**Pointer Arithmetic**

// Array elements don't have pointers for each element, but the array itself has a pointer to the first element of the array

// Array are contiguous, which means each element of an array follows the proceeding one in memory

// The compiler knows the address location of each element without pointers because it uses pointer math to figure out the amount of memory needed to store the array

// If an int array holds 5 values, and the pointer of that array starts at memory location hex f000000(16), then because an int needs 4 blocks of memory for each element (20 blocks total), then the array will go from memory slots 16-36.

// To navigate the array, the compiler goes to the pointer address stored in the first spot of that array, and then adds 4 to each spot after that to return that location of the element requested.

#include "stdafx.h"

#include <iostream>

using namespace *std*;

int main()

{

int array[] = { 1, 2, 3, 4, 5 };

*cout* << "The memory address location of the array is at: " << array << *endl*;

*cout* << &(array[0]) << " -> " << \*(&(array[0])) << *endl*;

*cout* << &(array[1]) << " -> " << \*(&(array[1])) << *endl*;

*cout* << &(array[2]) << " -> " << \*(&(array[2])) << *endl* << *endl*;

// add 1 to array - compiler knows to add 4 blocks of memory to array. +1 adds one index's worth of memory versus add one place in memory.

*cout* << "Using bracket notation to get value \*(&(array[2] + 1): " << \*(&(array[2]) + 1) << *endl*;

// error - & operator requires a left hand value. array + 3 is a value but not a referenceable variable unless [] notation is used

// cout << &(array + 3) << endl;

*cout* << "Using de-reference with pointer math to get value \*(array+3): " << \*(array + 3) << *endl*;

return 0;

}