ECS 36A, April 17, 2023

Announcements

- Be sure you use this command to run your program in the CSIF before submitting it to Gradescope:

  `gcc -ansi -pedantic -Wall` *`filename`*`.c -o` *`filename`*

  This is the command Gradescope uses to compile your progrm!
- The following causes compile errors:

  ```
  for(int x = 0; x < maxval; x++)
  ```

  - Not part of the C99 standard
- TA's Office Hours: MWF 10:00am–11:00am in 53 Kemper

How to Copy to/from the CSIF

- Download file *f* to CSIF:
  1. Activate the VPN that connects you to the CSIF (like Pulse)
  2. Give the following command on *your* computer:

     ```
     scp f pc12.cs.ucdavis.edu:
     ```

  The file *f* is now in your CSIF home directory
- Upload file *f* from CSIF:
  1. Activate the VPN that connects you to the CSIF (like Pulse)
  2. Give the following command on *your* computer:

     ```
     scp pc12.cs.ucdavis.edu:f .
     ```

  The file *f* is now in the current working directory/folder on your system

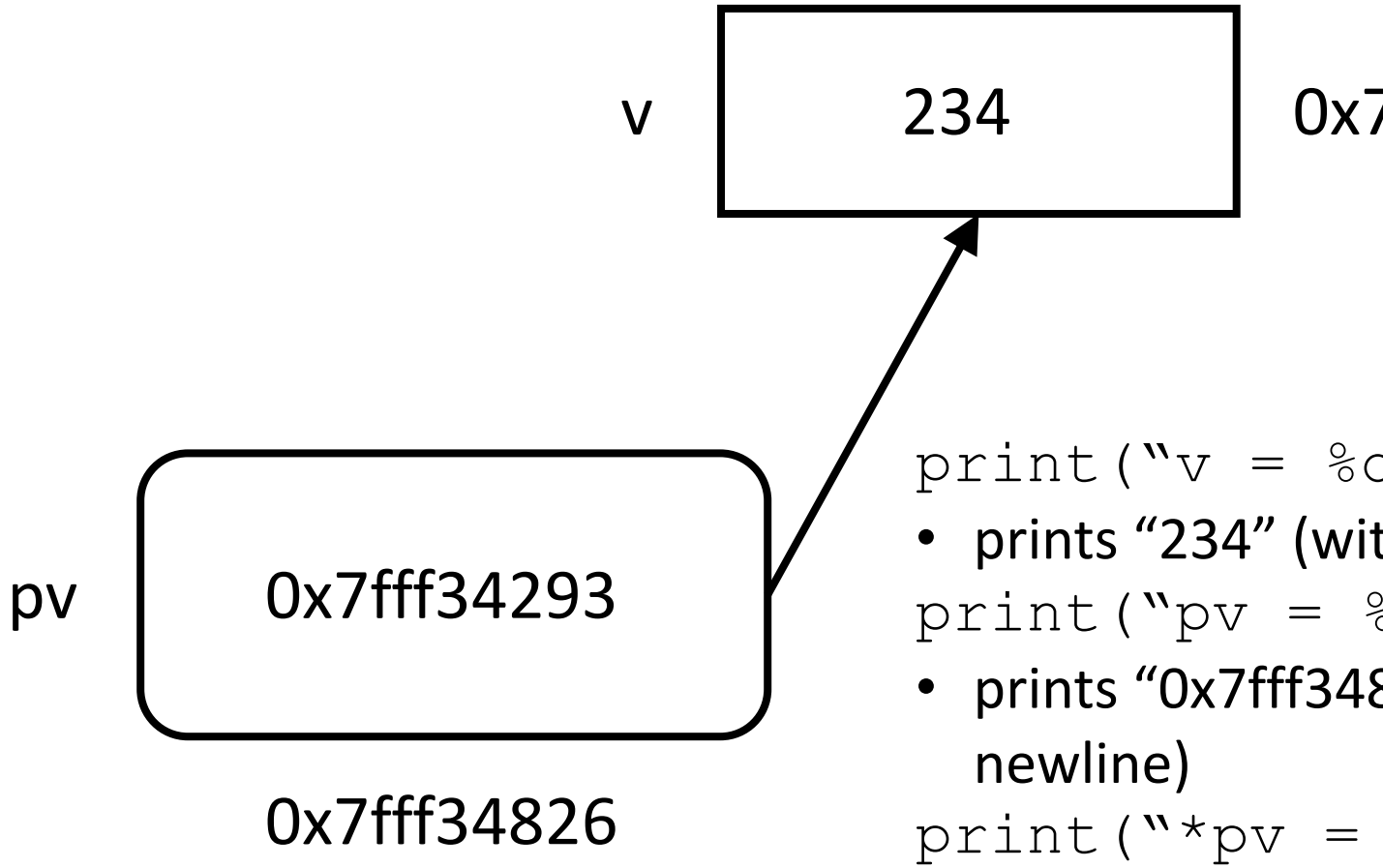
Pointers

- A variable containing the address of another variable
- Example:

```
int x = 0;
int *px;
px = &x;
printf("x = %d, px = %p, *px = %d\n", x, px, *px);
```

- Operators:
  - &*variable*: address of *variable*
  - **variable*: what is in the memory location with the address stored in *variable*

In Pictures




```
print("v = %d\n", v);
```

- prints "234" (without the "s, ending in newline)

```
print("pv = %p\n", (void *)pv);
```

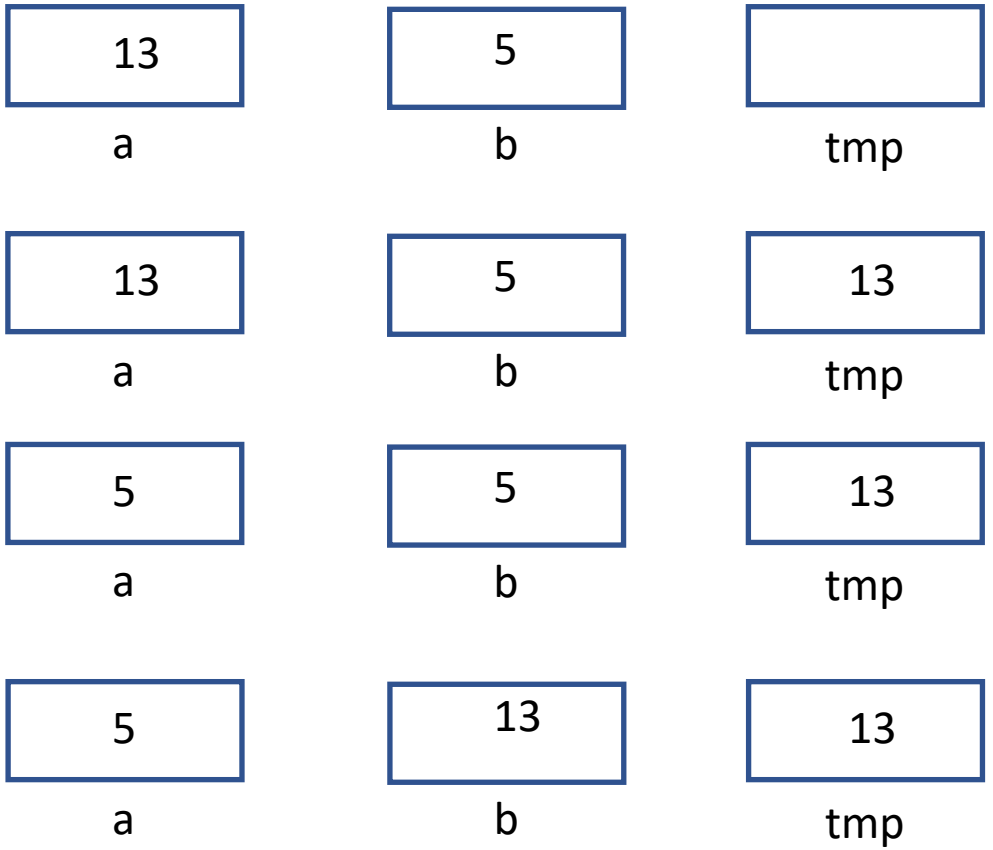
- prints "0x7fff34826" (without the "s, ending in newline)

```
print("*pv = %d\n", *py);
```

- prints "234" (without the "s, ending in newline)

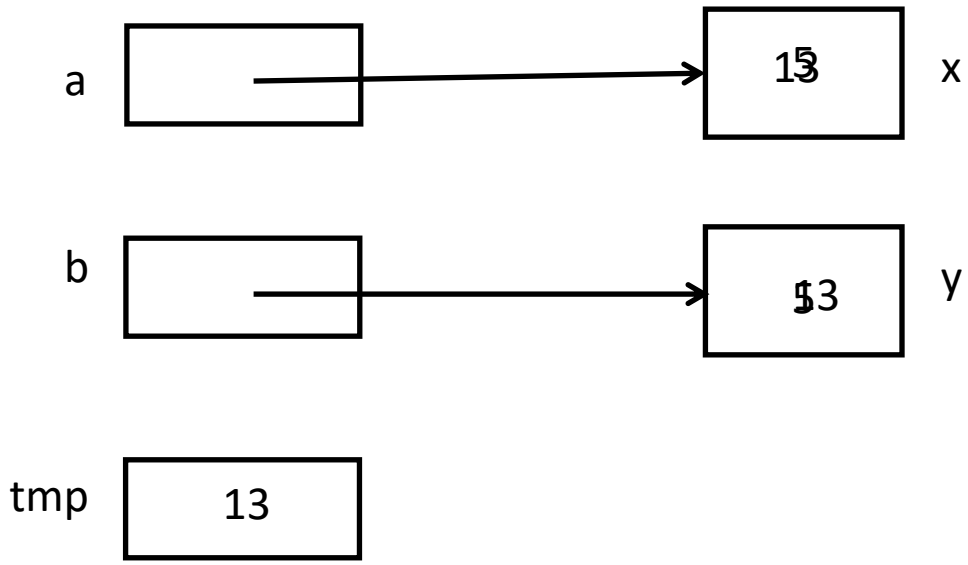
Function Arguments (Review)

```
void swap(int a, int b)
{
        int tmp;

        tmp = a;
        a = b;
        b = tmp;
}
```



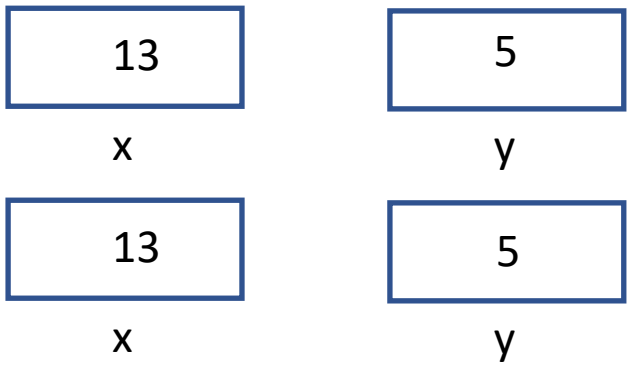
Function Arguments as Pointers

```
void swap(int *a, int *b)
{
        int tmp;

        tmp = *a;
        *a = *b;
        *b = tmp;
}
```



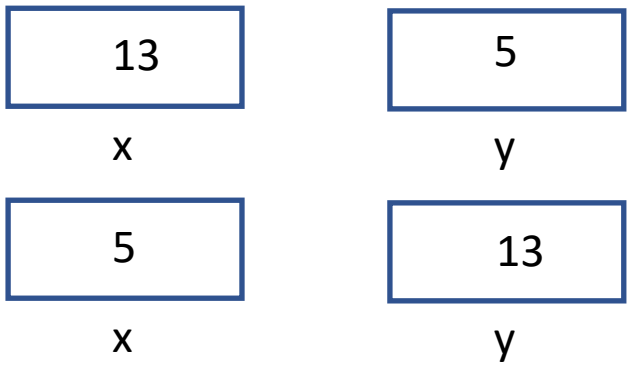
And On The Calling End (Review)

```
x = 13;
y = 5;
printf("x = %d, y = %d\n", x, y);
swap(x, y);
printf("x = %d, y = %d\n", x, y);
```



And On The Calling End

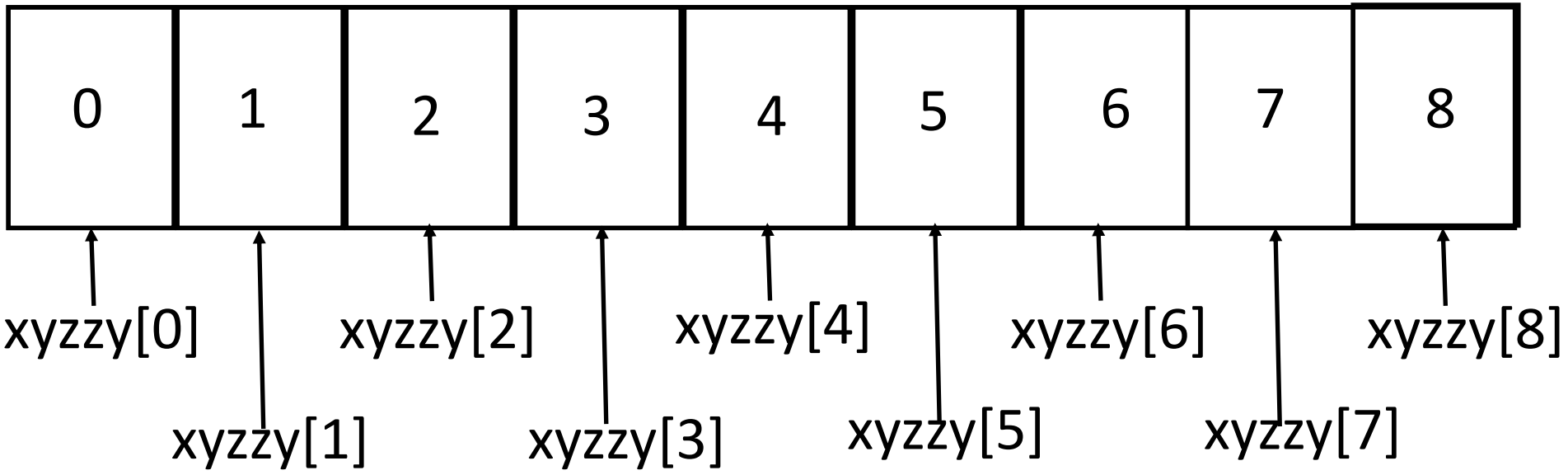
```
x = 13;
y = 5;
printf("x = %d, y = %d\n", x, y);
swap(&x, &y);
printf("x = %d, y = %d\n", x, y);
```



Scope

- When multiple variables have the same name, which one is used?
  - Rule #1: two variables cannot have the same name in a block (e.g., function)
- Use the variable that is “nearest” to the reference
  - That’s the one in scope

C Arrays


0
1
2
3
4
5
6
7
8
xyzzy[0]
xyzzy[1]
xyzzy[2]
xyzzy[3]
xyzzy[4]
xyzzy[5]
xyzzy[6]
xyzzy[7]
xyzzy[8]

Arrays as Pointers and *Vice Versa*

- Arrays are simply another way to express pointers
  - So xarray[0] and *xarray refer to the same memory location
  - And xarray[12] and *(xarray+12) refer to the same memory location

Pointer Arithmetic

- *type* *x;
  - x + 10 refers to the 10th *type* object; so if *type* is an int, x + 10 refers to the 10th integer memory location beyond that which x points to
  - This is why pointers and array names are equivalent
- x + *n*: refers to the *n*th *type* object beyond x
- x – *n*: refers to the *n*th *type* object before x
- x – y: refers to the number of *type* objects between x and y
- x + y: meaningless!!!

Multidimensional Arrays

- A 2-dimensional array look like this:

x[0]	x[0][0]	x[0][1]	x[0][2]	x[0][3]
x[1]	x[1][0]	x[1][1]	x[1][2]	x[1][3]
x[2]	x[2][0]	x[2][1]	x[2][2]	x[2][3]

- Stored in row-major order as consecutive elements of a row are stored next to each other
  - Column-major order has consecutive elements of a column stored next to each other