

Computer Systems

CS107

Cynthia Lee

Today's Topics

LECTURE:

- › More on building generics with void*
- › Function pointers and callbacks
- › Callbacks in assign2 (CleanupElemFn, CompareFn)

- Next time:
 - › Binary!



More on building generics with void*

Make some more "swap" functions

```
> void swap_str(char **a, char **b) {  
    char *tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
> void swap_int(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
> void swap_float(float *a, float *b) {  
    float tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
> void swap_double(double *a, double *b) {  
    double tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

We're glad you're here.\0

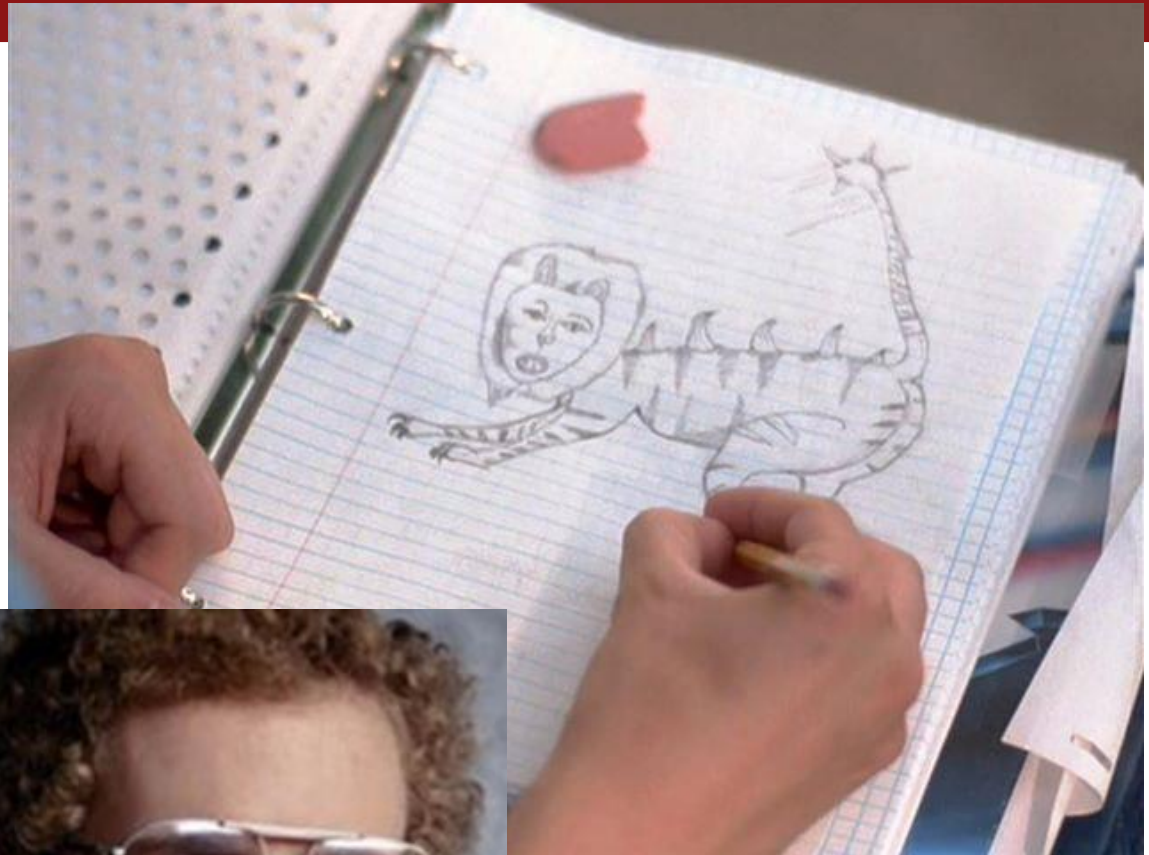
You can do it!\0

| |
|---------|
| 7.22 |
| 0x0FFF8 |
| 28.33 |
| 5 |
| 0x0FF00 |
| 4 |
| 1 |
| 8 |

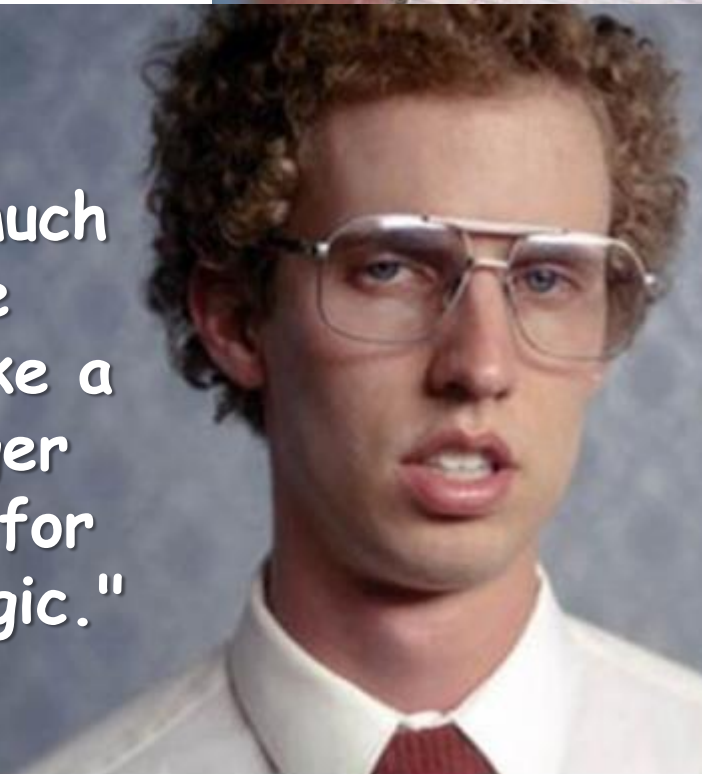
Stan

How NOT to use void pointers

**Let's make a
Liger!**



"It's pretty much
my favorite
animal. It's like a
lion and a tiger
mixed...bred for
its skills in magic."



How can we (mis-)use our swap to write the string "liger" at mem addr 0x0FF00?

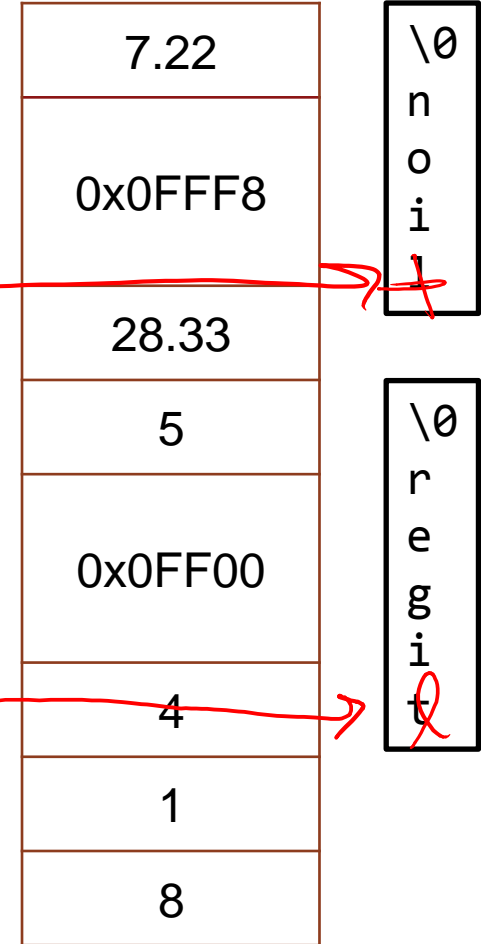
- Generic swap function:

```
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);
}
```

```
char *str1 = strdup("tiger");
char *str2 = strdup("lion");
```

- A. swap_any(&str1, &str2, sizeof(char*));
- B. swap_any(&str1, &str2, sizeof(char));
- C. swap_any(str1, str2, sizeof(char*));
- D. swap_any(str1, str2, sizeof(char));
- E. Other

Stack Heap



Even more horrifying hybrids!

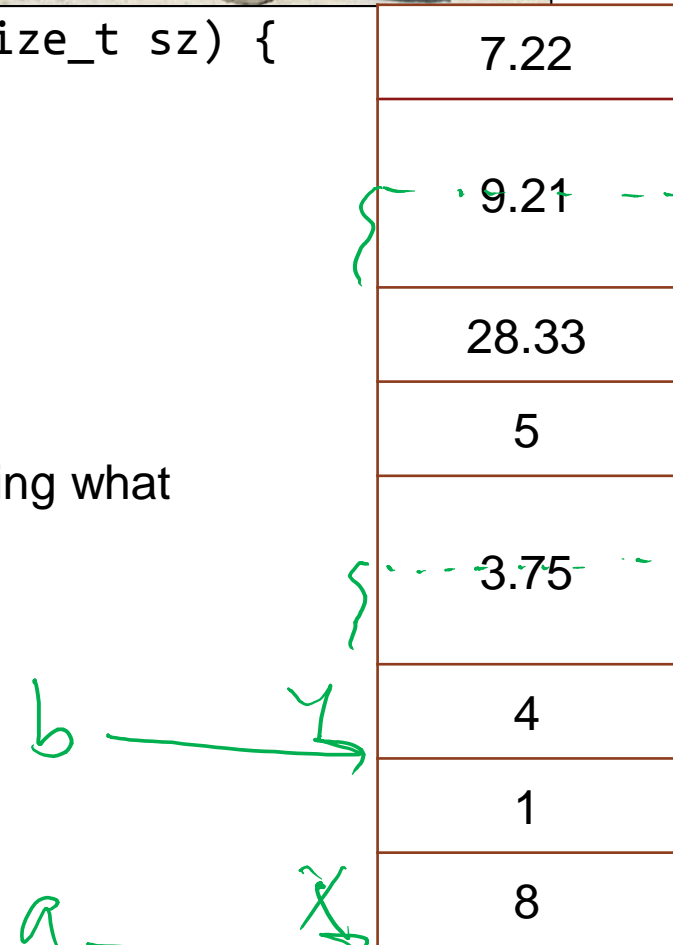


- Generic swap function:

```
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);  
}
```

- See if you can draw memory diagrams showing what happens in these examples of **mis-**usage:

```
int x = 8, y = 4;  
swap_any(&x, &y, sizeof(char));  
double dx = 3.75, dy = 9.21;  
swap_any(&dx, &dy, sizeof(int));  
int a = 8, b = 1;  
swap_any(&a, &b, sizeof(double));
```



Function pointers

Finding the max of an array: integer 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;
}
```

- Key elements:
 - › Keep track of max so far in a temporary variable of array's type (int)
 - › Iterate over the array, one int 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

```
/* 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?**

- › 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

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

```
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;  
}
```

- 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

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

```
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;  
}
```

- 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

Finding the max of an array: generic version

```
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;  
  
        ...  
    }  
}
```

```
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;  
}
```

- 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

Finding the max of an array: generic version

```
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;
}
```

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

- 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??**

Callback functions

- There is no way that our `find_max_any` function can know how to compare two `<something>`s without know anything about what that `<something>` is
- **When it comes time for that, use a function the caller specified**
 - › **“callback function” is a function provided by the caller to do a specific for you in the middle of doing your job**
- Like giving your baby to a sitter—sitter takes care entertaining the baby as much as they can, but there are some things (inconsolable crying, dirty diaper) that sitter can't handle and they must let you (the parent) handle
- **Timeline:**
 1. You give your baby (the data) and your phone number (the “callback”) to a sitter
 2. Sitter plays with baby for a while—fun!
 3. Sitter reaches a point where they need the you (the parent), so they call the phone number you provided and *temporarily* give the baby back to you to handle the issue
 4. After you handle the issue, you return baby back to sitter for more playtime—fun!
 5. After a full day of playtime, sitter returns baby back to you for good



Bringing the analogy back to code

```
int main(int argc, char *argv[]) { //main = you, the parent
    int jonas = 1;
    babysit(&jonas, our_phone_num); //1. give baby and phone number to sitter
    //5. day is over and babysitter has now returned to parents for good
    return 0;
}
```

```
void babysit(void *baby, void (*emergency_phone_num)(void*)) {
    //2. sitter plays with baby for a while--fun!
    printf("peek-a-boo!\n");
    if (baby_is_crying()) { //3. sitter reaches a point where they need parent
        emergency_phone_num(baby); //so call and temporarily return baby to you...
    }
    //4. parent intervention is done, more playtime--fun!
    printf("peek-a-boo!\n");
}
```

```
void our_phone_num(void *baby) { //you, the parent, provide this
    //3. (continued) so you can do something that relies on unique parent
    //knowledge of child that babysitter doesn't have
    *((int*)baby) = 2; //sitter doesn't know baby is actually an int!
}
```

cast based on personal knowledge of how being used

Returning to our int max example

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

```
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;  
}
```

```
int cmp_int(const void *a, const void *b) {  
    int one = *(int *)a, two = *(int *)b;  
    return one - two;  
}
```

Generic functions/callbacks: USE WISELY!!

```
/* Now try to imagine bad (hybrid-like) usage... */
```

```
int main(int argc, char *argv[]) {  
    int nums[] = {40, 99, 23, 45, 12, 45, 23, 59, 33, 92};  
    printf("%d\n", *(int*)find_max_any(nums, 5, sizeof(double), cmp_int));  
}
```

```
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;  
}
```

```
int cmp_int(const void *a, const void *b) {  
    int one = *(int *)a, two = *(int *)b;  
    return one - two;  
}
```

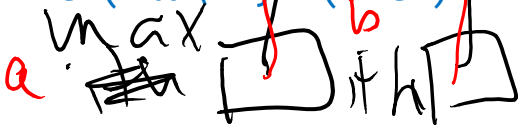
bear shark octopus



Generic functions/callbacks: USE WISELY!!

```
/* Now try to imagine bad (hybrid-like) usage... */
```

```
int main(int argc, char *argv[]) {  
    int nums[] = {40, 99, 23, 45, 12, 45, 23, 59, 33, 92};  
    printf("%d\n", *(int*)find_max_any(nums, 5, sizeof(double), cmp_int));  
}
```



```
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;  
}
```

```
int cmp_int(const void *a, const void *b) {  
    int one = *(int *)a, two = *(int *)b;  
    return one - two;  
}
```

PREDICT: What is printed?

- (a) Nothing—compiler error
- (b) Nothing—crashes
- (c) 99
- (d) 40
- (e) Unpredictable /other

Callbacks specific to CVector and CMap

ASSIGN2 GETTING STARTED