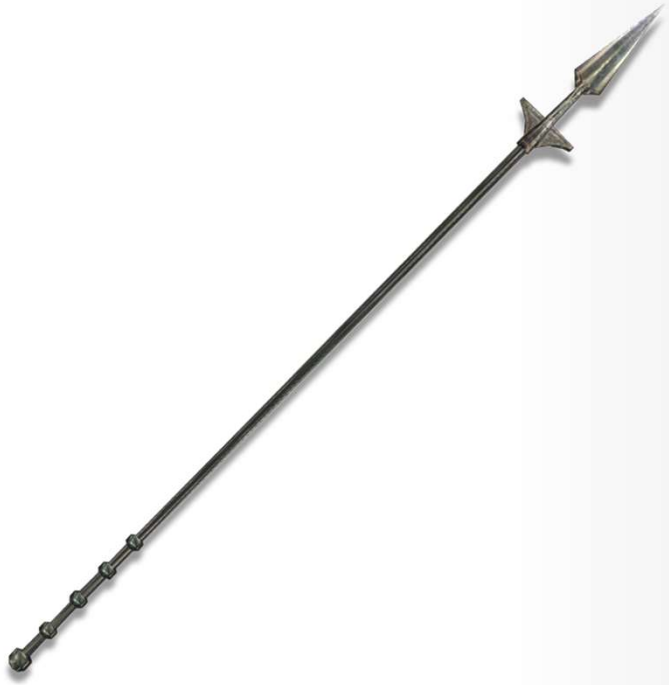
CS221
C and Systems Programming



1

Function pointers

```
char* capitalize(char* input);
char* decorate(char* input);

char* (*funcPointer)(char*) = capitalize;
funcPointer = decorate;
```

What??

funcPointer is a "pointer to a function returning a char*". Need the parens.

Function pointer details

1) Actually call the function by:

```
(*funcPointer)("Input string goes here"); // or…
funcPointer("Input string goes here");
```

2) The types of return value and function inputs are part of the function pointer type

```
int (*fptr1)(int);
int (*fptr2)(double);
float (*fptr3)(int);
```

3) Why to use this? Often as a "callback"…

Do a sample program that codes up capitalize() and decorate(). Call one at random?

Callback: GUI button, event handler, etc.

Function pointer example

```
atexit()
```

Do cleanups, print message, save data, etc when a program exits

In a really big program, you might not know what causes a program to exit

Do a sample program (exitdemo.c)

Another function pointer example

```
qsort()
```

What is
`int (*foo[30])(double)` ?

Look at online man page for qsort()
Do a sample program (sorting.c)

SOMEONE ELSE WROTE qsort() FOR US. We must pass in a helper function.

"Success Patterns"

Common code patterns that work!

```
float myData[45];
for (int n = 0; n < 45; n++) {
 doSomething(myData[n]);
}
```

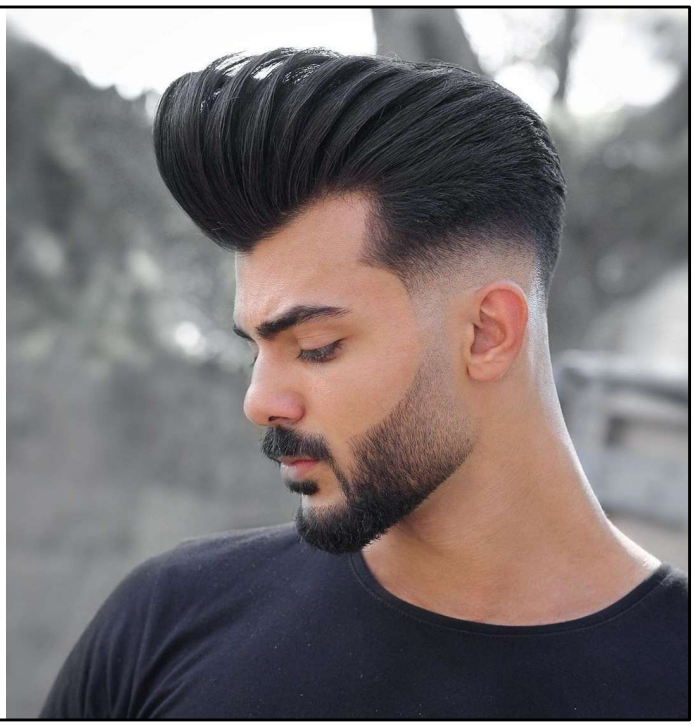
Common code patterns that work!

```
char buf[32];

int retval = scanf("%31s", buf);
while (retval > 0) {
   // handle buf ONLY if read successfully
   CheckIfNameInDatabase(buf);
   UpdateDatabase(buf);

   // read next value at END of loop
   retval = scanf("%31s", buf);
}
```

Good coding style



What do you think?

What do you think?

Descriptive variable names

camelCase

Always use {}, even if only 1 statement in code block

Methods < 50 lines long

Files < 500 lines long

Comment every method and struct

Comment code every 10-15 lines

When adding to an existing project, follow its existing style!

Testing

SW testing

SW professionals spend 1x – 9x more time "testing" their code than writing it

Types of tests

- Unit Tests
- System Tests
- Negative Tests
- Performance Tests
- Regression Tests
- etc

Planning tests, writing test SW, running test SW, etc

My experience…

Think about how to test

Don't assume anything works…till you've tested it

Use SW to test SW

- Repeatable
- Shell scripts
- Continuous Integration

Most basic unit test: **assert.h**

Look at asserts.c, testAssert.sh
"-v" option prints each command before running it