C isn't that hard:

```c
void (*)(*f[])(())();
```

Defines f as an array of unspecified size of pointers to functions that return pointers to functions with a return type of void.

Week 5 Thursday
Generics II

Always has been

Wait it's all void*?

```c
void (*)(*f[])(())();
```
git clone /afs/ir/class/archive/cs/cs107a/cs107a.1226/WWW/exercises/generics2
Announcements

- Midterm next Tuesday
- Next Tuesday’s section (4-5pm, 2 hours before midterm) is optional, but I’ll be in Thornton 110 for office hours
- CS 107A Midterm Review Session next Sunday, 3-5pm, location TBD
  - Practice Exam released later
Unix Tip Spotlight

- None today!
Agenda

- Review: Resizing Arrays
- Function Pointers
- Comparison Functions
- map and filter
Review: Resizing Arrays

https://cplayground.com/?p=quokka-dragon-snake
Your code is data too
Function Pointer Types

- [https://goshdarnfunctionpointers.com/](https://goshdarnfunctionpointers.com/)
- There's a variant of this with a name you'd expect
- [https://cdecl.org/](https://cdecl.org/)
- It can get pretty bad

```c
#include <stdlib.h>

void *(*return_my_function_pointer_arg(void *(*arg)(size_t)))(size_t) {
    return arg;
}

int main() {
    void *(*malloc_ptr)(size_t) = return_my_function_pointer_arg(malloc);
    return 0;
}
```
What gets printed?

```c
int sum(int a, int b) {
    return a+b;
}

int square(int x) {
    return x*x;
}

int apply_the_things(int (*f)(int, int), int (*g)(int), int x, int y) {
    return f(g(x), g(y));
}

int main() {
    printf("%d\n", apply_the_things(sum, square, 2, 3));
    return 0;
}
```
From last section: `void*`

- You can’t do anything with `void*` unless you’re told more information about the type you’re pointing to
  1) Maybe you’re told the size of the type you point to – allows you to use `memcpy` to copy or move the values around
  2) Maybe you’re given a function pointer that you’re told you can use to manipulate values of this mysterious type
Comparison
Functions
Usage of comparison function

```c
int compare_mysterious_value(void *a, void *b);
```

// We know it’s a string!
```c
int compare_mysterious_value(void *a, void *b) {
    char *a_int = *(char**)a;
    char *b_int = *(char**)b;
    return strcmp(a, b);
}
```

// No idea what type it is
```c
void *value1 = get_mysterious_value_ptr();
void *value2 = get_mysterious_value_ptr();
if (compare_mysterious_value(v1, v2) == 0) {
    printf("EQUAL!\n");
}
```

Mysterious Type Owner

User of generic interface
Comparison Functions

- Pointers to these are always of the form
  - `int (*compare_fn)(void *a, void *b);`
- Arguments: pointers to the values being compared
- Return value:
  - Negative if `a` less than `b`
  - 0 if `a` equal to `b`
  - Positive if `a` more than `b`
- The comparison does not have to be based on literal value!
Normal Int Compare Function

```c
int int_compare(void *a, void *b) {
    ???
}
```
Normal Int Compare Function

```c
int int_compare(void *a, void *b) {
    int x = *(int*)a;
    int y = *(int*)b;
    if (x < y) return -1;
    if (x > y) return 1;
    return 0;
}
```
Normal Int Compare Function

```c
int int_compare(void *a, void *b) {
    int x = *(int*)a;
    int y = *(int*)b;
    return x - y;
}
```
Magic Int Compare Function

```c
int int_magic_compare(void *a, void *b) {
    ???
}
```

An int is magic if it is 7, otherwise it isn’t.

All other integers

7

<----------EQUALLY NOT MAGIC-----------------MAGIC------>

The Magic Number Scale
Magic Int Compare Function

```c
int int_magic_compare(void *a, void *b) {
    int x = *(int*)a;
    int y = *(int*)b;
    if (x != 7 && y == 7) return -1;
    if (x == 7 && y != 7) return 1;
    return 0;
}
```

An `int` is magic if it is 7, otherwise it isn’t.

All other integers

7

<---------------EQUALLY NOT MAGIC-----------------MAGIC------->

The Magic Number Scale
map and filter
Hopeful Plan

● We will do `map.c` together
  ○ First implement `map`, and test with `demo_map_for_strings`
  ○ Then finish `demo_map_for_ints`
● `map_soln.c` contains the solution

● You will do `filter.c` on your own
  ○ First implement `filter`, and test with `demo_filter_for_ints`
  ○ Then finish `demo_filter_for_strings`
● `filter_soln.c` contains the solution

● We’ll go over it afterwards