

# Computer Systems

CS107

Cynthia Lee

## Today's Topics

- Pointers to pointers!
- void pointers and generic functions
- Callbacks (cue Carly Rae Jepsen 🎵...)

# Strings in C: passing them as arguments

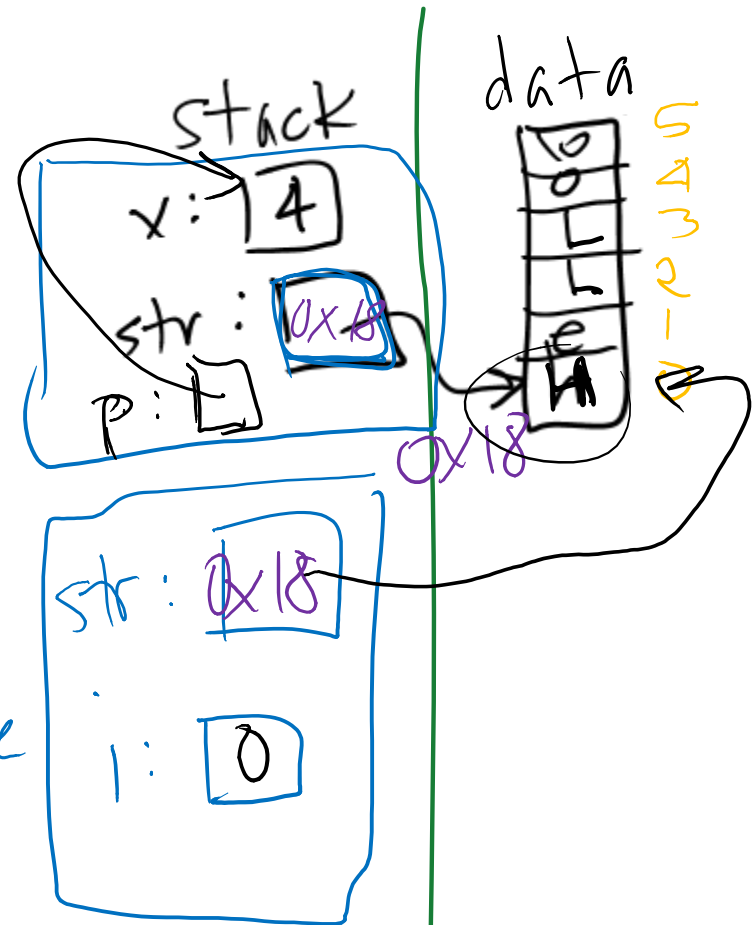
Version 1: edits in place; does NOT work if argument is string literal

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

main

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

lowercase



▪ 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]);  
    }  
}
```

must be  
editable

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

stack

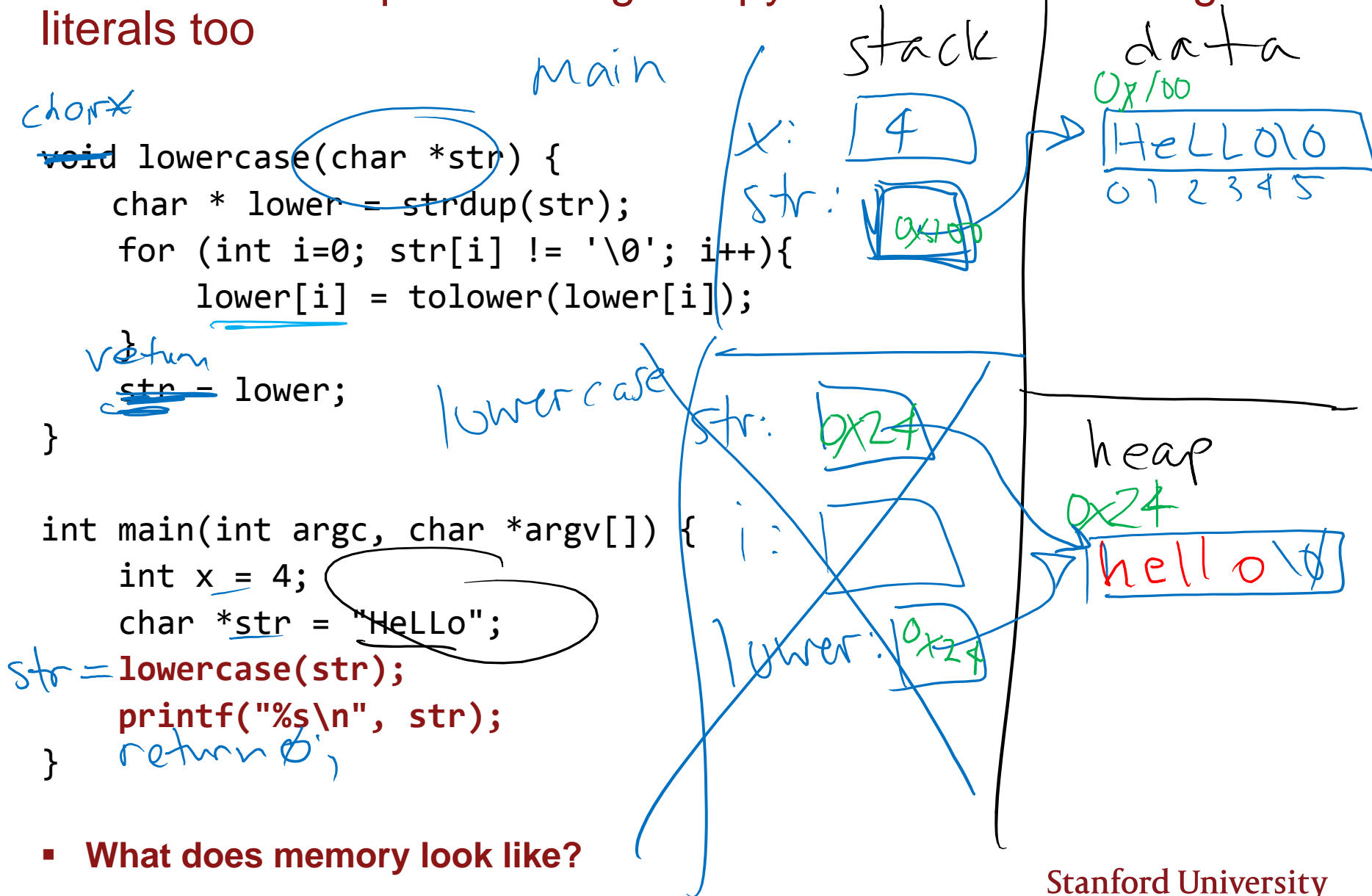
data  
0 1 2 3 4 5  
H e L L O \0

heap

h e l l o \0

- What does memory look like?

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

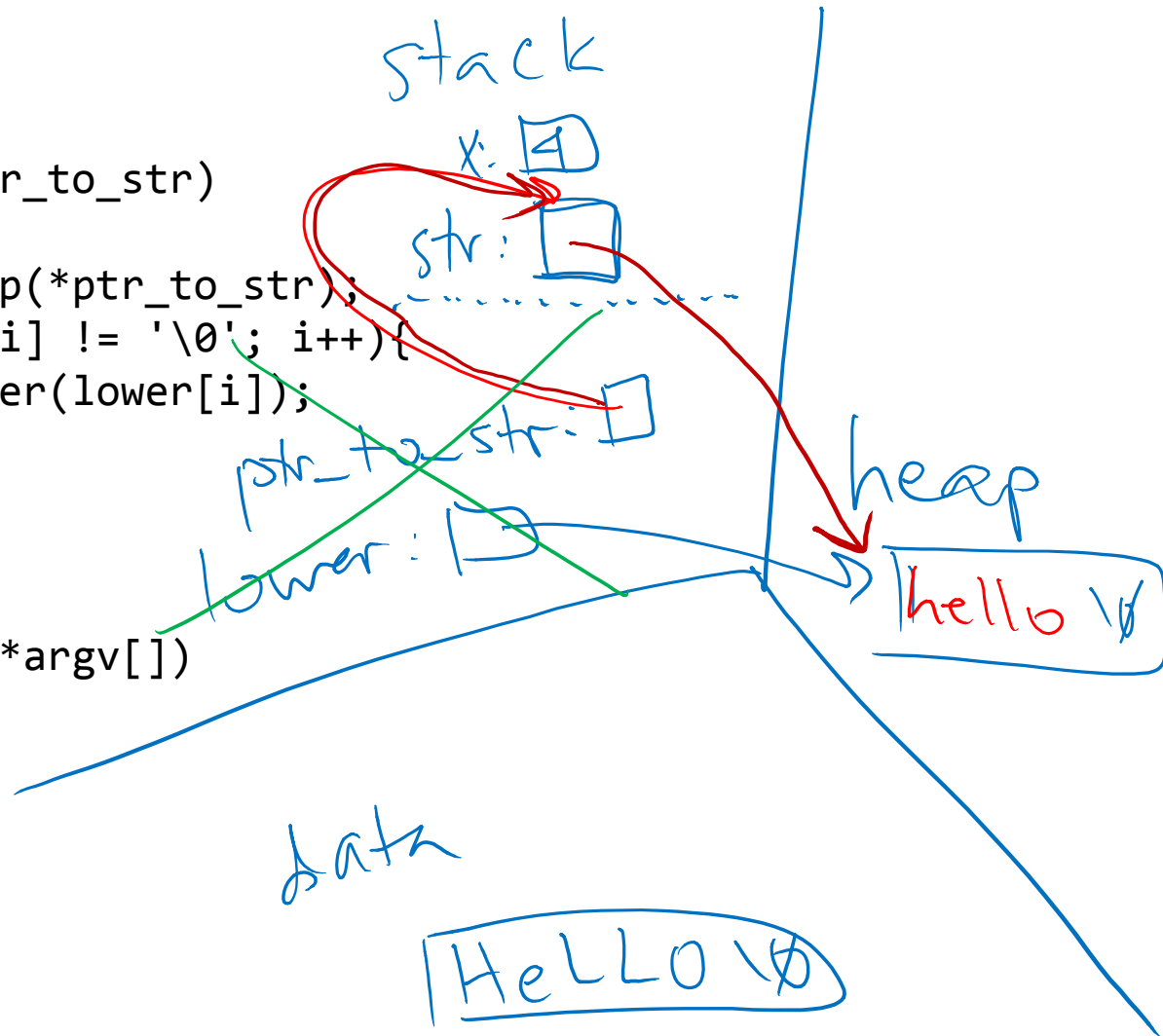


- What does memory look like?

## Version 3: 2<sup>nd</sup> 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);
    return 0;
}
```



- 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;  
}
```

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

```
> void swap_float(float *a, float *b) {  
    float tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

(not to mention nondescript variable names a,b)

```
> void swap_double(double *a, double *b) {  
    double 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;  
}
```

We're glad you're here.\0

You can do it!\0

7.22
0x0FFF8
28.33
5
0x0FF00
4
1
8

Stan

# 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;
}
```

b: [ ]

a: [ ]

7.22
9.21
28.33
5
3.75
4
1
8

Stan

# Making an all-purpose “swap” function

- 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);  
}
```

*/\* why char? \*/*

↑  
2 ?

- 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));
```

	7.22
	3.75
	28.33
	5
	9.21
y:	<del>4</del> 8
	1
x:	<del>8</del> 4

## 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);
```

Stack      Heap

7.22	\0 n o i l
0x0FFF8	
28.33	\0 r e g i t
5	
0x0FF00	
4	
1	
8	

# 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;  
}
```

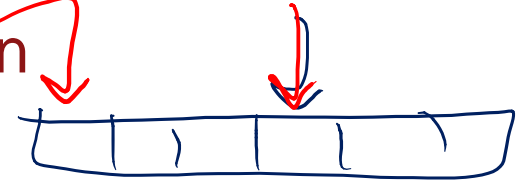
- Try to fix these one at a time:
  - › **No longer know array's type**

- 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??

## Function pointers and callbacks

```
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.

- C has a mechanism for passing a function as a pointer!
- The caller will provide us with a compare function
- This is the essence of the “callback” as used with C generic functions