// Several examples of pointer dereferencing and incrementing
// George F. Riley, Georgia Tech, Spring 2012

#include <iostream>
using namespace std;

// First a global array for illustration
#define ASIZE 8
// Array a is the array used in most of the examples below
int a[ASIZE] = { 0, 1, 2, 3, 4, 5, 6, 7};
// Array b is used for the array copying loop below
int b[ASIZE] = { 10, 20, 30, 40, 50, 60, 70, 80};
int c = 100; // A global variable
int d = 200;

int main()
{
    int * pA = a; // pA is a pointer, pointing to array "a", element 0
    // See below showing that the pointer, pA, can be dereferenced
    // with the ’*’ operator, or with the indexing ’[]’ operator.
    cout << "pA is " << pA << " * pA is " << * pA
    " pA[0] is " << pA[0] << endl;
    // Note that the incrementing operator ’++’ has precedence over
    // the dereferencing operator ’*’. But keep in mind that the
    // VALUE of the expression pA++ is the value of pA BEFORE the
    // increment takes place. Thus the below should result in the
    // value 0 stored in j0 and 1 in j1;
    int j0 = * pA++;
    int j1 = * pA++;
    // j0 should be zero and j1 should be one
    cout << "j0 is " << j0 << " j1 is " << j1 << endl;
    // At this point, pA points to the ’2’ in array a. Try using
    // the pre-increment operator to see the difference.
    int j2 = ++pA;
    // This is tricky...what should j2 be here? The VALUE of the expression
    // ++pA is the INCREMENTED value of pA (which will then point to the
    // 3 in array a, so we expect j2 to be 3.
    cout << "j2 is " << j2 << endl;
    // Another try using parens. At this point pA points to the 3 in array a
    int j3 = (*pA)++;
    // Again tricky. Using parens, we said to evaluate "*pA" and then
    // post-increment the results. Evaluating *pA results in the
    // value 3 (what is pointed to by pA). The post-increment operator
    // evaluates to the value before the increment, so (*pA)++ evaluates
    // to 3. But, two important things. First, pA is UNCHANGED. Second,
    // the 3 in array a is changed to a 4.
    cout << "j3 is " << j3 << endl;
    // This is illustrated later.
    int j4 = (*pA)++;
    // j4 should be four, but pA still points to the address where the
    // original 3 was.
    cout << "j4 is " << j4 << endl;
    // One more try. What should j5 be below?
    int j5 = ++(*pA);
    cout << "j5 is " << j5 << endl;
    // Illustrate array copying using pointers
    // Reset pA back to beginning of array a

    return 0;
}
pA = a;  // pA points to the a array
int* pB = b;  // pB points to the b array

cout << "&c " << &c
   << " &d " << &d
   << " pB " << pB
   << " pA " << pA << endl;
for (int i = 0; i < ASIZE; ++i)
{  // copy a to b
    *pB++ = *pA++;
}

// Print out b
for (int i = 0; i < ASIZE; ++i)
{
    cout << b[i] << " ";
}

// What would we get if we dereferenced pB here?
cout << "*pB is " << *pB << endl;

// This last one is tricky...think about what should be printed here
pA = a;  // reset pA
cout << " first " << *pA++ << " second " << *pA++
   << " third " << *pA++ << " fourth " << *pA++
   << endl;

Program pointer-dereferencing.cc (continued)