RECITATION 8

INFO

- Jon Rutkauskas
- Recitation: Tue 12-12:50
- Office Hours: Tue 11-11:50
  Thur 11-12:50
  SENSQ 5806
  (additional hours by appointment if needed)

- On discord: @jrutkauskas
- By email: jsr68@pitt.edu
- Website:
  https://github.com/jrutkauskas/spring2019-449-rec
- Ask me any questions you have!!!

EXAM 1

- So, all of you have taken it now :-)
- I haven't seen it or the key, but I officially can answer any questions you may have.

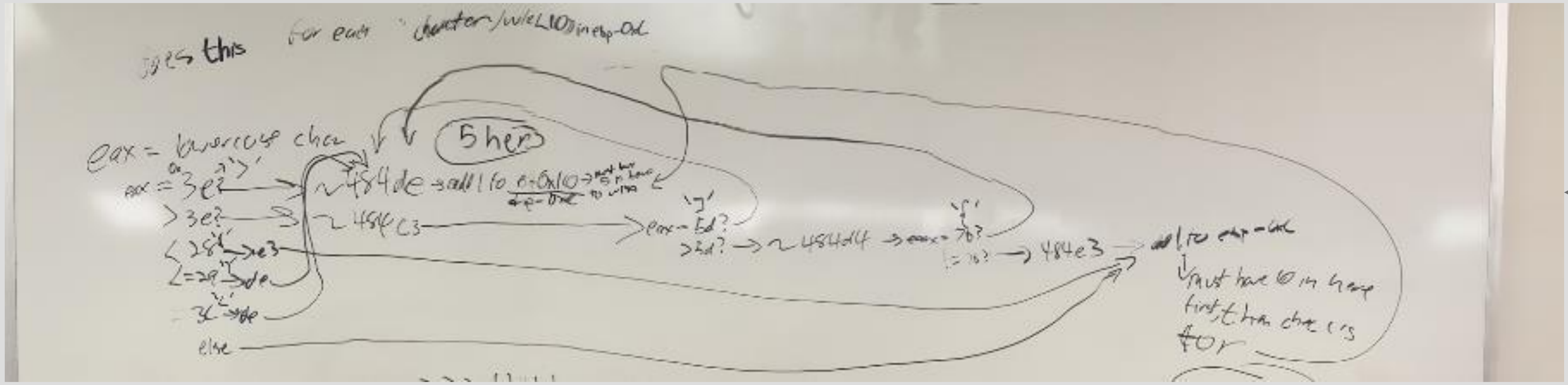
WARMUP POLL

PROJECT 3

- Project 3 is out, and it's lots of fun.
- Start with mystrings. It's relatively straightforward, and kind-of easy points for this project.
- Then, you have your executables to crack
- Use your mystrings, your gdb skills, your x86 skills, your Google skills, etc. to help
- https://jarrettbillingsley.github.io/teaching/classes/cs0449/projects/proj3_details.html ← MAJOR HELP RIGHT HERE!

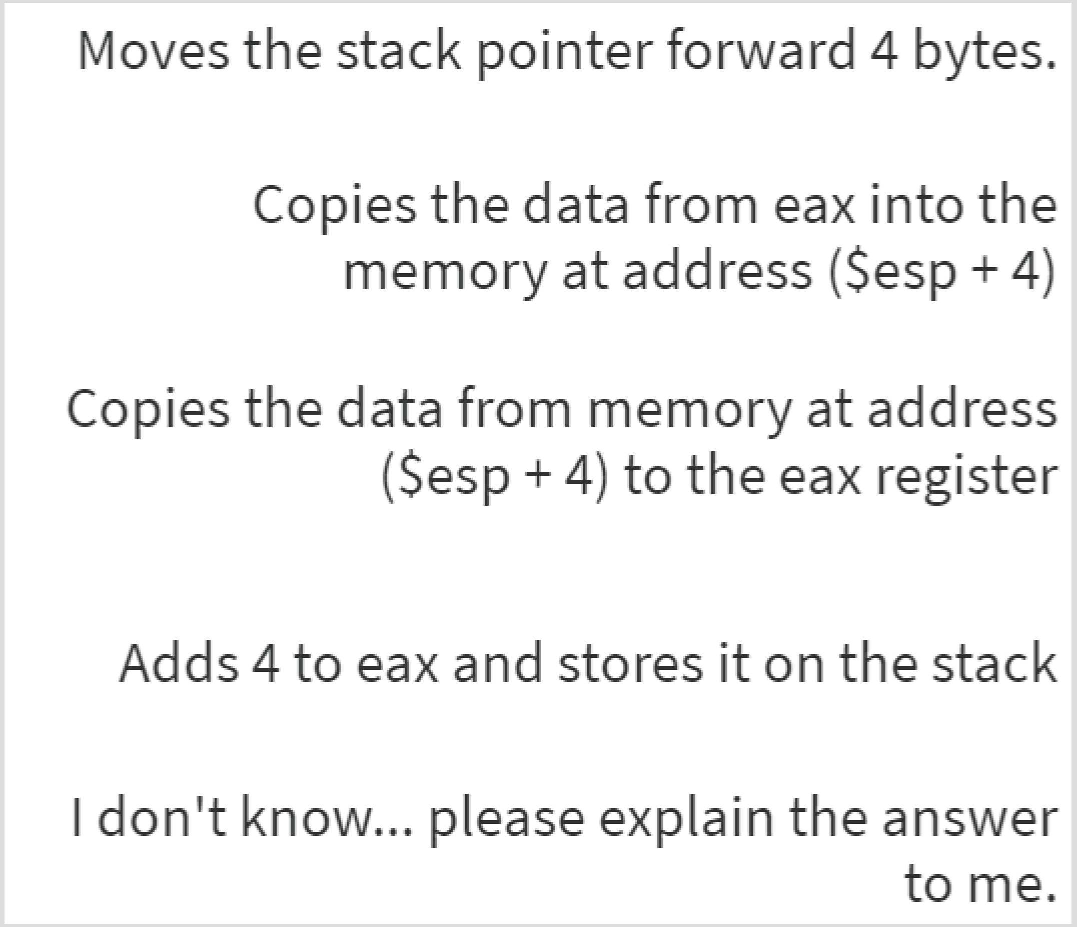
Instead, learn the patterns by writing your own little test programs with if-elses, loops, and switches. Compile them **with the -m32 and -g flags** and inspect their disassembly. Once you see what the compiler will produce for each kind of control structure, you'll start to notice the patterns and be able to think "oh, this is an if-else."

- This tidbit of info from the project description? You probably shouldn't ignore it.
- x86 looks so messy and complicated, it's good to be able to see something familiar.
- Also possibly recommended: Go get a whiteboard or large piece of paper and just draw out the control structure



My old whiteboard scribbles from this project

What does `mov [esp+4], eax` do?



Moves the stack pointer forward 4 bytes.

Copies the data from eax into the memory at address ($esp + 4)

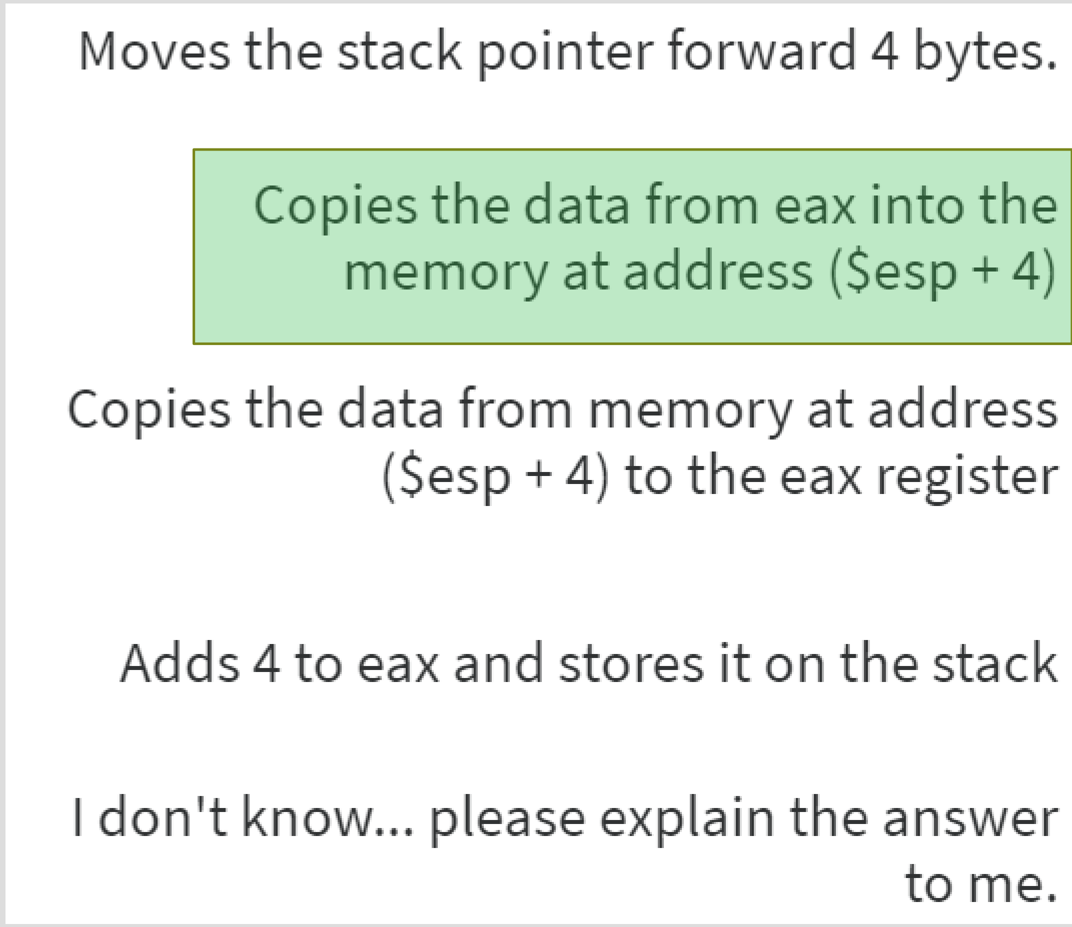
Copies the data from memory at address ($esp + 4) to the eax register

Adds 4 to eax and stores it on the stack

I don't know... please explain the answer to me.

What does `mov [esp+4], eax` do?

- We aren’t changing the value of stack pointer
- **We are going into memory at the address of (esp+4) and storing the contents of eax there.**
  - Like saying `*(esp + 4 bytes) = eax`
- Remember the destination comes first, then the source
- We aren’t changing the data in eax



Moves the stack pointer forward 4 bytes.

[✓ Copies the data from eax into the memory at address ($esp + 4)]

Copies the data from memory at address ($esp + 4) to the eax register

Adds 4 to eax and stores it on the stack

I don't know... please explain the answer to me.

WHAT IS THIS FUNCTION DOING?

```
(gdb) disas fun
Dump of assembler code for function fun:
   0x08048394 <+0>:     push   ebp
   0x08048395 <+1>:     mov    ebp,esp
   0x08048397 <+3>:     cmp    DWORD PTR [ebp+0x8],0xa
   0x0804839b <+7>:     jle    0x80483a4 <fun+16>
   0x0804839d <+9>:     mov    eax,0x32
   0x080483a2 <+14>:    jmp    0x80483a9 <fun+21>
   0x080483a4 <+16>:    mov    eax,0x0
   0x080483a9 <+21>:    pop    ebp
   0x080483aa <+22>:    ret
End of assembler dump.
```

IT'S JUST THIS…

```
int fun(int x)
{
   if(x > 10)
      return 50;
   else
      return 0;
}
```

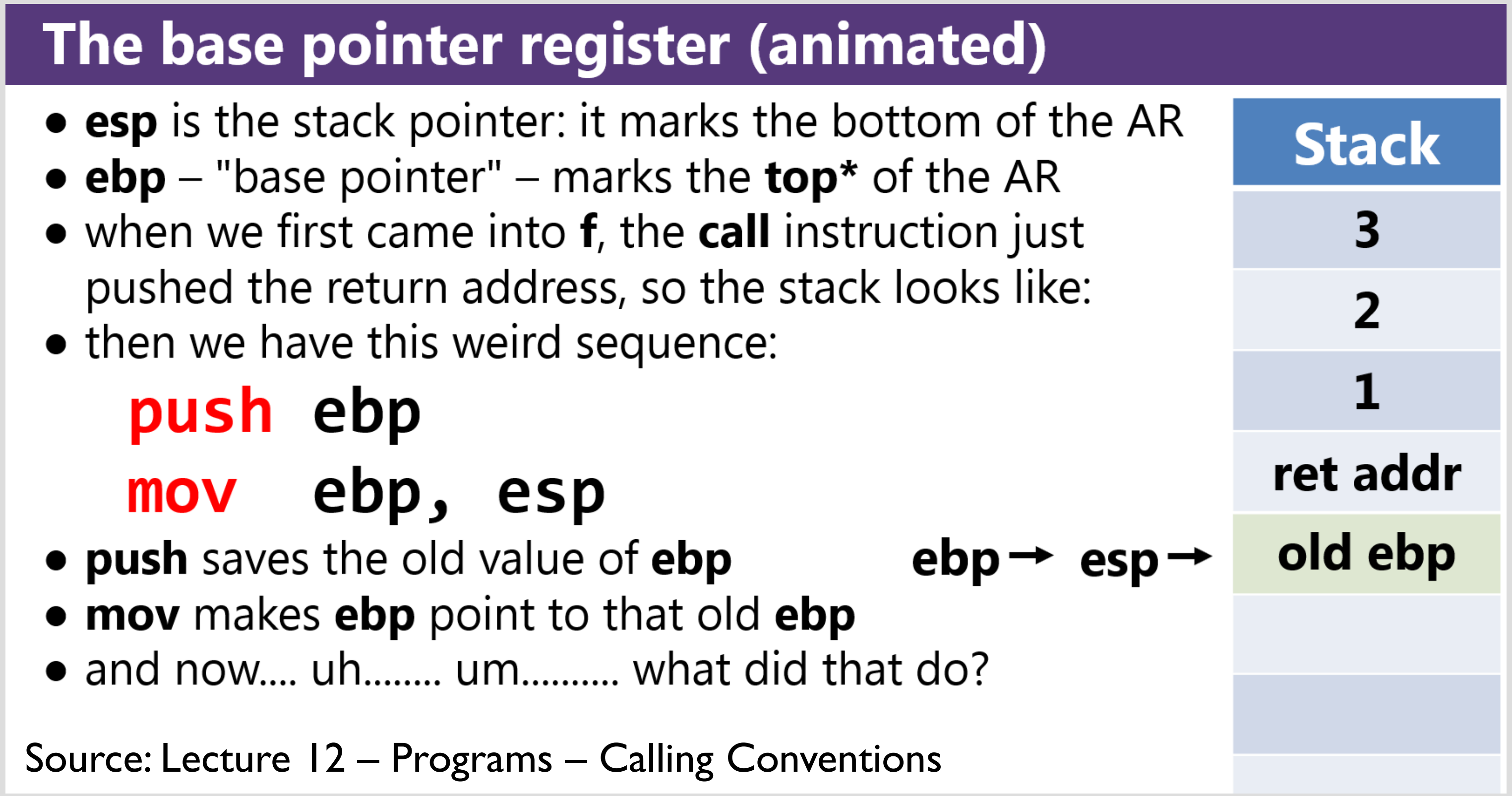
```
(gdb) disas /m fun
Dump of assembler code for function fun:
4       {
   0x08048394 <+0>:     push   ebp
   0x08048395 <+1>:     mov    ebp,esp

5               if(x>10)
   0x08048397 <+3>:     cmp    DWORD PTR [ebp+0x8],0xa
   0x0804839b <+7>:     jle    0x80483a4 <fun+16>

6                       return 50;
   0x0804839d <+9>:     mov    eax,0x32
   0x080483a2 <+14>:    jmp    0x80483a9 <fun+21>

7               else
8                       return 0;
   0x080483a4 <+16>:    mov    eax,0x0

9       }
   0x080483a9 <+21>:    pop    ebp
   0x080483aa <+22>:    ret

End of assembler dump.
```

```
(gdb) disas /m fun
Dump of assembler code for function fun:
4         {
   0x08048394 <+0>:     push   ebp
   0x08048395 <+1>:     mov    ebp,esp

5                 if(x>10)
   0x08048397 <+3>:     cmp    DWORD PTR [ebp+0x8],0xa
```

As we add local variables in our function, we will move esp down more. ebp will stay in place there

ebp + 0x8 is actually referencing the first argument passed to our function!

Recall:



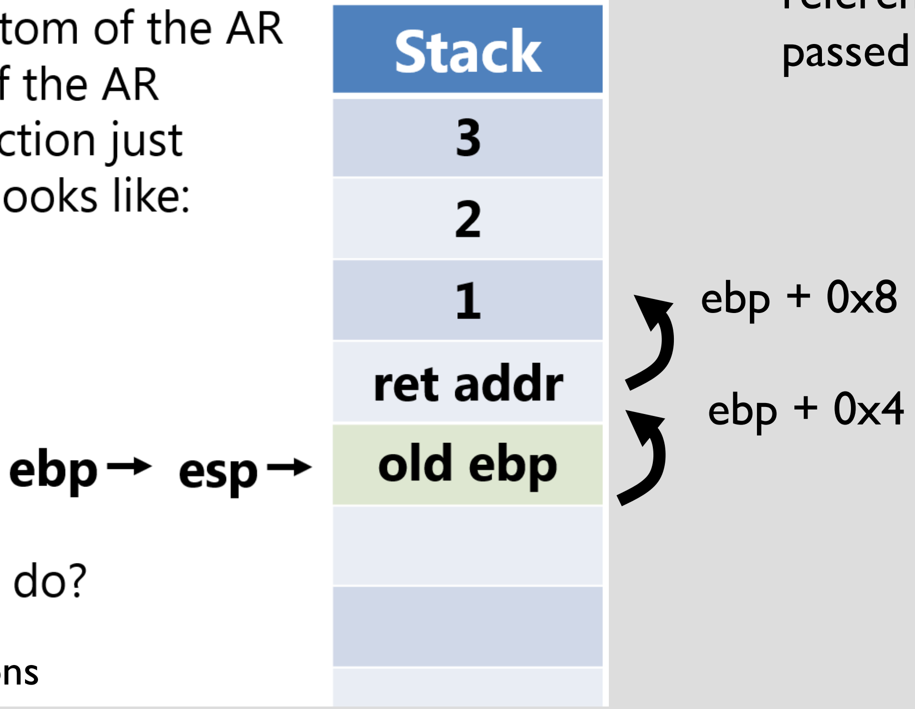
The base pointer register (animated)

- **esp** is the stack pointer: it marks the bottom of the AR
- **ebp** – "base pointer" – marks the **top*** of the AR
- when we first came into **f**, the **call** instruction just pushed the return address, so the stack looks like:
- then we have this weird sequence:

```
push  ebp
mov   ebp, esp
```

- **push** saves the old value of **ebp**
- **mov** makes **ebp** point to that old **ebp**
- and now.... uh........ um.......... what did that do?

Source: Lecture 12 – Programs – Calling Conventions



```
(gdb) disas /m fun
Dump of assembler code for function fun:
4       {
   0x08048394 <+0>:     push   ebp
   0x08048395 <+1>:     mov    ebp,esp

5               if(x>10)
   0x08048397 <+3>:     cmp    DWORD PTR [ebp+0x8],0xa
   0x0804839b <+7>:     jle    0x80483a4 <fun+16>

6                       return 50;
   0x0804839d <+9>:     mov    eax,0x32
   0x080483a2 <+14>:    jmp    0x80483a9 <fun+21>

7               else
8                       return 0;
   0x080483a4 <+16>:    mov    eax,0x0

9       }
   0x080483a9 <+21>:    pop    ebp
   0x080483aa <+22>:    ret

End of assembler dump.
```

ASSEMBLY IS ‘FUN’?

LAB 5

- Lab 5 is out and it's on everyone's favorite topic: *POINTERS!*
- Two functions to implement:

```
int filter(void* output, const void* input, int length, int item_size, PREDICATE pred)
{
    // FILL ME IN!
}

int less_than_50(const void* p)
{
    // FILL ME IN!
}
```

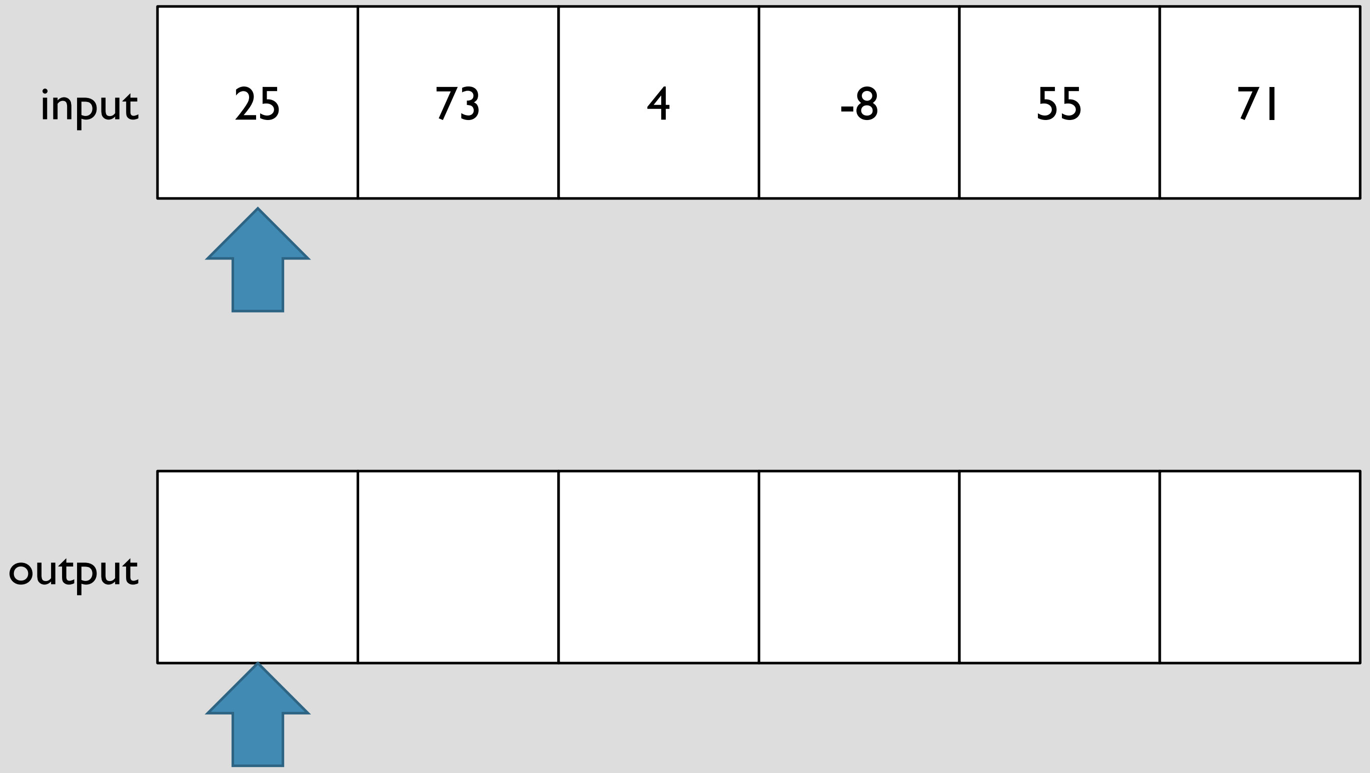
```c
int filter(void* output, const void* input, int length, int item_size, PREDICATE pred)
{
    // FILL ME IN!
}

int less_than_50(const void* p)
{
    // FILL ME IN!
}
```

- Start with less_than_50. It's your 'predicate' function.
  - Predicate functions are just "something that gives a yes-or-no answer."
- You return zero or nonzero based on whether the value of p is less than 50 or not.
  - If it's less than 50, return *anything that isn't zero*, otherwise, return 0
- You'll need to change that data type! Void pointers are useless here.
- This function will be used in filter to determine whether or not a value should be included in the output array.

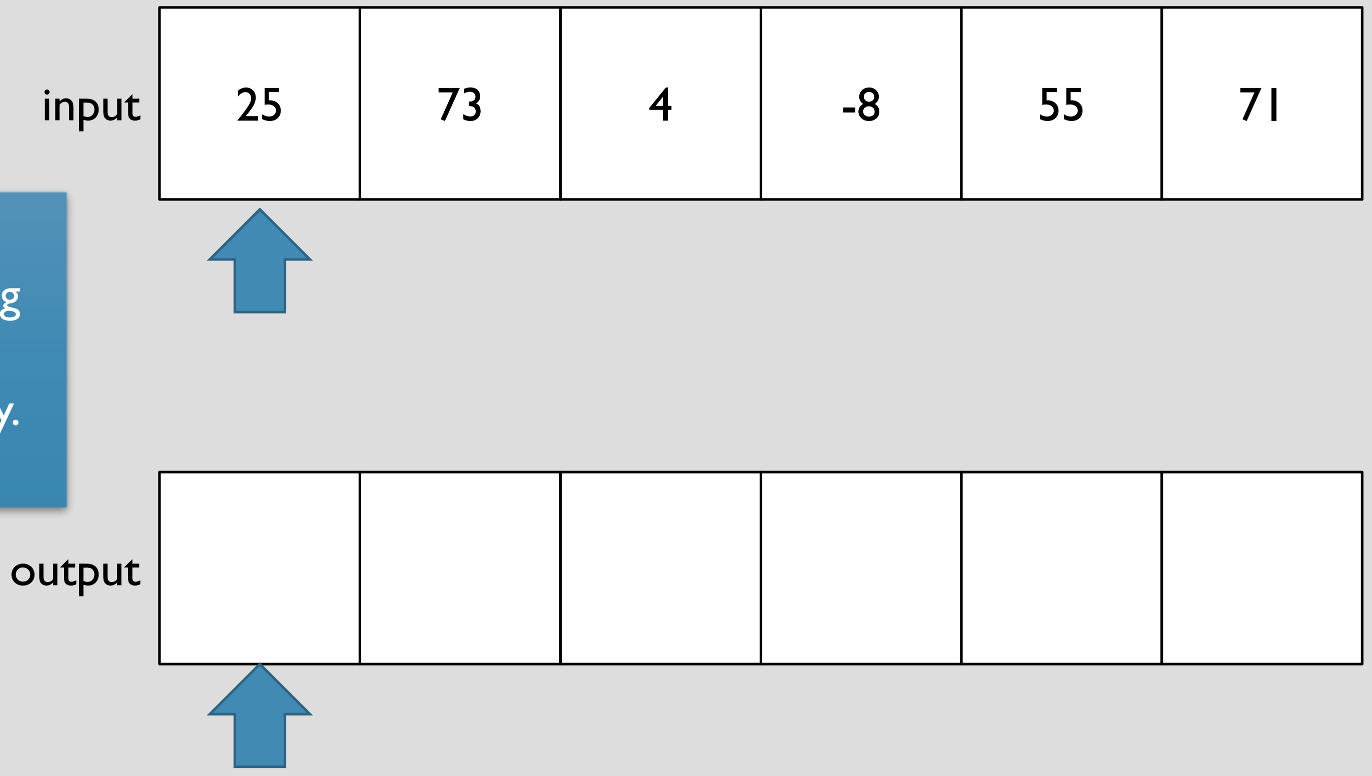
```c
int filter(void* output, const void* input, int length, int item_size, PREDICATE pred)
{
    // FILL ME IN!
}

// call in main
...
int filtered_len = filter(filtered, float_values, NUM_VALUES, sizeof(float), &less_than_50);
...
```

- Filter is a bit more complicated.
- You have your input array and its length. You'll iterate over it. Remember that each element in the array is item_size in length (be careful moving that pointer forward the correct amount, no PTR_ADD_BYTES here!).
- On each of those elements, call the predicate function on them (pred)
  - If it returns a nonzero value, then use memcpy to copy the data from input to output (again, be careful with size)
  - Remember to be careful about maintaining your place in the output array as well.
- Oh yeah, and also keep a quick count of how many items you copied to output and return that

HIGH-LEVEL OVERVIEW OF FILTER

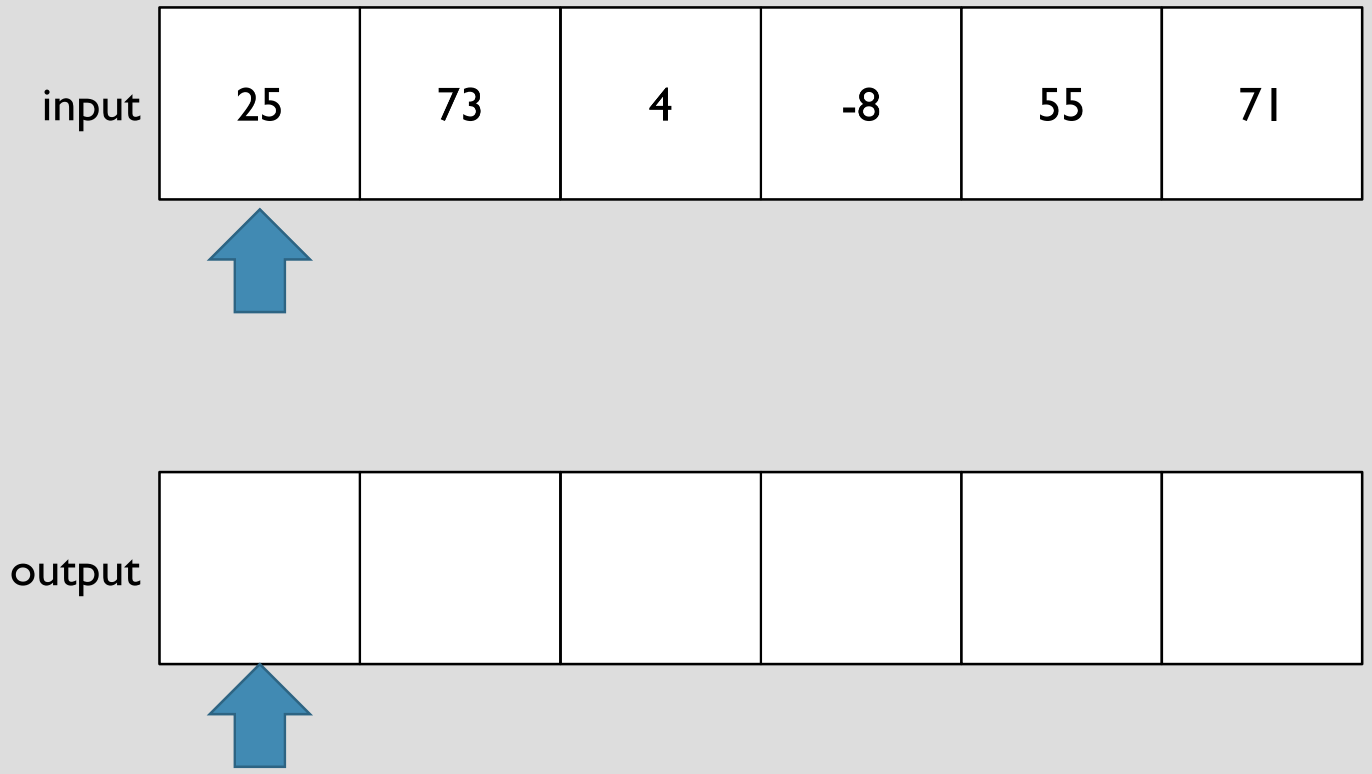


HIGH-LEVEL OVERVIEW OF FILTER

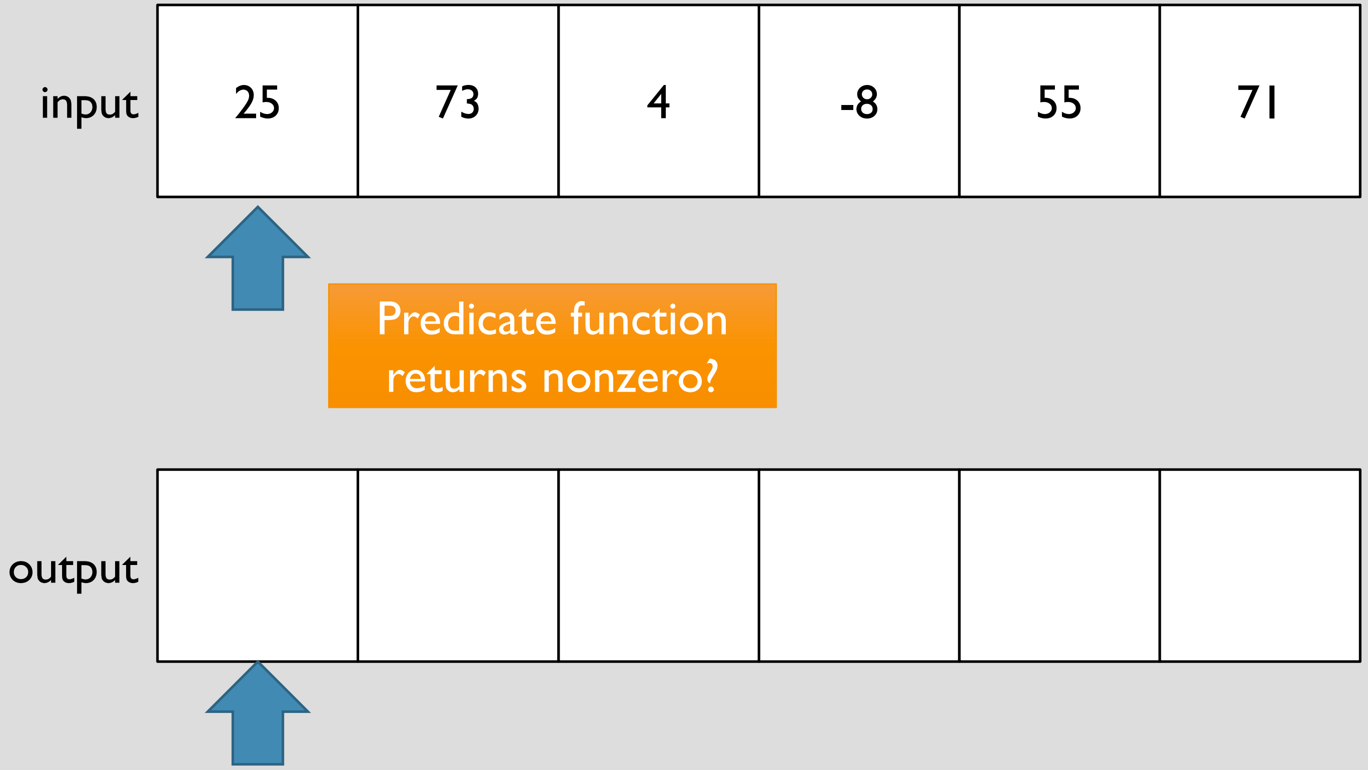


Remember, we're using pointers here, not indexing into the array.

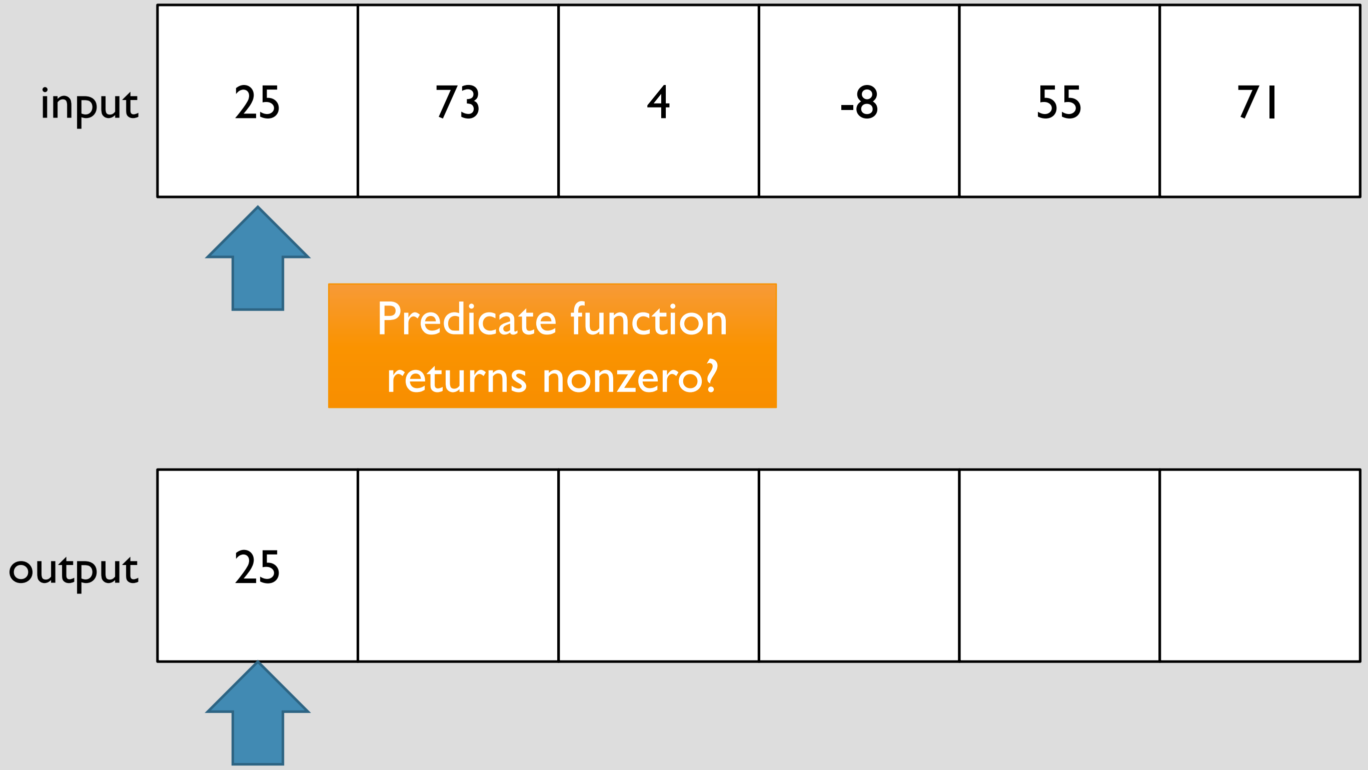
HIGH-LEVEL OVERVIEW OF FILTER



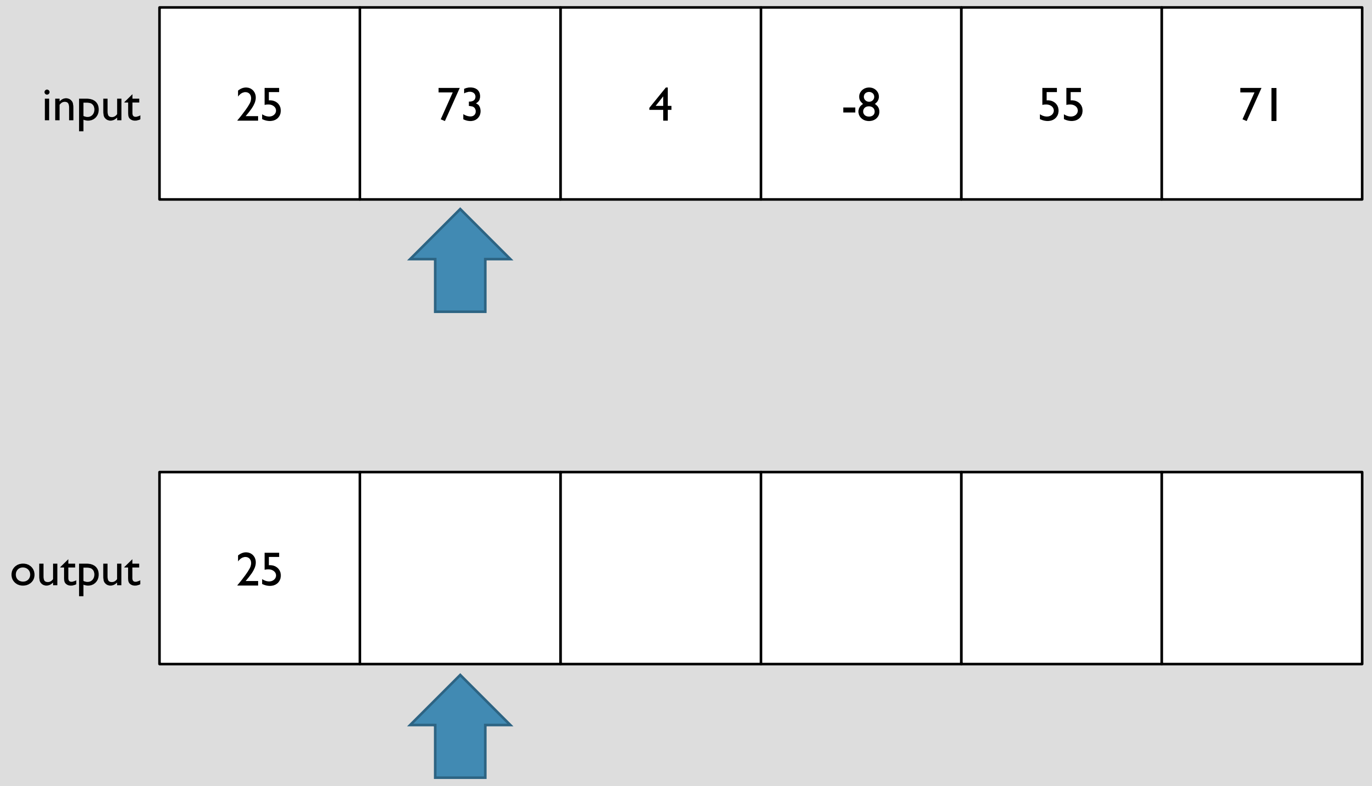
HIGH-LEVEL OVERVIEW OF FILTER



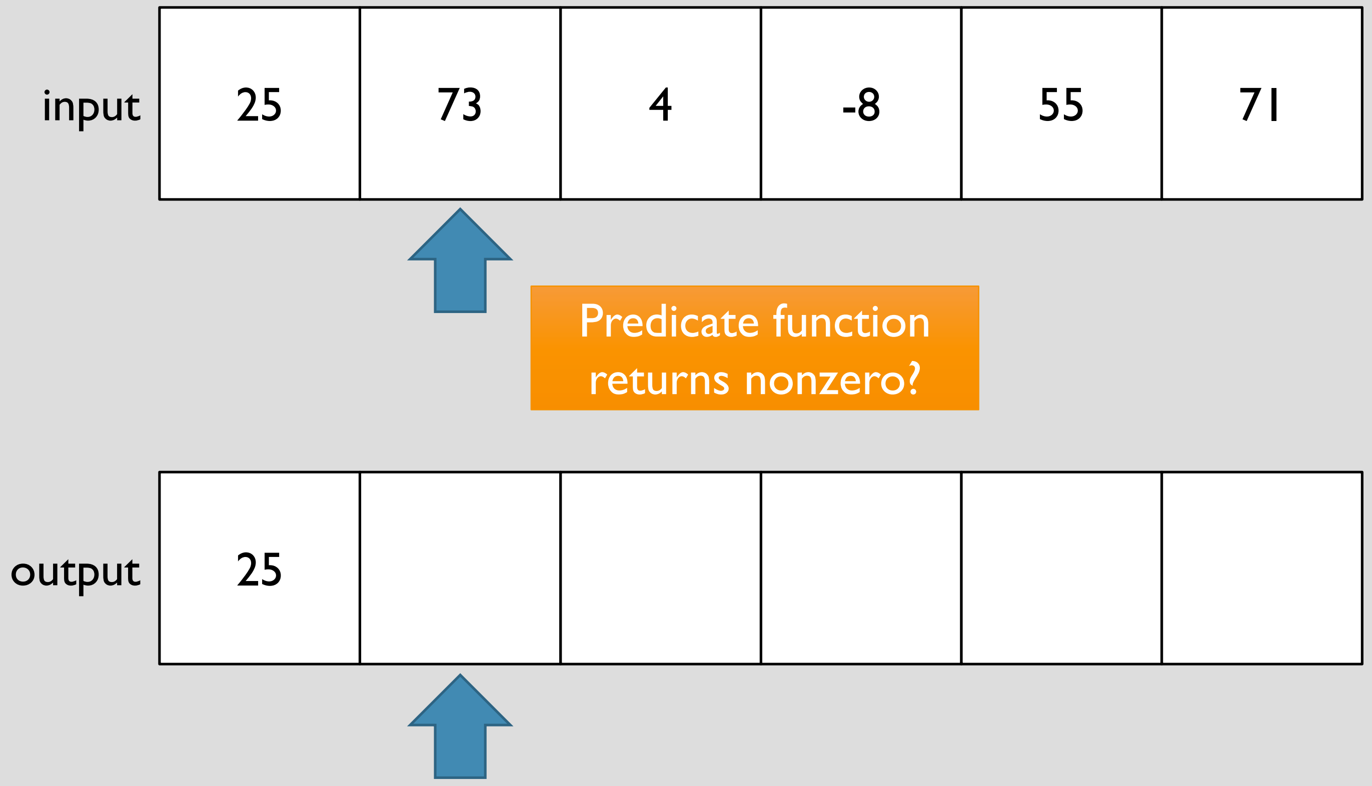
HIGH-LEVEL OVERVIEW OF FILTER



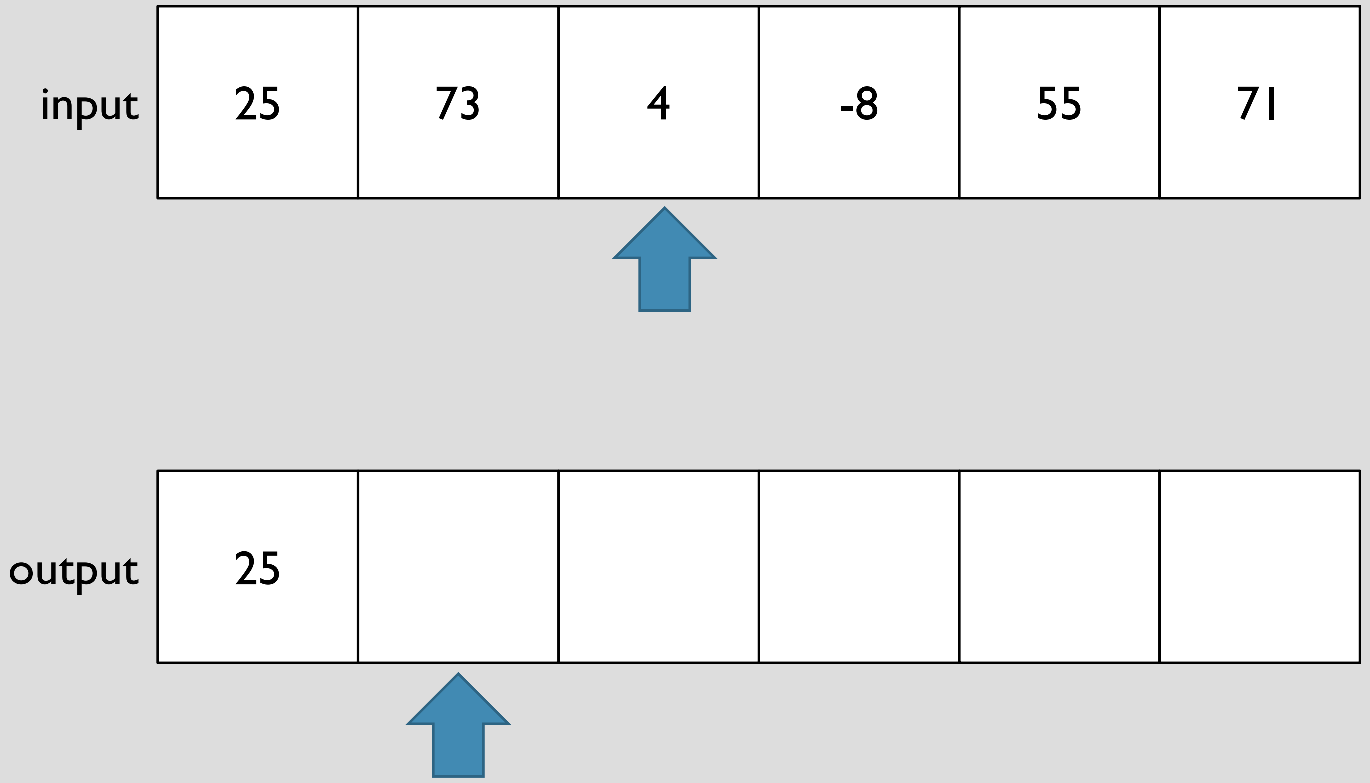
HIGH-LEVEL OVERVIEW OF FILTER



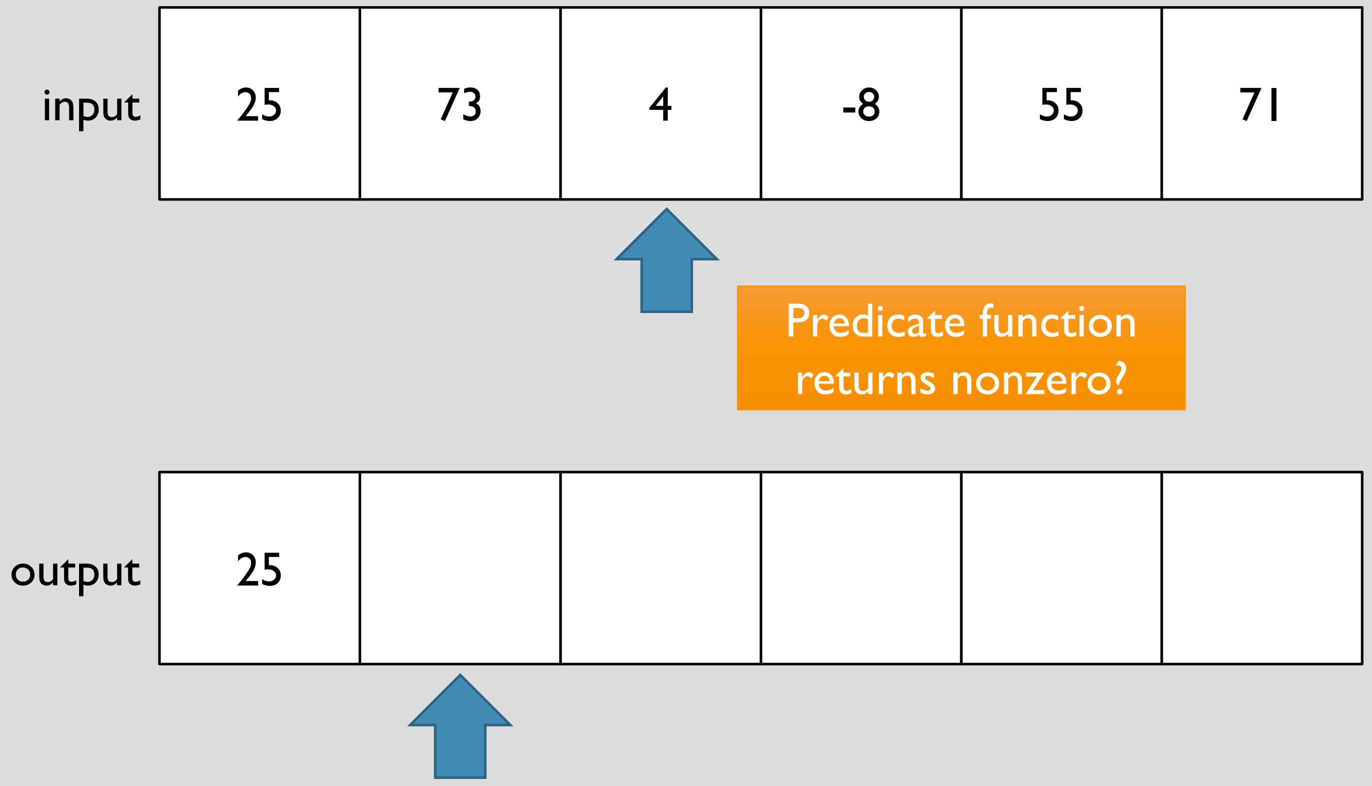
HIGH-LEVEL OVERVIEW OF FILTER



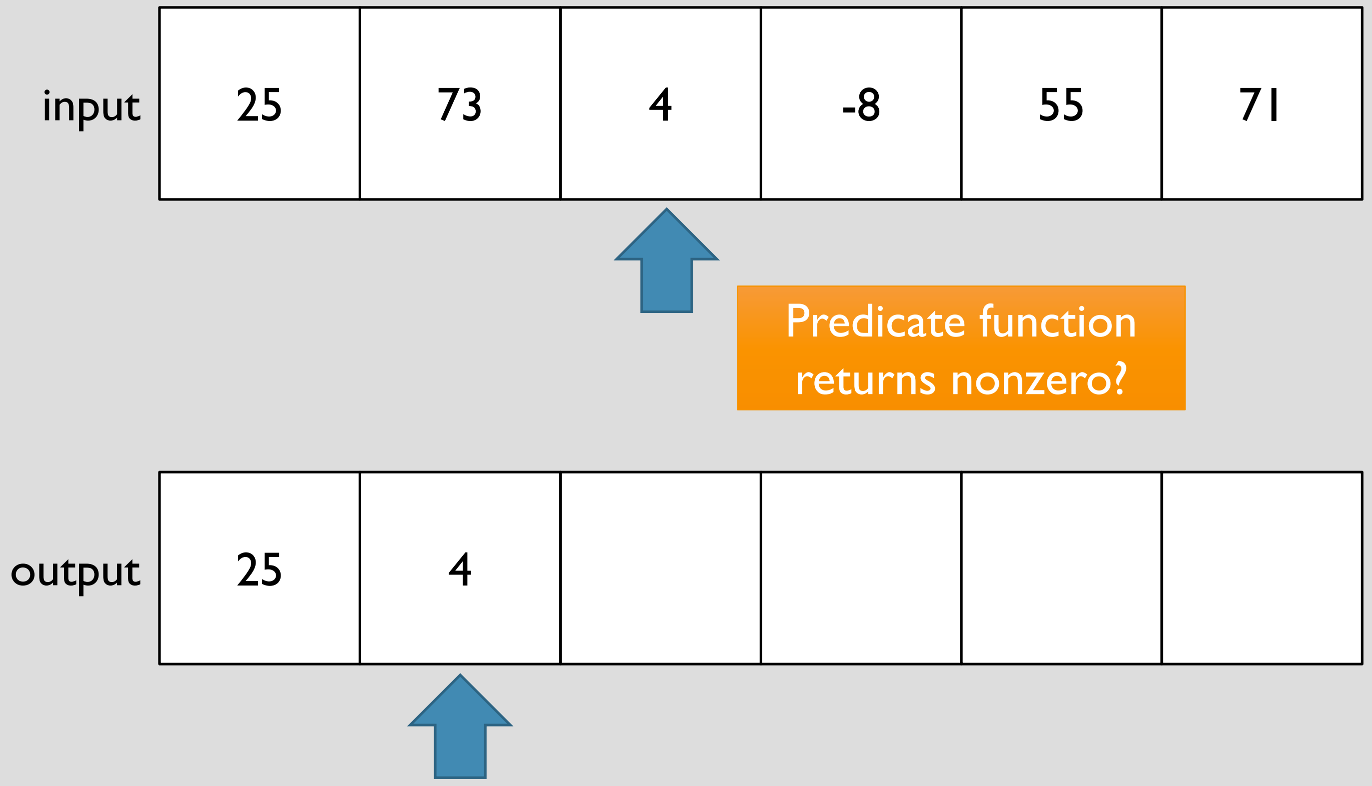
HIGH-LEVEL OVERVIEW OF FILTER



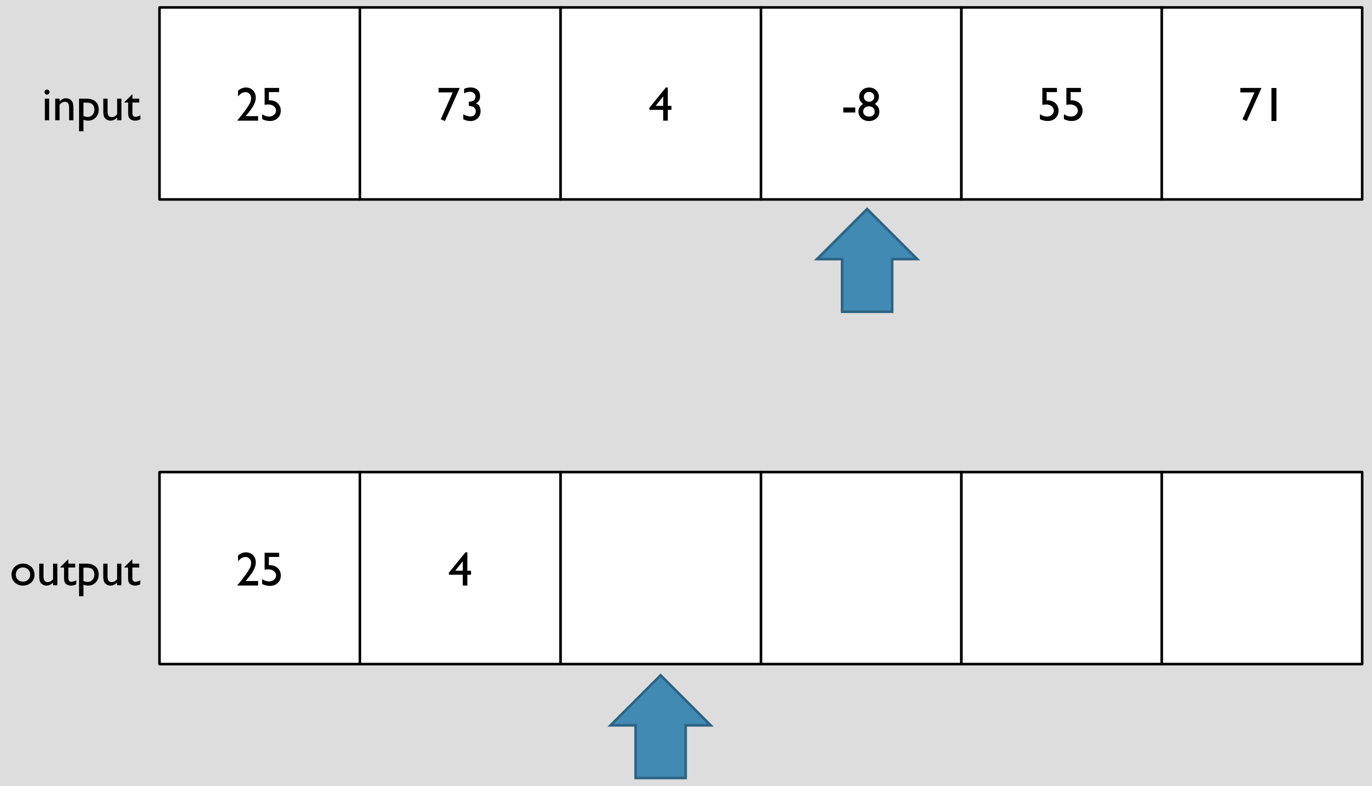
HIGH-LEVEL OVERVIEW OF FILTER



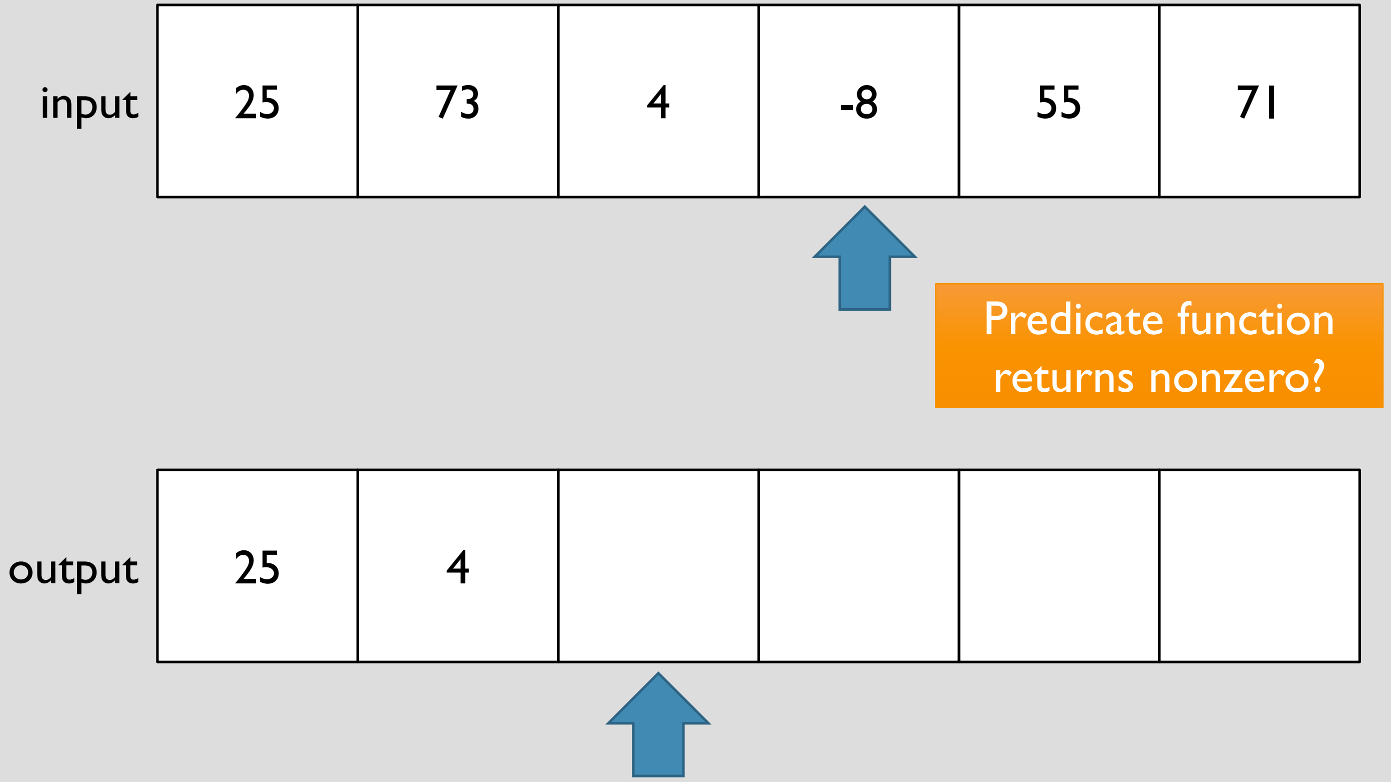
HIGH-LEVEL OVERVIEW OF FILTER



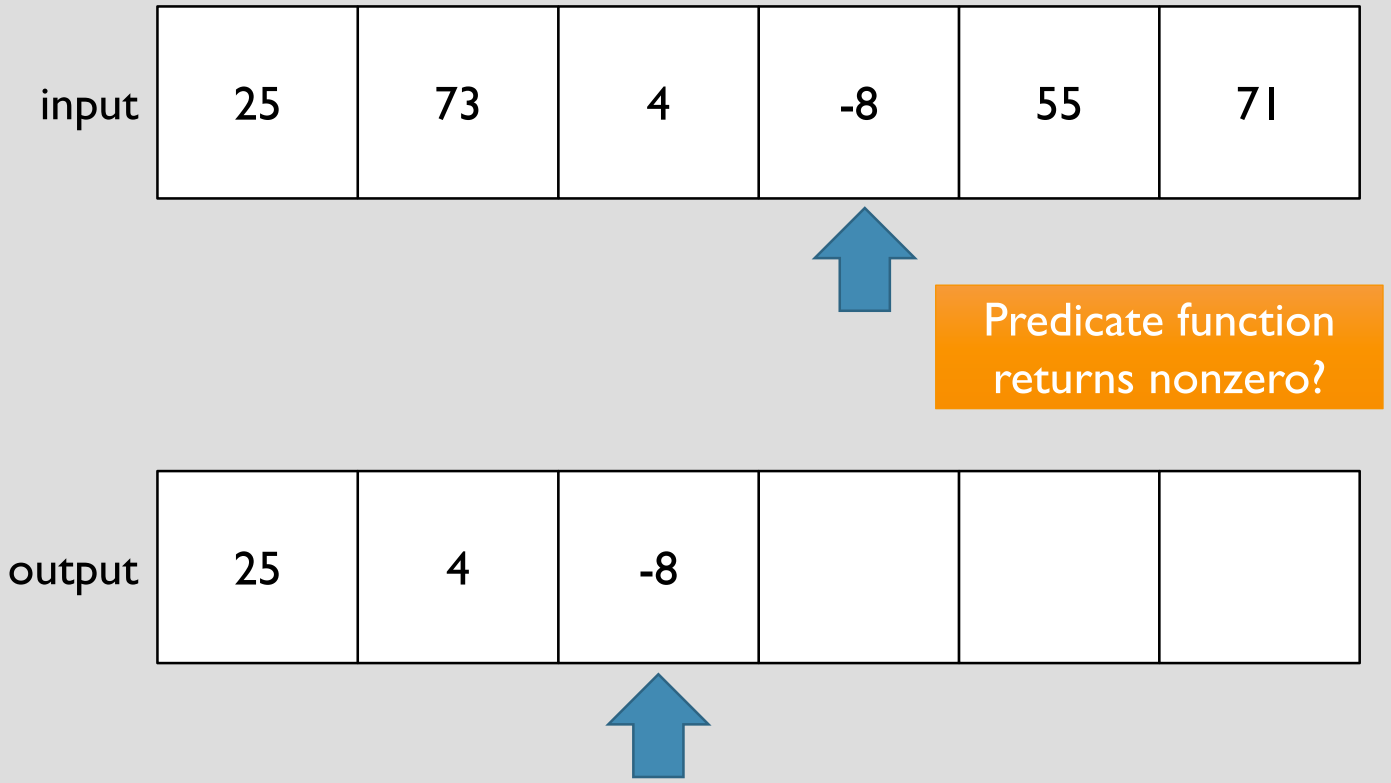
HIGH-LEVEL OVERVIEW OF FILTER



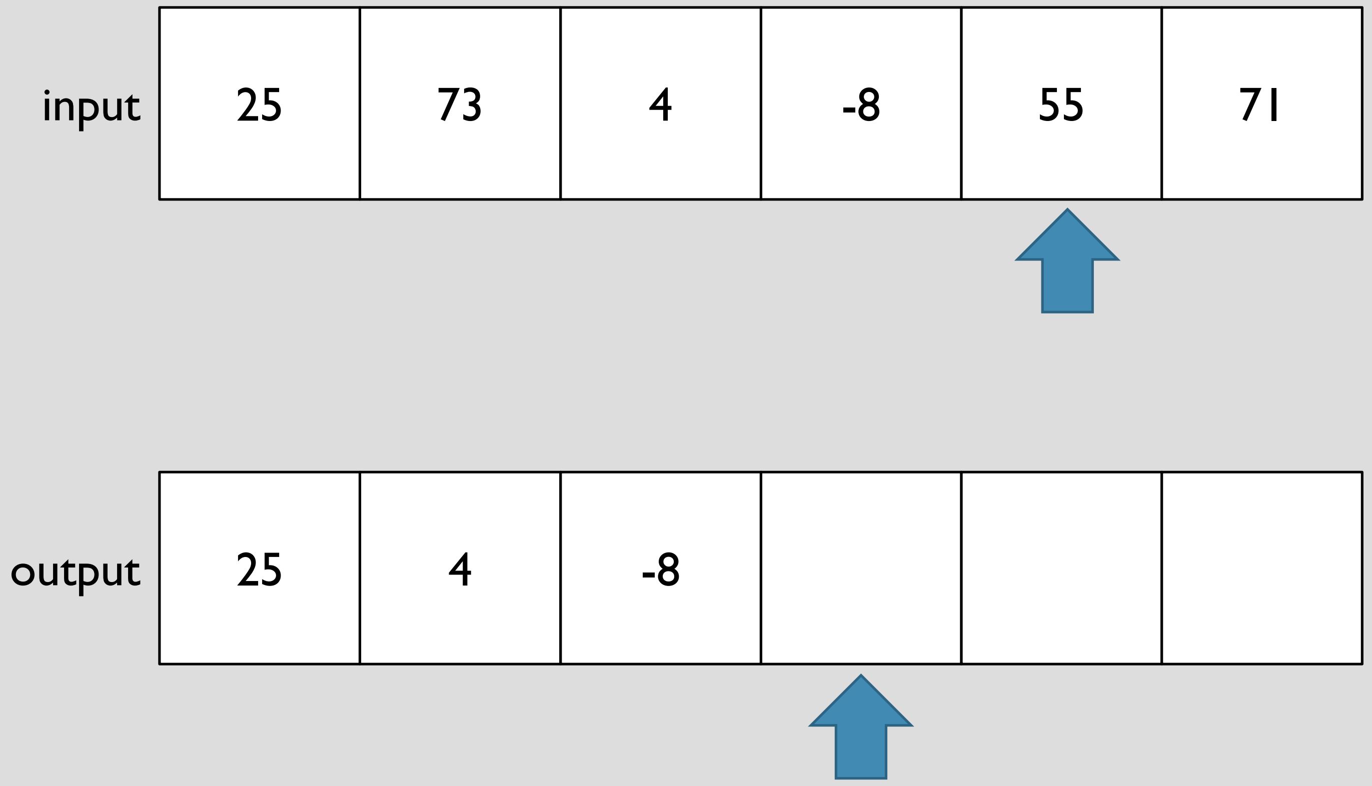
HIGH-LEVEL OVERVIEW OF FILTER



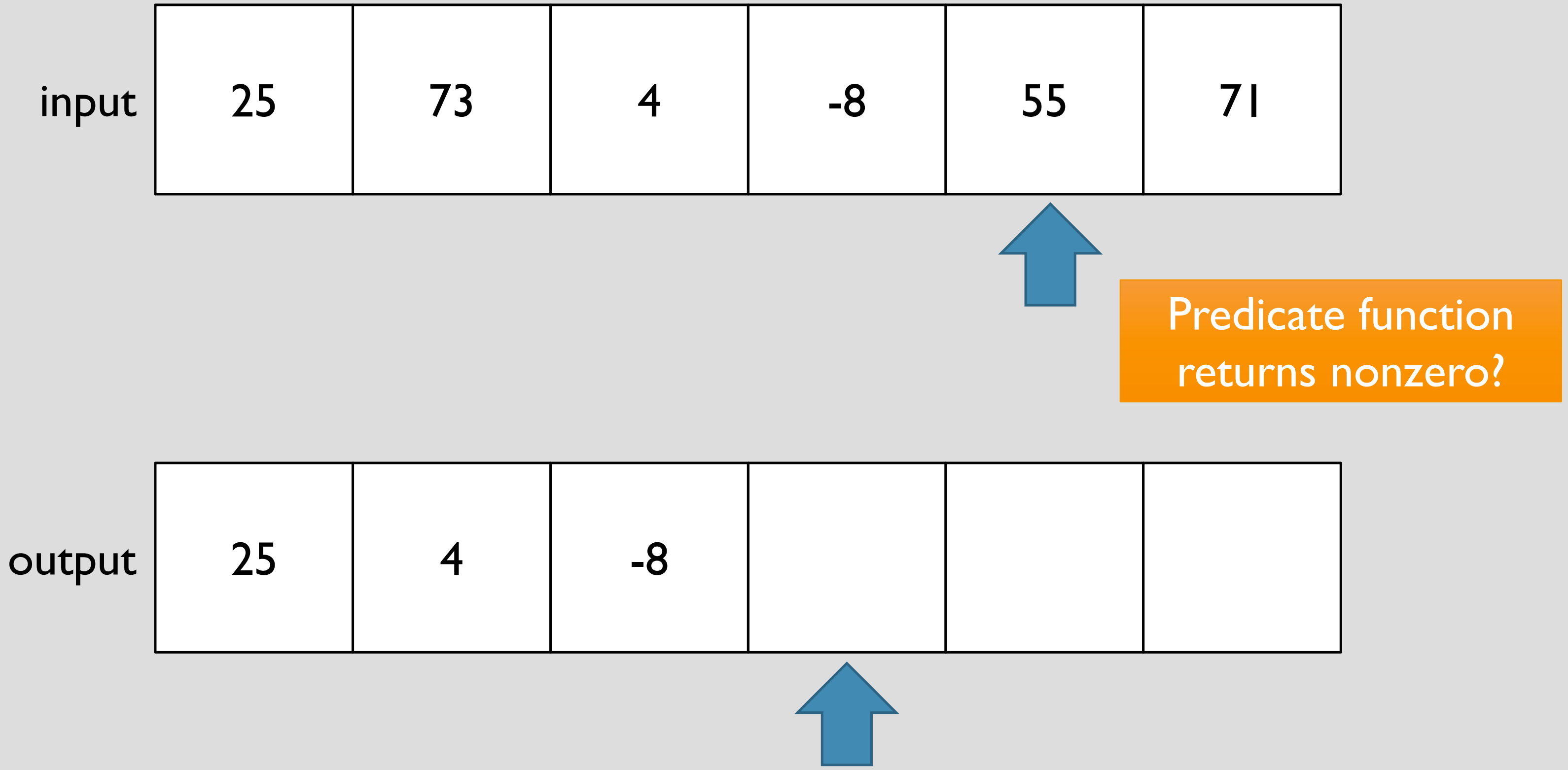
HIGH-LEVEL OVERVIEW OF FILTER



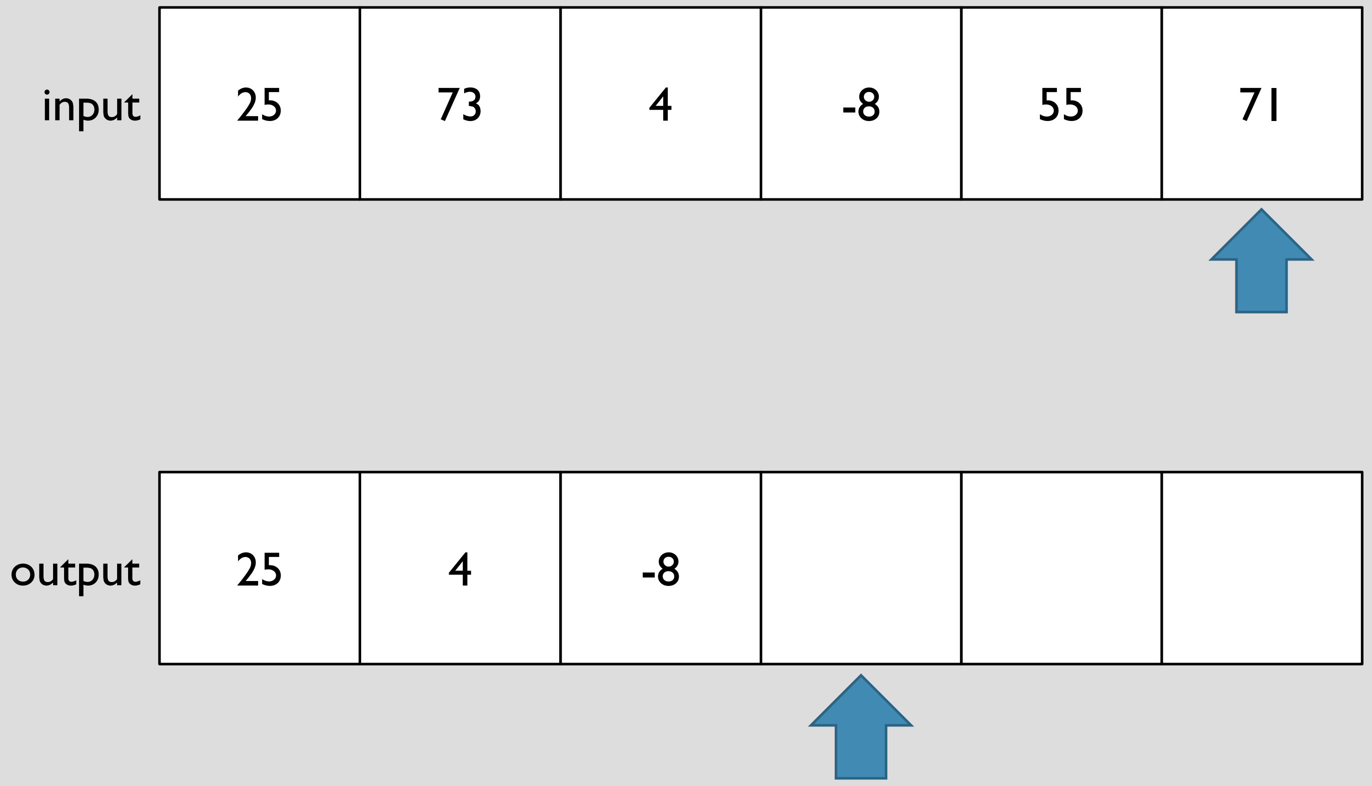
HIGH-LEVEL OVERVIEW OF FILTER



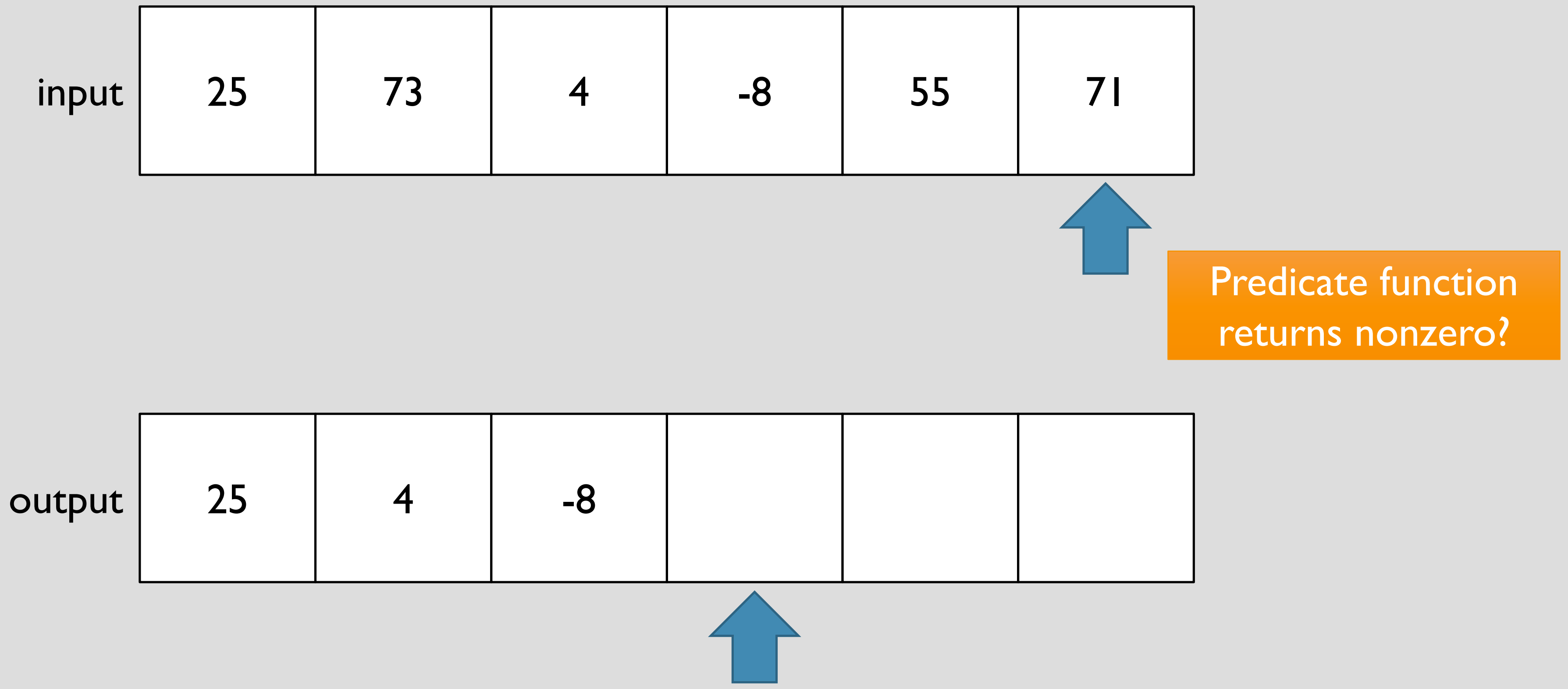
HIGH-LEVEL OVERVIEW OF FILTER



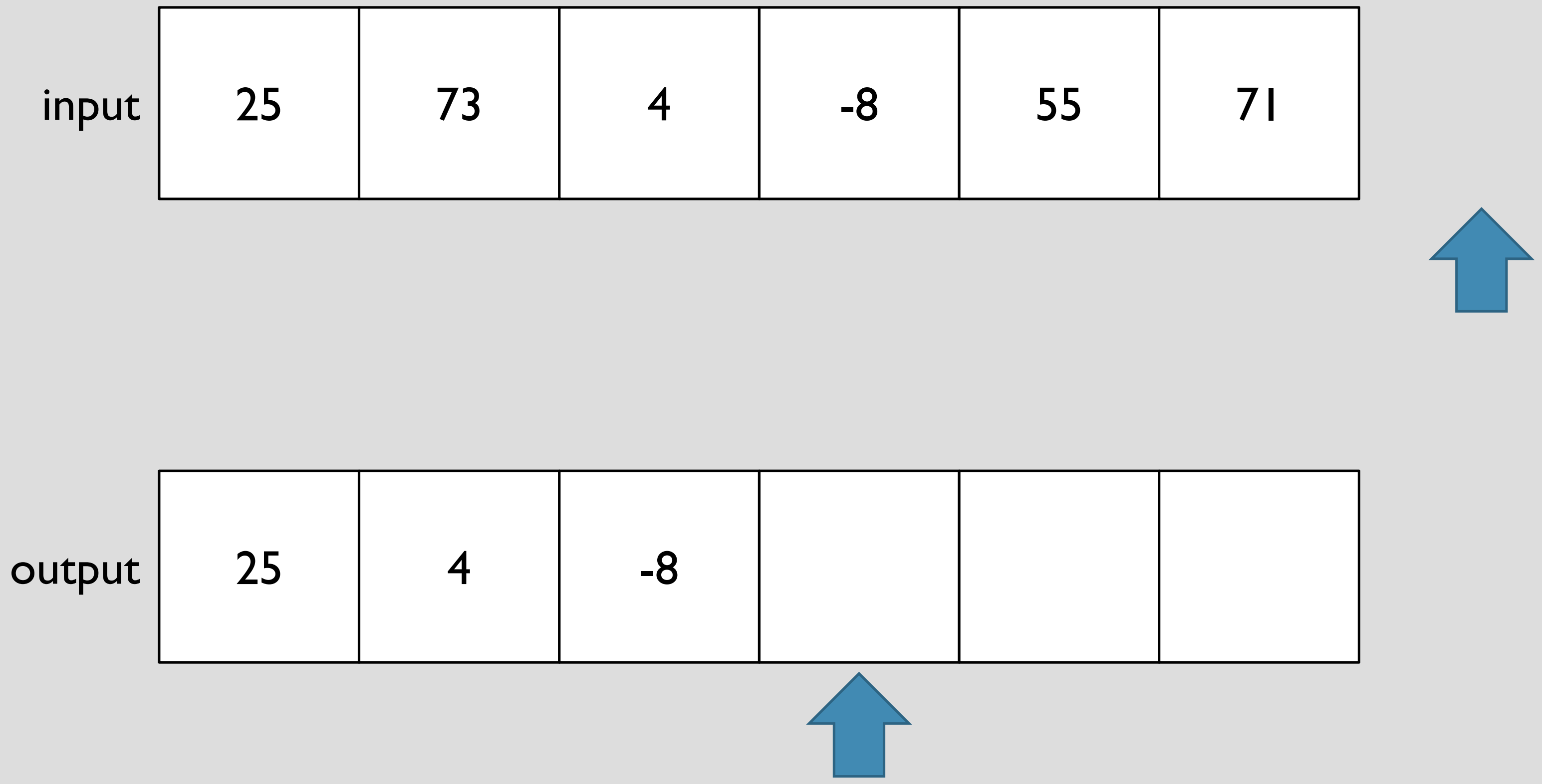
HIGH-LEVEL OVERVIEW OF FILTER



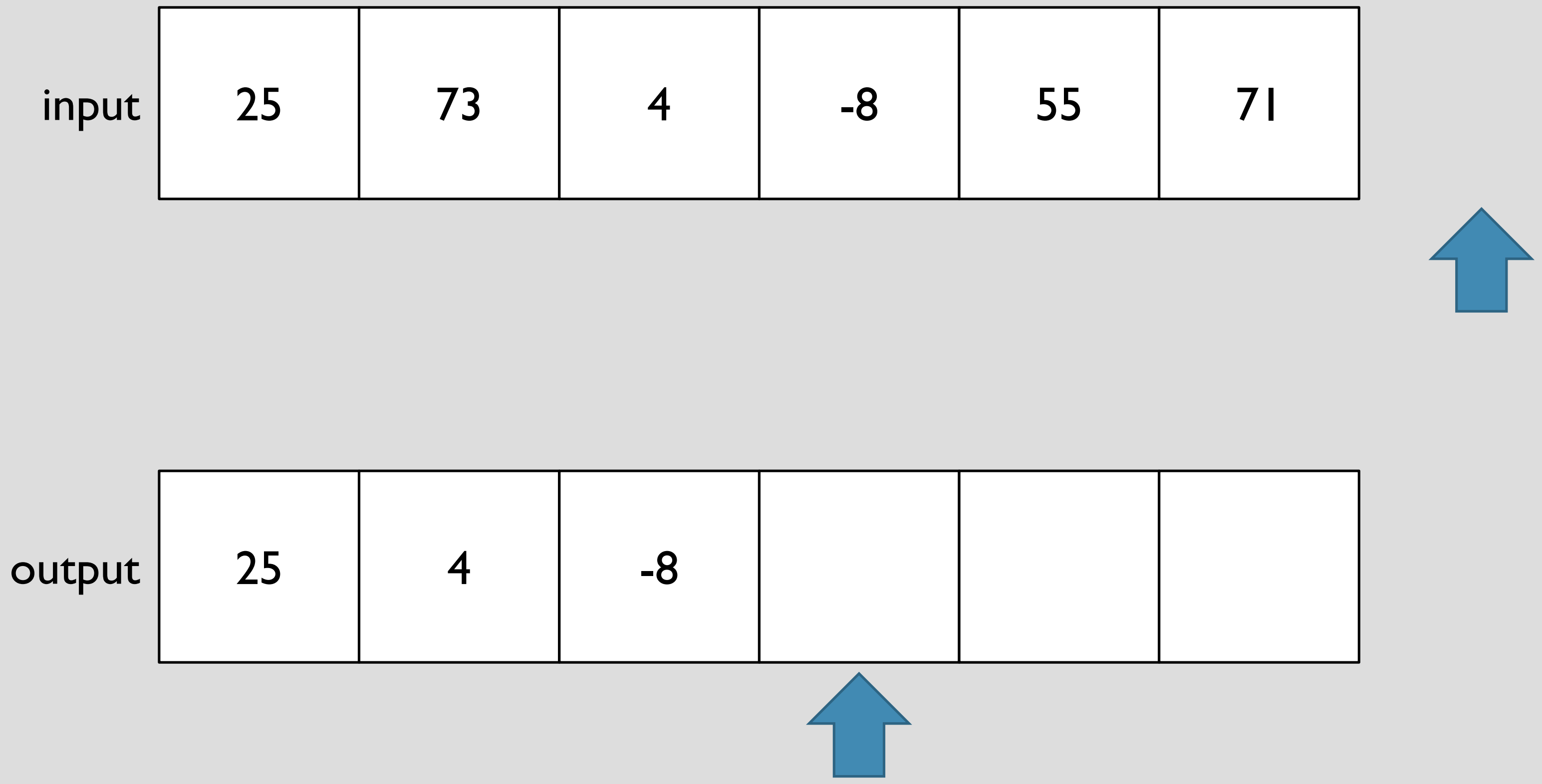
HIGH-LEVEL OVERVIEW OF FILTER



HIGH-LEVEL OVERVIEW OF FILTER



HIGH-LEVEL OVERVIEW OF FILTER



Return **3**, the number of items we copied into **`output`**

LAB 5

- https://jarrettbillingsley.github.io/teaching/classes/cs0449/labs/lab5.html
- Take a look at the implementation of qsort demo code
- Follow the instructions.
- Due Saturday