ARRAYS AND POINTERS IN C

CSSE 120 — Rose-Hulman Institute of Technology

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: Variables with "Pointer Types" Store Addresses

- Besides holding "things" like ints and doubles, variables in C can also hold memory addresses
- Examples:
  - **int *xPtr;** /* *reserves enough space to hold an address, names it* **xPtr**, *says that xPtr can store the address of a variable that holds an int* */
  - **double *dPtr;** /* *reserves enough space to hold an address, names it* **dPtr**, *says that dPtr can store the address of a variable that holds a double* */

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);**

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**.

    - ***dPtr = 3.14159;**

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

Lists in Python

- Consider the following Python Code:
  - list = [1, “spam”, 4, “U”]
  - list.append(2)
  - list.remove(“U”)
  - list.pop(0)
- 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

List in Python vs Array in C

- No built-in lists in C
- Array is closest data structure to model 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: **x = a[i]**
  - C: **x = a[i];**

Array Practice in Pairs on Paper (if you please?)

```
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;
}
```

Q4

Working with arrays

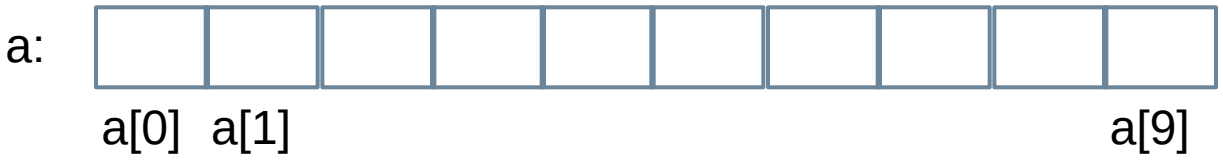
1. Check-out the project ***ArraysAndRefs***
2. In function **main()** declare a variable, **scores**, to store an array of integers.
3. Implement the function **readScores()** that initializes an array of integers
4. Test the function by invoking it in **main()** and using function **printArray()**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

Arrays and Pointers

- In C there is a strong relationship between arrays and pointers
- Any operation that can be achieved by array subscripting can be done with pointers
- The pointer version will be a bit more challenging to implement, but faster, in general

How arrays and pointers relate

```
int a[10];
```

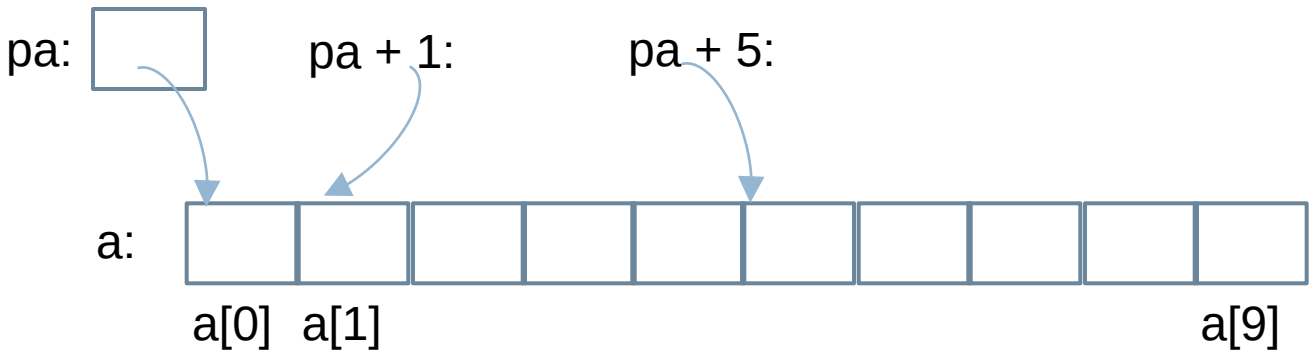


How arrays and pointers relate

```
int a[10];

int *pa;

pa = &a[0]; or
pa = a;
```



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, element 5 of **a** can be equivalently referred to as
  - a[5]
  - *(a+5)

Using pointers with arrays

- How do we modify printArray() so that it uses pointers instead of array subscripting?
- 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 sort an array of doubles
- Given:
  ```
  double ratings[] = {2.4, 5.0, 4.4, 3.2, 0.1};
  ```
- What would we do to sort **`ratings`** in ascending order?