ARRAYS AND POINTERS IN C

CSSE 120 — Rose-Hulman Institute of Technology

Pointer review session

- What does the declaration of a variable do?
- What does the declaration of a pointer do?
- What is the significance of the '**address of**' operator?
- What is the dereferencing operator (*) and how do we use it?

Recap: Declarations Reserve Space

- Variable declarations reserve space in memory:
  - **int x;** /* *reserves enough space to hold an int, names it* **x** */
  - **double d;** /* *reserves enough space to hold a double, names it* **d** */
- Formal parameter declarations do the same:
  - **void average(double sum, int count) {…}**
  - /* *reserves enough space to hold a double (named* **sum**) *and an int (named* **count**)*/

Recap: Pointers Store Addresses

- We have seen that variables in C can hold ints, doubles and chars.
- In addition, they can hold memory addresses.
- We call those variables "pointers"
- Examples:
  - **int *xPtr;** /* *reserves enough space to hold an address, names it* **xPtr**, *says that xPtr can store the address of an* ***int*** *variable* */
  - **double *dPtr;** /* *reserves enough space to hold an address, names it* **dPtr**, *says that dPtr can store the address of a double variable* */

Recap: 'Address of' Operator, &

- The '*address* of' operator, **&**:
  - **&var** gives the memory location (or address) where **var**'s value is stored
  - Examples:
    - **xPtr = &x;** /* Read *"**xPtr** gets the address of **x**"* */
    - **dPtr = &d;** /* Read *"**dPtr** gets the address of **d**"* */

Binky says, "xPtr is a *pointer* and x is a *pointee*!" Thanks, Binky.

Recap: Pointer Operators, *

- Use * two ways:
  - In type declarations, * says that the name refers to address of something: **int *xPtr; double *dPtr;**
  - In expressions, ***var** gives the "thing" pointed to by **var**
  - Examples:
    - **printf("%d", *xPtr);**
    - ***dPtr = 3.14159;**

The format string, **"%d"**, says that we want to print an int. ***xPtr** is the thing pointed to by **xPtr**. That is, ***xPtr** is the value of **x**.

This says that the thing pointed to by **dPtr** should get the value **3.14159**. So the result is the same as **d = 3.14159**.

From the last homework:

- **swap**: a function to exchange the values of two variables
- Let's look at some approaches you may have tried and why they did or did not work…
  - Create a new Hello World ANSI C project in Eclipse

I think I need a list

- If you think about your room, there are several ways in which you organize your earthly possessions.
- Piles of stuff
- Bookcases
- File cabinets
- Boxes
- Drawers

I think I need a list (2)

- Think about bookcases. Most people have books, bookends, as well as various mementoes in their bookcases.
- You can easily access your items from the bookcase, because their order does not change
- You can add items to the bookcase
- You can remove items from the bookcase
- Your bookcase is like a list in Python
  - See the following example

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?
  - Type does not matter
  - They are dynamically allocated
  - Can be expanded or shrunk
  - Size not specified

Do we have lists in C

- Does the bookcase analogy work in C?
  - No because C is a more strongly typed language.
  - The size of the "bookcase" has to be specified and does not change
- A "*list*" in C is more like an egg cartoon
  - You can only add eggs to the egg cartoon
  - The size of the cartoon is fixed and does not change
  - A "*list*" in C is called an array
- Do we have lists in C?

List in Python vs Array in C

- No built-in lists in C
- Array is closest data structure to Python's list
- Consider this C code

```
int SIZE = 4;
int num[SIZE];
int x;
for(x = 0; x < SIZE; x++)
    num[x] = x * x;
```

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

Q2, 3

Initialization and access

- Consider the Python code that initialize lists
  - `>>> a = [1, 3, 5]`
  - `>>> b = [1, 3, 5]`
- How would that be done in C?
  - `int a[ ] = {1, 3, 5};`
  - `int b[ ] = {1, 3, 5};`
- How do we access an element?
  - Python list: `x = a[i]`
  - C array: `x = a[i];`

Array Practice in Pairs on Paper

```
int countEvens(int nums[], int m) {
   /* Returns a count of even numbers in nums,
    * an array of size m. */
   // TODO: complete this function…
   return count;
}
int main() {
   int a[] = {16, 5, 23, 19, 42, 17, 12};
   int evens = countEvens(a, 7);
   printf("The number of even numbers is %d.\n", evens);
   return 0;
}
```

Working with arrays

1. Checkout the ***ArraysAndRefs*** project from SVN
2. In function **main()** declare a variable, **scores**, to store an array of integers.
3. Implement the function **readScores()** that initializes an array of integers
4. Test the function by invoking it in **main()** and using function **printArray()** to print the values stored in the array
5. If time permits, also enter your **countEvens()** function from the quiz and test it

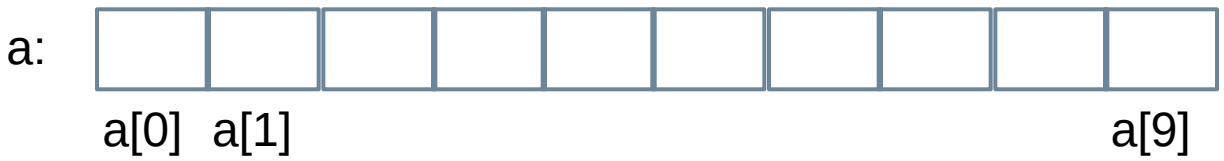
Arrays and Pointers

- 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
  - but faster in most cases

How arrays and pointers relate

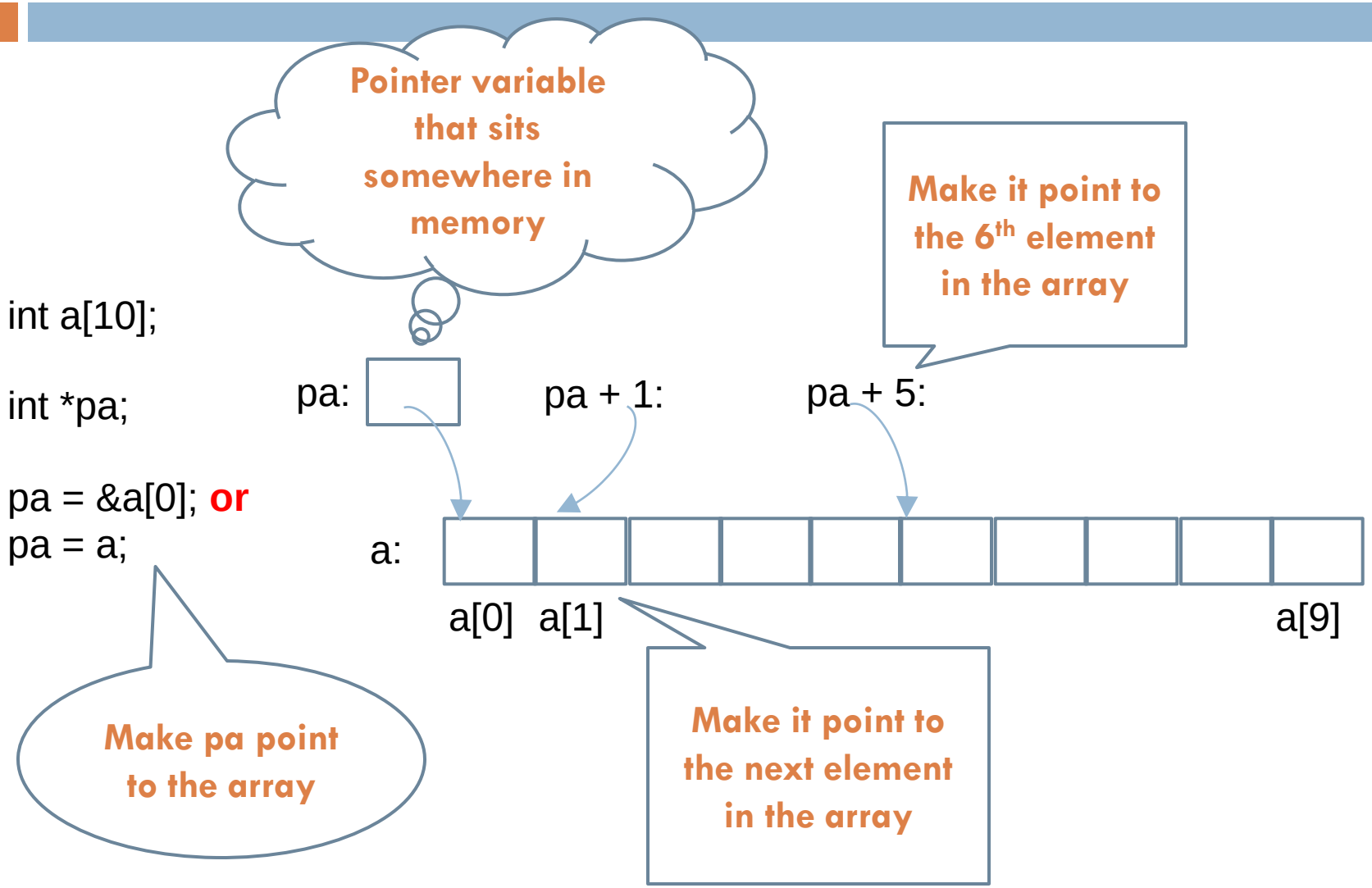
Each element in the array is accessed using the notation a[i] where i is the index of the ith element.

int a[10];



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

How arrays and pointers relate



Summary of arrays and pointers

- int *pa; declares a pointer to an integer
- Set pa to point to array a (a[0])
  - pa = a;
  - pa = &a[0];
- Copy the content of a[0] into x
  - int x = a[0];
  - int x = *pa

Summary of arrays and pointers (2)

- Point to the second element in the array
  - pa + 1
  - &a[1]
- Copy the content of a[1] into y
  - int y = a[1];
  - int y = *(pa + 1);

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)

Using pointers with arrays

- How do we modify printArray() so that it uses pointers instead of array indexing?
- Implement:
  - `void printArrayThePointerWay(int* a, int m) {…}`
- Test the function by invoking it in **main()**, like so:
  - `printArrayThePointerWay(scores, size)`

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?