PASSING REFERENCES TO SIMPLE VARIABLES, ARRAYS IN C

CSSE 120—Rose Hulman Institute of Technology

Can functions modify actual parameters?

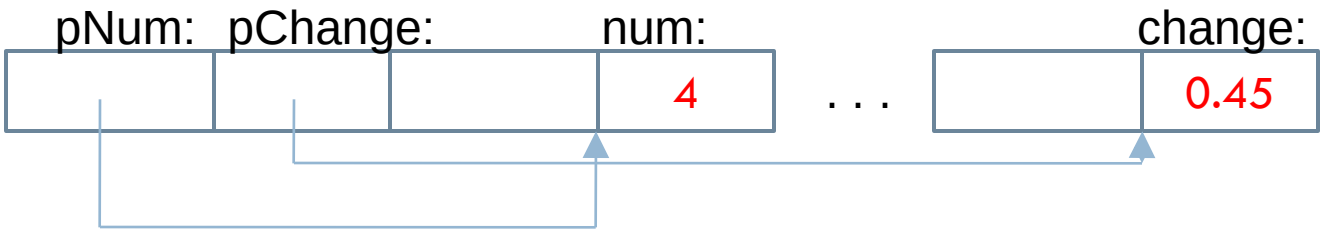
- In Python, no! (pass by value)
- Consider this function:

```
void downAndUp(int takeMeHigher, int putMeDown){
    takeMeHigher += 1;
    putMeDown -= 1;
 }
```

- How is this C function invoked?
  - downAndUp(up, down);
- Will calling the function change the values of the actual parameters **up** and **down**?

References to simple variables

- Pointers need to be used to change the value of an actual parameter
- Pointer is a variable that holds the address of a variable
  - int num = 4;
  - int *pNum;
  - pNum = #
  - *pNum == num == 4
  - double change = 0.45;
  - double *pChange;
  - pChange = &change
  - *pChange == change



Pass pointers to values to be changed

- How do we modify downAndUp() so that it actually changes the values of its parameters?
  - By passing pointers to the parameters to be changed
- Together, implement a function that passes pointers to values to be changed
  - Checkout project ArraysAndRefs from SVN
  - Implement **`downAndUpthatWorks()`**
  - Use function **`testdownAndUpthatWorks()`** to test **`downAndUpthatWorks()`**

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 different from lists in Python?

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[3] = {1, 3, 5};**
  - **int b[3] = {1, 3, 5};**
- How do we access an element?
  - Python: **x = a[i]**
  - C: **x = a[i];**

Comparing arrays for equality

- Consider the Python transcript:
  - `>>> a == b`
  - **True**
- What about the equivalent code in C?
  - **printf ("Array equality: %d\n", a == b);**
  - **Array equality: 0**
  - Recall **0** means false
  - Why is that the answer?

Array not totally different from list

```
def rotateFromLeft(alist):
    """ Rotates alist so that the leftmost
    element appears at the right end of the
    list """
    temp = alist[0]
    length = len(alist)
    for i in range(0, length -1):
        alist[i] = alist[i+1]
    alist[length - 1] = temp
```

Working with arrays

- In function main() declare a variable, **scores**, to store an array of integers.
- Implement function **readScores()** that initializes an array of integers
- Test the function by invoking it in main() and using function **`printArray()`** to print the values stored in the array

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 faster, in general
  - Harder to understand

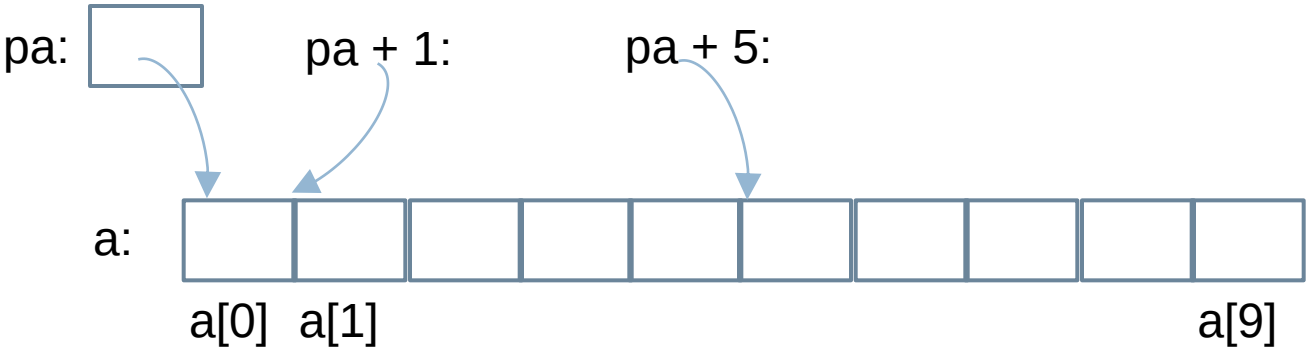
How arrays and pointers relate

int a[10];

See below instead

int *pa;

pa = &a[0];



Using pointers with arrays

- How do we modify printArray() so that it uses pointers instead of array subscripting?
- Implement **`printArrayWithPointers()`** a function that takes a pointer to an int as its parameter
- Test the function by invoking it in main()