

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 stat in C?

What are the manual sections?

Section 2? 3? 7?

Q2: For the start of the program, `main(int argc, char** argv)`

Sketch the `argv` memory structure

What is special about `argv[0]`

What is special about `argv[argc]`

How do you print out all of the arguments of a program?

```
1: int main(int argc, char**argv)
2:     for(int i=0; i<argc; i++) printf("%s;", argv[i])
3:     //or...?
4:
5:
6:
```

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

- Allocate:
- Free:

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

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

Best Practice: Always set free'd pointers to NULL.

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

Q4 Puzzle 3: Fix a custom string concat (append) 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.
<code>const char *</code>	<code>src</code>	<code>0x1000</code>
<code>char *</code>	<code>dest</code>	<code>0x2000</code>

⇒ **Line 3:** What does `(*src)` do?

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

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

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

Q5. 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 EVERYTHING 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 341: ");
2: fprintf(stdout, "System ");
3: fprintf(stderr, "Programming ");
4: fprintf(stdout, "\n");
```

⇒ Result:

Q8: What is `asprintf()`?

```
int printf(const char * format, ...)
int fprintf(FILE * stream, const char * format, ...)
int sprintf(char * str, const char * format, ...);
int asprintf(char **strp, const char *fmt, ...)
```

⇒ `char **strp`:

⇒ `const char *fmt`:

Q9. Puzzle 6: Pointer Arithmetic

```
1: // Count the number of elements in an int-array
2: // until a number > 100 appears in the array:
3: int count (int *start) {
4:     int *ptr = start;
5:
6:
7:
8:
9:
10:    return _____ / _____;
11: }
```

Q10 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 to add useful debugging info for the debugger):
- Production mode (#NDEBUG):

Best Practice: Always 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:    while ( *ptr ) {
10:
11:
12:        sum += *(ptr++);
13:
14:
15:    }
16:    char *text = NULL;
17:
18:    asprintf(
19:        return text;
20:
21: }
```