**C++ (Pointers)**

Notes for the instructor

Important differences w/ Java/C#/dynamic languages

* Manual memory management vs. Garbage collection
* Raw memory access (pointer arithmetic, loose typing, etc) vs. opaque memory references
* Class defines a type (a 'memory stamp/format') and code then decides where to put it (in the heap, or in the stack/object/surrounding 'context') vs. Java/C# classes must be allocate in the heap
	+ B-Trees make a whole lot of sense in C++ because you can actually store everything the disk sector 'image'; BTrees make almost no practical sense in C#/Java because any arrays, strings, or objects won't be stored in the 'image' (although you could serialize them to/from disk)

Example:

int x;

int nums[5];

int \*nums = new int[3];

int cNums = 3; // point out the lack of .Length

* Emphasize that \* will mean two things – here it means declaration
* <Diagram this, stack & heap>
* Note that C++ does NOT initialize variables for you (either x, or the array contents)

For( int I = 0 ;I < 5; i++)

 Cout << Nums[i] << endl;

* Talk about lack of bounds-checking: over the top & negative indeces

Int \*ptr = nums;

// print example

For( int I = 0 ; I < cNums; i++)

{

 Cout << \*ptr << endl;

 Ptr++;

}

* Emphasize that HERE \* means 'derference' / follow the arrow
* ++ moves up by the size of an int, NOT by a single byte
* Show them the results of moving up by a single byte (last 3 bytes of current int plus first byte of next int)

Fade Guidance:

* Jointly create a 'sum up' program

EXERCISES:

* Find the min
* Find the max
* Print in reverse order
* Find a target value
* BubleSort
* Etc

For( int I = 0 ; I < cNums; i++)

{

 Cout << \*(ptr + i) << endl;

}

Fade Guidance:

* Jointly create a 'sum up' program

EXERCISES:

Same as for prior topic (use the same ones so they can scaffold AND see the differences)

Next topics:

* Equivalence of \*(ptr + i) and ptr[i]
* & operator
	+ Why you can't get free memory by taking the address of local variables