Computer Systems

CS107

Cynthia Lee
Today’s Topics

FUNCTION POINTERS AND CALLBACKS

› (This is a continuation of our theme of making “generic” functions in C)
› New tool:
  • Function pointer mechanism for callbacks
› Common callback utilities:
  • CleanupElemFn
  • CompareFn
Function pointers
Another tool for making “generic” functions in C
First, some context for function pointers

› This is a continuation of our theme of making “generic” functions in C
› So far we’ve seen these tools:
  • Use void* to pass argument of any type
  • Send the size of the item in bytes as a separate argument
  • Use char[] for temporary storage of items of arbitrary size
› Example—our generic swap function:

```c
void swap_any(void *a, void *b, size_t sz) {
    char tmp[sz];
    memcpy(tmp, a, sz);
    memcpy(a, b, sz);
    memcpy(b, tmp, sz);
}
```

› Today: new tool!
  • Function pointer mechanism for callbacks
Finding the max of an array

We will use this as an example of taking an int-specific function and making it generic.
Finding the max of an array: int version

```c
int find_max(int *arr, int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max)
            max = arr[i];
    }
    return max;
}
```

- Key elements:
  - Keep track of max so far in a temporary variable of array’s type (int)
  - Iterate over the array, one bucket at a time
  - Use “<” to compare two int values
  - If current int is larger than max so far then update max
Finding the max of an array: towards a **generic** version

```c
/* this is still int version */
int find_max(int *arr, int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max)
            max = arr[i];
    }
    return max;
}
```

**QUESTION: What needs to change to make this generic?**

- Brainstorm and submit free text response
- List one thing that needs to change per response
Finding the max of an array: towards a generic version

/* this is still int version */
int find_max(int *arr, int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}

- What needs to change to make this generic?
  - No longer know array’s type
  - Can’t use array’s type as our temporary variable
  - Need to know how much to increment to move to next “bucket” of array
  - Can’t use “>” to compare
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n) {
    ...
}
```

---

- Try to fix these one at a time:
  - **No longer know array’s type**
    - Now return a pointer to max element, not the max element itself
  - Can’t use array’s type as our temporary variable
  - Need to know how much to increment to move to next “bucket” of array
  - Can’t use “>” to compare
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n) {
    void *max = arr;
    ...
}
```

- Try to fix these one at a time:
  - No longer know array’s type
  - **Can’t use array’s type as our temporary variable**
    - Now store a pointer to the current max element, not the current max element itself
  - Need to know how much to increment to move to next “bucket” of array
  - Can’t use “>” to compare

```c
/* int version, for reference */
int find_max(int *arr, int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n, size_t sz) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        ...
    }
}
```

- Try to fix these one at a time:
  - No longer know array’s type
  - Can’t use array’s type as our temporary variable
  - **Need to know how much to increment to move to next “bucket” of array**
    - Add a size argument and use it to advance the pointer
  - Can’t use “>” to compare

```c
/* int version, for reference */
int find_max(int *arr, int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n, size_t sz) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        ...
    }
}
```

- Try to fix these one at a time:
  - No longer know array’s type
  - Can’t use array’s type as our temporary variable
  - **Need to know how much to increment to move to next “bucket” of array**
    - Add a size argument and use it to advance the pointer
  - Can’t use “>” to compare

**QUESTION:** Why are we casting `arr` to type `char*`?

- A. Because `arr` is an array of chars
- B. `char` is part of the `int` family of types (ASCII code means that chars are just numbers), so `>` will work
- C. For pointer math
- D. Help! I’m lost!
- E. Other
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n, size_t sz) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        if (cmp(ith, max) > 0)
            max = ith;
    }
    return max;
}
```

- Try to fix these one at a time:
  - No longer know array’s type
  - Can’t use array’s type as our temporary variable
  - Need to know how much to increment to move to next “bucket” of array
  - **Can’t use “>” to compare**
    - Assume we have a function that compares—**but how??**

Where does this “cmp” function come from? **THE CALLER must provide**—they’re the only ones who can know how to compare appropriately.
Callback functions

- There is no way that our `find_max_any` function can know how to compare two `<something>`s without knowing anything about what that `<something>` is.

- `find_max_any` must ask the function that called it (the “caller”) for help!

- **When it comes time for that help, use a function the caller specified**
  - “**callback function**” is a function provided by the caller to do a specific that the callee needs the caller’s help with.
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n, size_t sz,
                   int (*cmp)(const void *, const void *)) {
  void *max = arr;
  for (int i = 1; i < n; i++) {
    void *ith = (char *)arr + i * sz;
    if (cmp(ith, max) > 0)
      max = ith;
  }
  return max;
}
```

- Try to fix these one at a time:
  - No longer know array’s type
  - Can’t use array’s type as our temporary variable
  - Need to know how much to increment to move to next “bucket” of array
  - **Can’t use “>” to compare**
    - Caller provides us with a function to compare as an argument
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n, size_t sz,
                   int (*)(const void *, const void *)) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        if (cmp(ith, max) > 0)
            max = ith;
    }
    return max;
}
```

- Try to fix these one at a time:
  - No longer know array’s type
  - Can’t use array’s type as our temporary variable
  - Need to know how much to increment to move to next “bucket” of array
  - **Can’t use “>” to compare**
    - Caller provides us with a function to compare as an argument
Finding the max of an array: generic version

```c
void *find_max_any(void *arr, int n, size_t sz,
    int (*cmp)(const void *, const void *)) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        if (cmp(ith, max) > 0)
            max = ith;
    }
    return max;
}
```

- Try to fix these one at a time:
  - No longer know array’s type
  - Can’t use array’s type as our temporary variable
  - Need to know how much to increment to move to next “bucket” of array
  - Can’t use “>” to compare

• find_max_any is saying that it wants you to give it a function that returns int. It won’t accept a function that returns anything other than int, because it wouldn’t be able to use that.
• Notice where we use the compare function, it says “if (cmp(ith, max) > 0)”, so it’s expecting the function to return an int to compare to 0.
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n, size_t sz,
                    int (*cmp)(const void *, const void *)) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        if (cmp(ith, max) > 0) max = ith;
    }
    return max;
}
```

- Try to fix these one at a time:
  - No longer know array’s type
  - Can’t use array’s type as our temporary variable
  - Need to know how much to increment to move to next “bucket” of array
  - Can’t use “>” to compare

- Caller provides us with a function to compare as an argument

  - The second part is the **name** of the function—“cmp” for “compare.”

**FAQ:**

- Why is the name in parentheses with an asterisk? That’s kind of a “just memorize that’s how to do it” thing, but one way to think of it is that it’s a function pointer, hence the *, and the () just keeps the * and the function name bundled together.

- Does that have to be the actual name of the function we pass in? NO. Just like the caller’s array could have a name other than “arr,” but “arr” is just what we call that array inside find_max_any.
Finding the max of an array: **generic** version

```c
void *find_max_any(void *arr, int n, size_t sz,  
    int (*cmp)(const void *, const void *)) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        if (cmp(ith, max) > 0)
            max = ith;
    }
    return max;
}
```

The third part is the **parameter list** of the function.

- It’s in parentheses and comma separated, just like we’re used to for function parameter lists—easy!
- `find_max_any` is saying that it wants you to give it a function that takes two arguments, and they both are type `const void *`.  
- Notice where we use the compare function, it says “if `cmp(ith, max) > 0`”, so it’s expecting the function to take two arguments. Both `ith` and `max` are type `void *`.  
- Try to fix these one at a time:  
  - No longer know array’s type  
  - Can’t use array’s type as our temporary variable  
  - Need to know how much to increment to move to next “bucket” of array  
  - Can’t use `>` to compare  
  - Caller provides us with a function to compare as an argument
Putting it together with a sample `main`

// File: myfile.c
int compare_int(const void *a, const void *b) {
    int one = *(int*)a;
    int two = *(int*)b;
    return one - two;
}

int main(int argc, char *argv[]) {
    int nums[] = {40, 99, 23, 45, 12, 45, 23, 59, 33, 92};
    int *ptr_to_max = (int*)find_max_any(nums, 10, sizeof(int), compare_int);
    printf("%d\n", *ptr_to_max);
    return 0;
}

-------------------------------------------------------------------------

// File: generic_max.c
void *find_max_any(void *arr, int n, size_t sz, int (*cmp)(const void *, const void *)) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        if (cmp(ith, max) > 0)
            max = ith;
    }
    return max;
}
Now imagine some (accidental?) misuse of generics...

// File: myfile.c
int compare_int(const void *a, const void *b) {
    int one = *(int*)a;
    int two = *(int*)b;
    return one - two;
}

int main(int argc, char *argv[]) {
    int nums[] = {40, 99, 23, 45, 12, 45, 23, 59, 33, 92};
    int *ptr_to_max = (int*)find_max_any(nums, 10, sizeof(double), compare_int);
    printf("%d\n", *ptr_to_max);
    return 0;
}

// File: generic_max.c
void *find_max_any(void *arr, int n, size_t sz,
    int (*cmp)(const void *, const void *)) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        if (cmp(ith, max) > 0)
            max = ith;
    }
    return max;
}
Now imagine some (accidental?) misuse of generics...

```c
// File: myfile.c
int compare_int(const void *a, const void *b) {
    int one = *(int*)a;
    int two = *(int*)b;
    return one - two;
}

int main(int argc, char *argv[]) {
    int nums[] = {40, 99, 23, 45, 12, 45, 23, 59, 33, 92};
    int *ptr_to_max = (int*)find_max_any(nums, 10, sizeof(double), compare_int);
    printf("%d\n", *ptr_to_max);
    return 0;
}
```

// File: generic_max.c
```c
void *find_max_any(void *arr, int n, size_t sz,
    int (*cmp)(const void *, const void *)) {
    void *max = arr;
    for (int i = 1; i < n; i++) {
        void *ith = (char *)arr + i * sz;
        if (cmp(ith, max) > 0)
            max = ith;
    }
    return max;
}
```

PREDICT: What is printed?
(a) Nothing—compiler error
(b) Segmentation fault/crash
(c) 99
(d) 40
(e) Unpredictable /other
Common utility callback functions

You’ll see these in lab/assignments
Common utility callback functions

- **Compare function**

  ```c
  int (*cmp)(const void *, const void *)
  ```

  - As you’ve seen, compares to elements in a “less than”-equivalent way, specific to the type
  - Here we used it for `find_max_any`, but it’s also used in generic versions of popular algorithms like binary search and quicksort

- **Cleanup function**

  ```c
  void (*CleanupElemFn)(void *addr);
  ```

  - This one is useful for “cleaning up” (calling `free` on) elements of an array that might point to heap memory, but only the caller can know if they do and exactly how they do
  - This essentially allows for a “deep” free by the callee, by relying on the caller to handle all the deep stuff

Stanford University