `assert(ptr && counter > 5);`

C provides the library macro assert that be used to find bugs in debugging and completely disappear in production code! Two modes:

- Debug mode (-g flag):

- Production mode (#NDEBUG):

    Best Practice: <u>Always</u> assert pre-conditions and assumptions.

Puzzle 7: Putting it altogether

```
1:   // Sum an array of positive numbers, storing
2:   // the result in `result` (by ref)
3:   // and use asprintf to return a text version of result
4:   char* mysum(const int *ptr,  int *result) {
5:
6:
7:
8:
9:
10:    while ( *ptr ) {
11:
12:
13:      sum += *(ptr++);
14:
15:
16:    }
17:    char *text = NULL;
18:
19:    asprintf(
20:    return text;
21:  }
```