/* factorial.cpp
 *
 * @author Cynthia Lee
 * @version 1920-Aut
 *
 * A little "hello world"-style first look at recursion. This program computes
 * the factorial of a non-zero integer input by the user.
 *
 * n! = { 1    if n = 1
 *      { n(n-1)!  otherwise
 *
 * We use this program to demonstrate that the way stack frames push and pop
 * with function call and return is the same for "normal" functions as for
 * recursive functions. This version includes a little "debugging"-style
 * output of the memory addresses where the 'n' parameter for each recursive
 * call stack frame are stored, to see the layout of stack frames in memory
 * matches what we learned about the memory model in lecture.
 */

#include <iostream>
#include "console.h"
#include "simpio.h"
using namespace std;

int myfunction(int x);
int factorial(int n);

/*
 * A little "hello world"-style first look at recursion.
 */

int main() {
  int x = getInteger("Enter an integer > 0: ");
  while (x > 0) {
    int answer = myfunction(x);
    cout << "The answer is: " << answer << endl;
    x = getInteger("Enter an integer > 0: ");
  }
  return 0;
}

/*
 * Demonstrates a "normal" (non-recursive) function call, which in turn
 * initiates a call to a recursive implementation of factorial calculation.
 */

int myfunction(int x) {
  int xfac = 0;
  xfac = factorial(x);
  return xfac;
}

/*
 * A simple recursive implementation of the factorial calculation.
 */

int factorial(int n) {
  // We will learn about this line later in the quarter, but just glancing
  // at this output we see that the larger the n (the earlier in the
  // recursive stack), the larger the memory address. This matches what we
  // saw in the diagrams in the lecture slides.
  cout << n << " is stored at memory address: " << &n << endl;
  if (n == 1) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}