As you arrive:

1. Start up your computer and plug it in
2. ***Log into Angel*** and go to CSSE 120
3. Do the ***Attendance Widget*** – the PIN is on the board
4. Go to the course ***Schedule Page***
5. Open the ***Slides*** for today if you wish
6. Check out today's project: `Session26_Arrays`

Plus in-class time working on these concepts AND practicing previous concepts, continued as homework.

Arrays in C

- Review pointers as function parameters
- List in Python vs Array in C
- Arrays as function parameters
- Arrays and pointers

Using pointers as parameters
Box and Pointer Diagrams

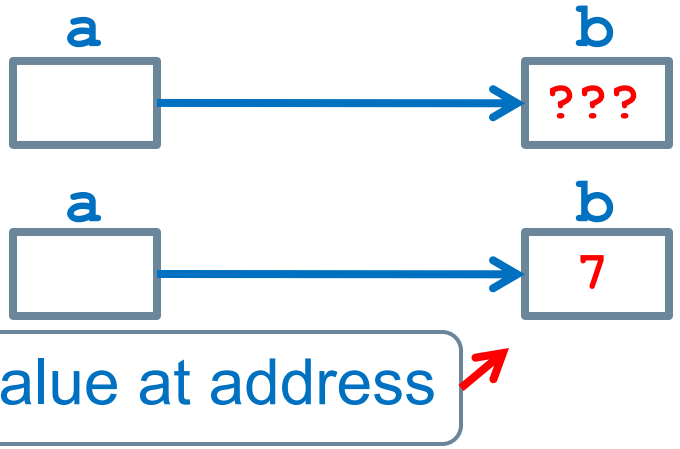
```
int b;
foo(&b);

void foo(int* a) {
    ...
    *a = 7;
}
```

Send the address of b

Receive an address

Modify value at address



Now `b` has the value 7 that was established in `foo`!

This is useful for:

- sending data back from a function via the parameters, and for
- passing large amounts of data to a function.

Thus pointers in C give us the same advantages as references-to-objects in Python.

From the last homework:

- **swap**: a function to exchange the values of two variables

- Let's look at some possibly wrong approaches and why they would not work

```
void swap1(int x, int y) {
    x = y;
    y = x;
}

void swap2(int x, int y) {
    int temp;
    temp = y;
    y = x;
    x = temp;
}


void swap3(int *x, int *y) {
    int *temp;
    temp = y;
    y = x;
    x = temp;
}
```

Q1

Arrays in C

- Arrays in C are like lists in Python
- But there are limitations on how they can be mutated

An example using lists in Python

- Consider the following Python Code:

```
list = [1, "spam", 4, "U"]
list.append(2)
list.remove("U")
length = len(list)
```

- What do these statements tell us about Python lists?
  - Elements can be mutated
  - Type does not matter
  - Size is not specified
  - Can be expanded or shrunk
  - Lists remember their length

C arrays (next slide) share only the first of these properties!

List in Python vs Array in C

- **No built-in *list* type in C**
- Array is closest data structure to list in Python
- Consider this C code

```
int size = 4;
int nums[size];
int i;
for (i = 0; i < size; i++) {
    nums[i] = i * i;
}
```

- How is this similar to lists in Python?
- Different?

Q2-3

Initialization and access

- How do we initialize a list or array?
  - Python list: **a = [1, 3, 5]**
  - C array: **int a[ ] = {1, 3, 5};**
- How do we access an element?
  - Python list: **x = a[i]**
  - C array: **x = a[i];**
- How do we access the last element?
  - Python list: **x = a[-1]**
  - C array: **x = a[size - 1];** // the array doesn't know its size.

```
int main() {
    int size = 7;
    int a[size];

    initializeArray(a, size);

    return EXIT_SUCCESS;
}

void initializeArray(int a[], int size) {
    int k;

    for (k = 0; k < size; ++k) {
        a[k] = 100;
    }
}
```

Declare the array : type and size. Allocate space, uninitialized. Size cannot change. Can initialize elements with: `int a[] = {…};`

Pass the array to a function – just the array name. Must also send size; no *len* function.

Get an array as a parameter – array name plus empty brackets. Must also send size; no *len* function.

Loop through array. Reference array elements like in Python – square brackets with index, indices start at 0. NO CHECK that references stay within the array!

Quiz: Write countEvens

```
int countEvens(int nums[], int size) {
    // Returns the count of even numbers in the nums   array.
    // TODO: complete this function…
    return count;
}

int main() {
    int SIZE = 7;
    int a[] = {16, 5, 23, 19, 42, 17, 12};

    int evens = countEvens(a, SIZE);
    printf("The number of even numbers is %i.\n", evens);
    return EXIT_SUCCESS;
}
```

Q4

Working with arrays

1. Checkout the ***Session26_Arrays*** project from SVN
2. Do its TODO's in the order listed.
   - Exception: ***Don't*** do `printArrayWithPointers` (TODO's 7 and 8) until we discuss pointer arithmetic.

Summary: You will see in doing the TODO's that you write and test functions that:

- Get input from the user and put the input into an array
- Print a portion (or all) of an array
- Return the number of even numbers in an array of integers.
- (Eventually) Print an array using pointer arithmetic.

Additionally, you will ***declare*** an array.

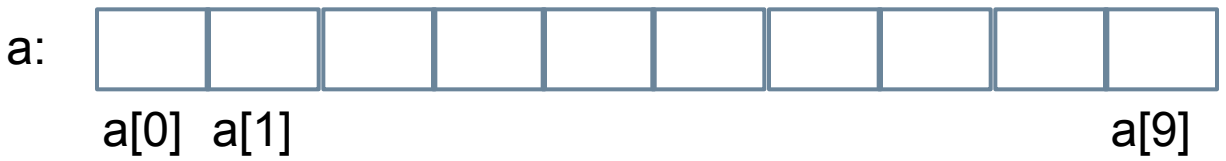
Arrays and Pointers

- In C there is a strong relationship between arrays and pointers
  - An array occupies a fixed location in memory
  - **Its address cannot be changed**
- Any operation that can be achieved by indexing (e.g., a[i]) can be done with pointers
- The pointer version will be
  - a bit **more challenging** to implement at first
  - but **faster** in some cases

How arrays and pointers relate

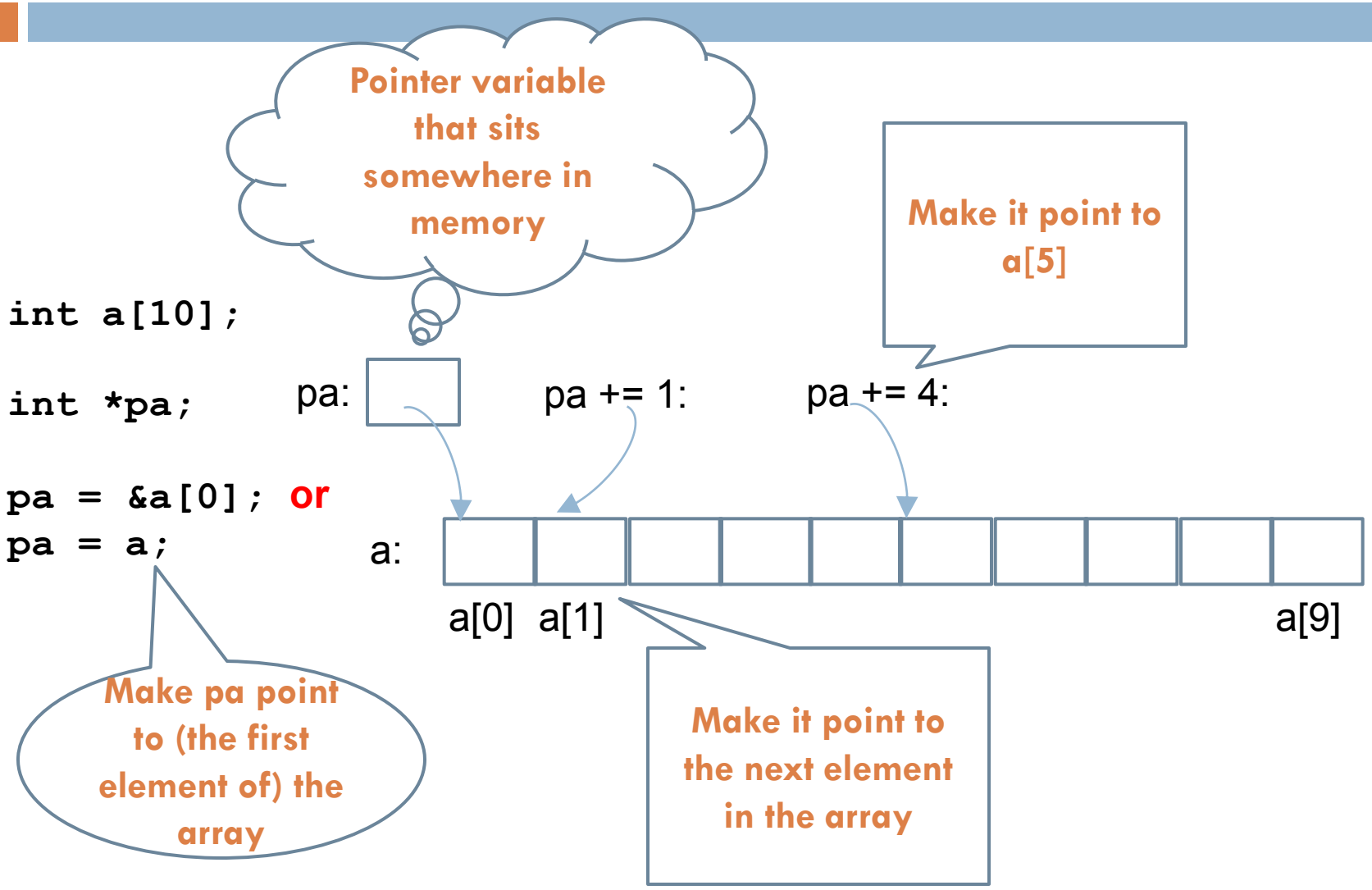
Each element in the array is accessed using the notation a[i] where i is the index of the ith element.

int a[10];



int a[10]; defines an array of size 10, i.e., a block of 10 consecutive integers named a[0], a[1], …, a[9]. a is really the starting address of the array.

How arrays and pointers relate



Q5

Summary of arrays and pointers

- `int *pa;` declares a pointer to an integer
- Set `pa` to point to array `a`
  - `pa = &a[0];` or `pa = a;` (your choice)
- Refer to array elements (given above assignment)
  - `a[0]` or `*pa` (your choice)
- Pointer arithmetic
  - Can increment pointers, so the following are equivalent:

```
                pa = &a[0];          pa = &a[0];
a[k]            *(pa + k)            pa = pa + k;
                                     *pa
```

Array notation vs. Pointer notation

```
void initializeArray(int a[], int size) {
    int k;

    for (k = 0; k < size; ++k) {
        a[k] = 100;
    }
}
```

```
void initializeArray(int *a, int size) {
    int *p;

    for (p = a; p < a + size; ++p) {
        *p = 100;
    }
}
```

Arrays as function parameters

- **int []** and **int *** are equivalent, when used as formal parameters in a function definition, e.g., …
  - **void f (int a[], int count) { …**
  - **void f (int *a, int count) { …**
- Note that in neither case can we know the size of the array, unless it is passed in as a separate parameter.
- In either case, the 6th element of **a** can be equivalently accessed as
  - a[5]
  - *(a+5) // treating array a as a pointer

Q6

Using pointers with arrays

- How do we modify printArray() so that it uses pointers instead of array indexing?
- Implement:

```
void printArrayThePointerWay(int *a, int size) {
    ...
}
```

Test the function by invoking it in **main()**, *exactly like you invoked the printArray function* (except changing the name, of course).

HW Warm-up: Thinking of a Sort

- Homework asks you to imagine you are a real estate agent who is helping potential home buyers to analyze the prices of homes in Vigo county.
- In order to analyze those prices you may need to sort the prices.
- Given:

```
double ratings[] = {2.4, 5.0, 4.4, 3.2, 0.1};
```

- What would we do to sort `ratings` in ascending order?

Selection Sort:

- Idea: Select the smallest and put it at the beginning of the array. Then select the 2nd smallest and put it at index 1 of the array. Etc.
- Algorithm:

```
for k from 0 to size - 2:
    j = index of smallest element in the array,
        starting at index k
    swap the array elements at indices j and k
```

- Back-of-the-envelope analysis: The k-loop goes about N times, where N is the size of the array. Each time through that loop, it does roughly N/2 chunks of work to find the index of the smallest remaining element. So the total work is roughly proportional to N^2. We write this as $O(N^2)$.
- Selection Sort is easy to understand and implement (good!). But it is MUCH slower than better sorting algorithms on large arrays – See the table and Wikipedia Sorting Algorithms for over 30 other choices!

This table assumes 10^6 chunks of work per second and makes various wrong assumptions, but it is fine for a back-of-the-envelope comparison.

N (size of array)	N^2 (selection sort)	N log N (better sorts)
1 thousand	1 second	< 1 second
1 million	278 hours	10 seconds
1 billion	317 centuries	3 hours