Computer Systems

CS107

Cynthia Lee

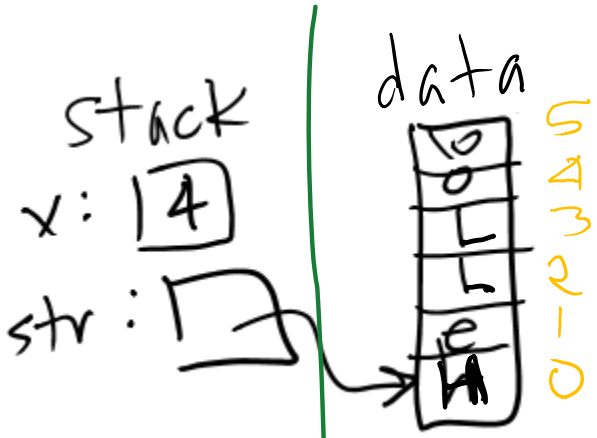
Today's Topics

- Pointers to pointers!
- void pointers and generic functions
- Callbacks (cue Carly Rae Jepson ♫…)

Strings in C: passing them as arguments

Version 1: edits in place; does NOT work if argument is string literal

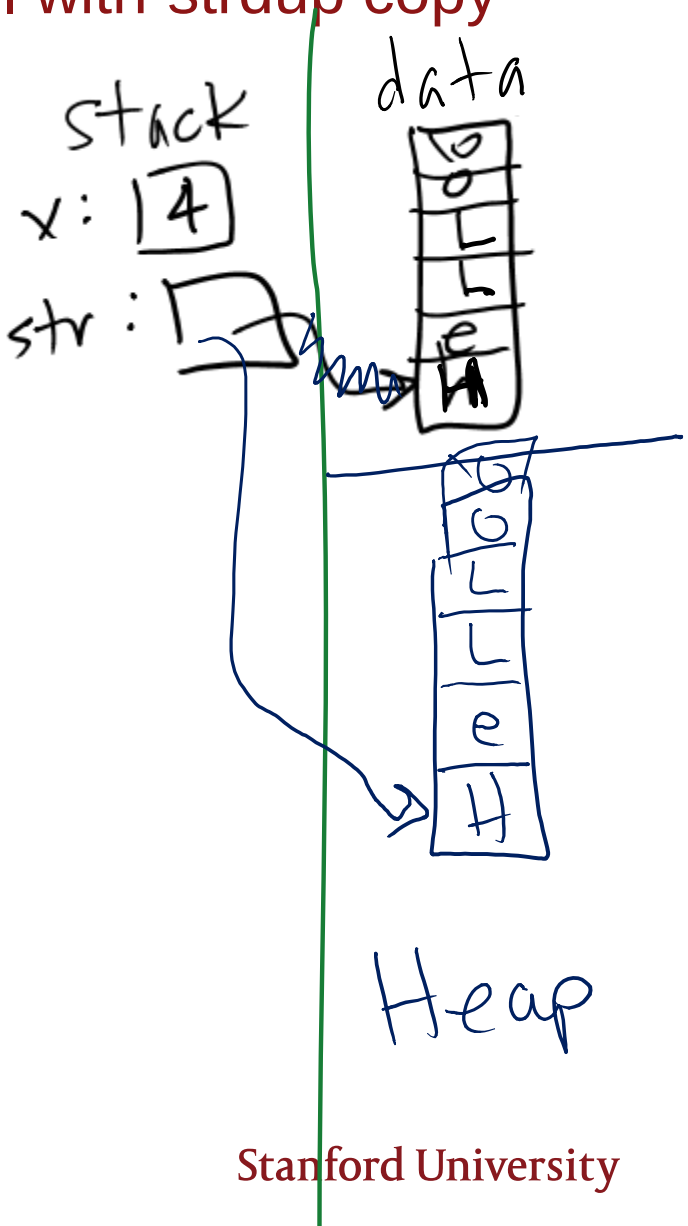
```
void lowercase(char *str) {
    for (int i=0; str[i] != '\0'; i++){
        str[i] = tolower(str[i]);
    }
}

int main(int argc, char *argv[]) {
    int x = 4;
    char *str = "HeLLo";
    lowercase(str);
    printf("%s\n", str);
    return 0;
}
```



- **What does memory look like?**

Version 1.1 (client workaround): only call with strdup copy

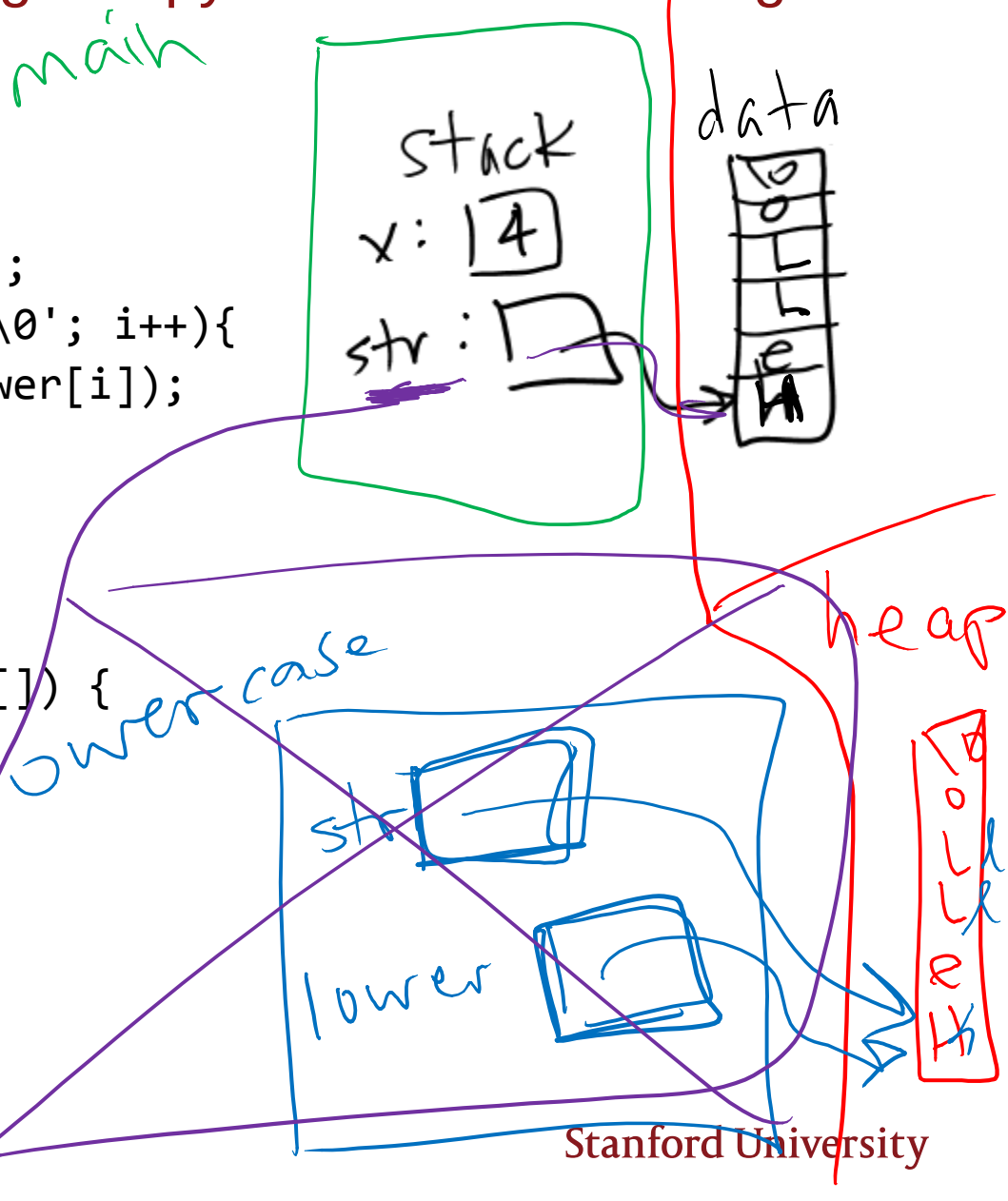
```
void lowercase(char *str) {
    for (int i=0; str[i] != '\0'; i++){
        str[i] = tolower(str[i]);
    }
}

int main(int argc, char *argv[]) {
    int x = 4;
    char *str = strdup("HeLLo");
    lowercase(str);
    printf("%s\n", str);
    return 0;
}
```



- **What does memory look like?**

Version 2: attempt at making a copy so it works for string literals too

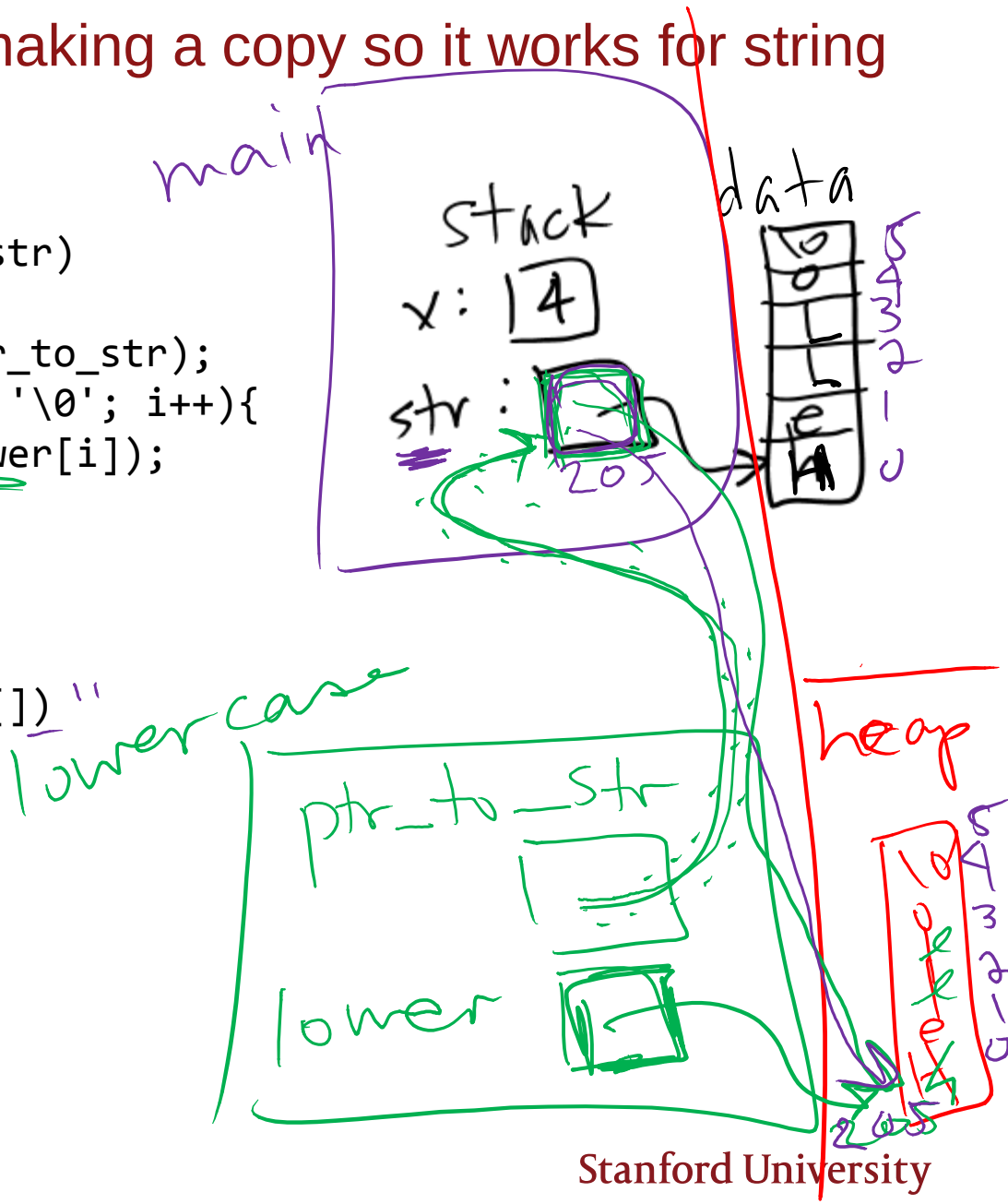
```
void lowercase(char *str) {
    char * lower = strdup(str);
    for (int i=0; str[i] != '\0'; i++){
        lower[i] = tolower(lower[i]);
    }
    str = lower;
}

int main(int argc, char *argv[]) {
    int x = 4;
    char *str = "HeLLo";
    lowercase(str);
    printf("%s\n", str);
}
```



- **What does memory look like?**

Version 3: 2nd attempt at making a copy so it works for string literals too

```
void lowercase(char **ptr_to_str)
{
    char * lower = strdup(*ptr_to_str);
    for (int i=0; lower[i] != '\0'; i++){
        lower[i] = tolower(lower[i]);
    }
    *ptr_to_str = lower;
}

int main(int argc, char *argv[])
{
    int x = 4;
    char *str = "HeLLo";
    lowercase(&str);
    printf("%s\n", str);
    free(str);
}
```



- **What does memory look like?**

Passing strings as arguments: key points

- You may alter the *contents* of a char* argument, but not redirect the pointer
  - For example, if you want to lengthen the string, you're out of luck with char*
- If you want to be able to redirect the pointer, add a level of indirection that gives you access to the char* pointer itself: char**

```
void lowercase(char **str) {
```

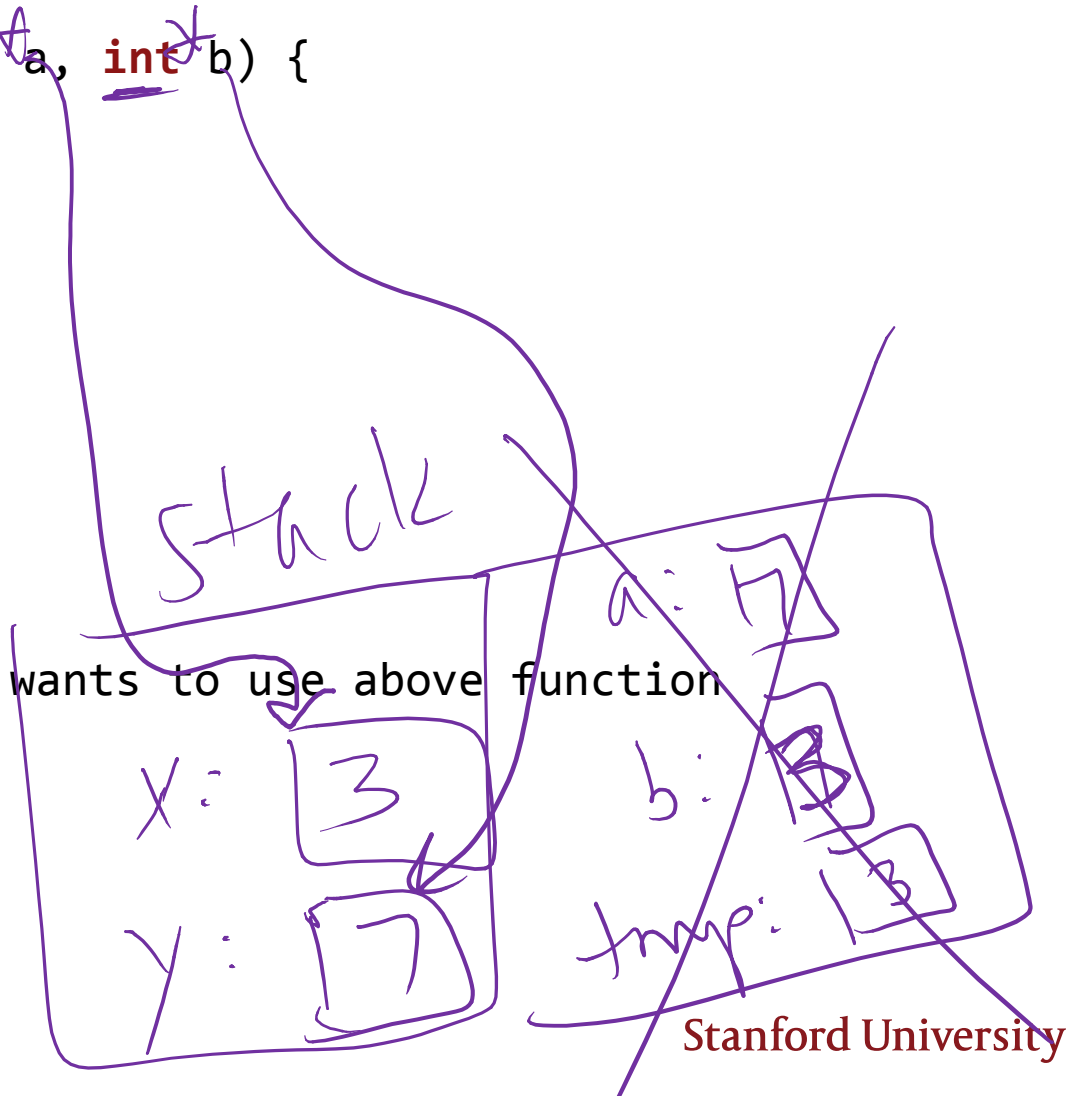
- Of course, returning a char* is another way to get the pointer to the new string back to the caller function, but you'll want to understand the principles behind the char** solution (in case your function needs to return something else too, and for other scenarios)

```
char* lowercase(char *str) {
```

Generic swap with void pointers

Swapping two integers – will this work?

```
› void swap_int(int a, int b) {
      int tmp = a;
      a = b;
      b = tmp;
  }

  …
  //some code that wants to use above function
  int x = 3;
  int y = 7;
  swap_int(x, y);
```



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;
}
```

CS106 SL says, “V- style grade: too repetitive!”

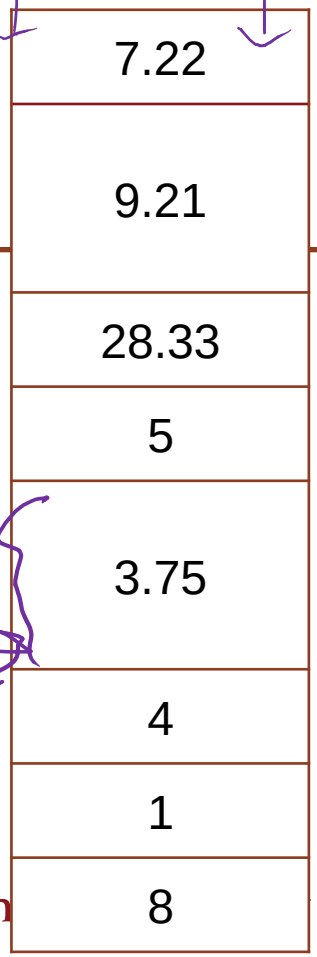
(not to mention nondescript variable names a,b)

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;
}
```

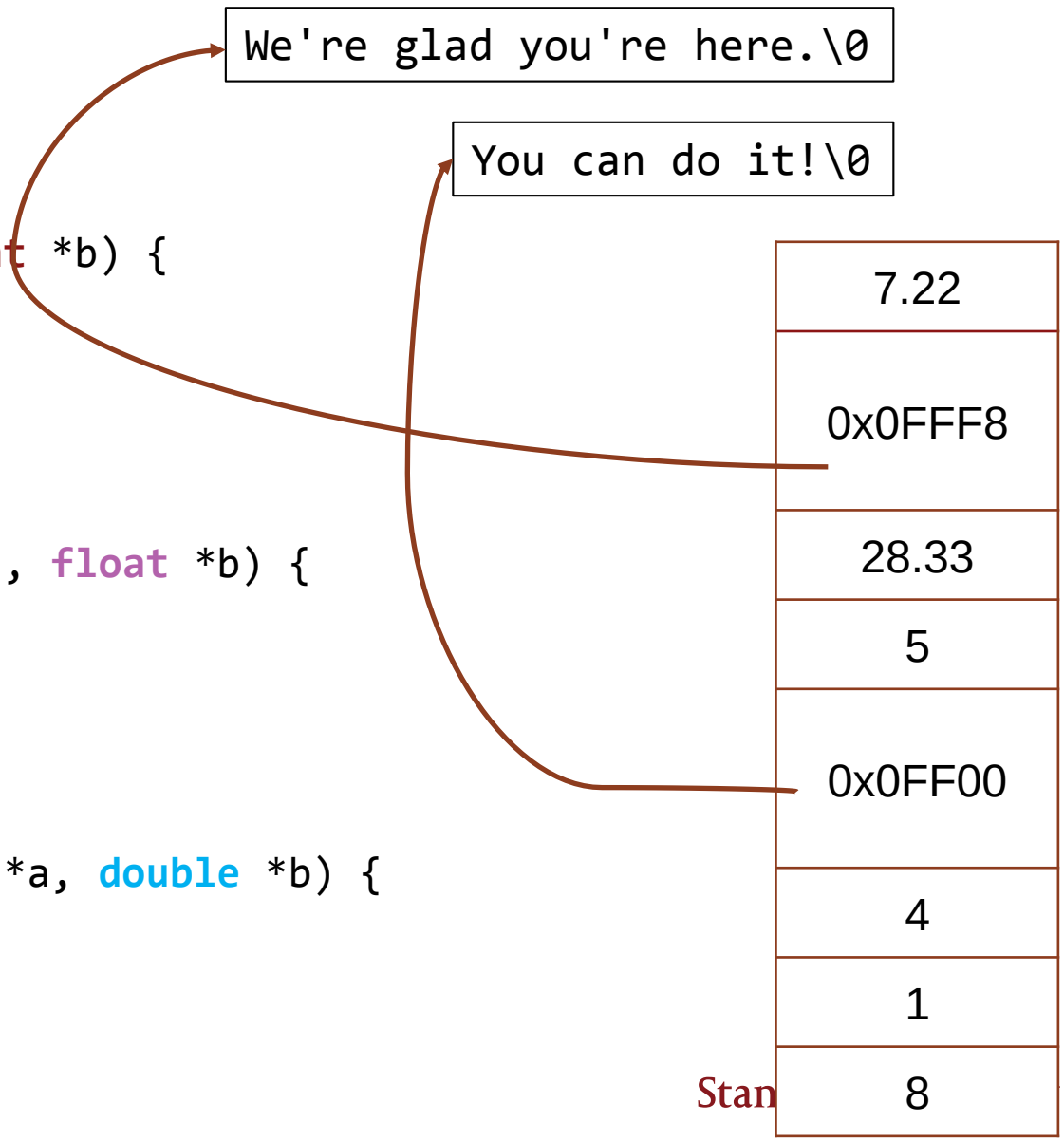
What are some issues with this attempt?

```
void swap_any(void *a, void *b)
{
    void tmp = *a;
    *a = *b;
    *b = tmp;
}
```



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;
  }
```



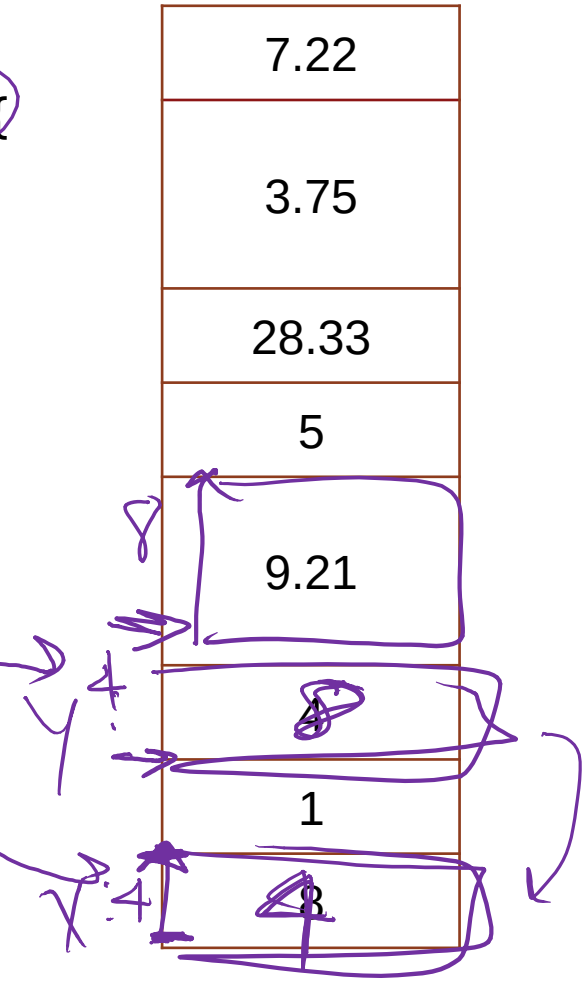
Making an all-purpose "swap" function

- Generic swap function:

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

- Example usage:

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



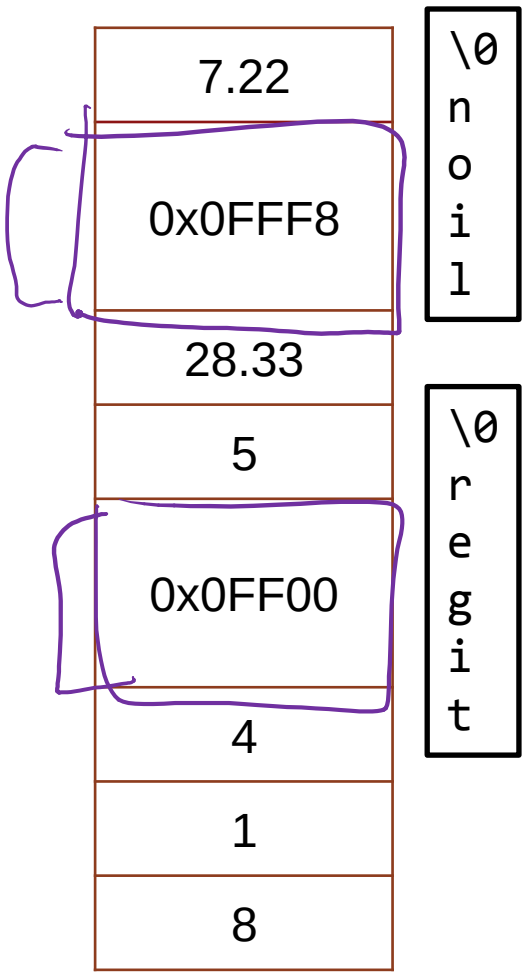
Making an all-purpose "swap" function

- Generic swap function:

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

- Example usage:

```
// double dx = 3.75, dy = 9.21;
// swap_any(&dx, &dy, sizeof(double));
char *str1 = strdup("tiger");
char *str2 = strdup("lion");
swap_any(&str1, &str2, sizeof(char*));
free(str1);
free(str2);
```



Callbacks and function pointers

(QUICKLY NOW, YOU'LL DO MORE IN LAB—NEED THIS FOR ASSIGN2)

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;
}
```

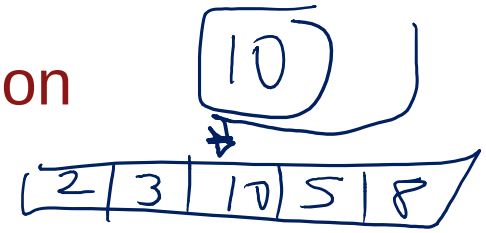
- Key elements:
  - Keep track of max so far in a temporary variable of array's type (int)
  - Iterate over the array, one int at a time
  - Use "<" to compare two int values
  - If current int is larger than max so far then update max

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

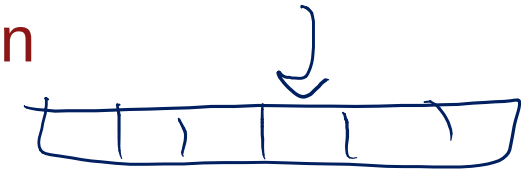



```
void *find_max_any(void *arr, int n) {
    ...
}
```

```
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!
}
```

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;
}
```

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 (segment-fault)
(c) 99
(d) 40
(e) Unpredictable /other