**Recursion**

1. Write a recursive function for X to the power of N. Test it by prompting the user for the value of X and N, calling the function and printing out the result.

| /\*  \* Program name: RecursionXtotheN.java  \*  \* By: Lucas Chow (Last edited: 2022-10-3)  \*  \* ICS4U1: Recursion  \*  \* This program uses recursion to find x^n  \*  \*  \*/  public class RecursionXtotheN{  /\*  \* int power(int x, int n)  \*  \* This program returns int - the power of x^n  \*  \* This program uses recursion, calling itself to find x^n  \*  \* x -> base, n -> power  \*  \*/  public static int power(int x, int n)  {  if (n == 1)  {  return (x\*n);  }  else  {  return x \* power(x,n-1);  }  }  public static void main(String[] args)  {  System.out.println("The power of 5 ^ 8 is "+ power(5,8));  }  } |
| --- |

1. One of the most famous algorithms known was stated over two thousand years ago by the Greek mathematician Euclid. *Euclid’s algorithm* provides a method of finding the greatest common divisor (gcd) of a pair of natural numbers. The algorithm is based on the following properties of gcd:

**Rule 1:** If m = n, then gcd(m, n) = m

**Rule 2:** If m > n, then gcd(m, n) = gcd(n, m-n)

**Rule 3:** If m < n, then gcd(m, n) = gcd(n, m)

| /\*  \* int gcd(int num1, int num2)  \*  \* This method finds the gcd recursively using Euclid's Algorithm  \*  \* num1 -> first natural number  \*  \* num2 -> second natural number  \*  \*/  public static int gcd(int num1,int num2)  {  if (num1 == num2)  {  return num1;  }  else if (num1 > num2)  {  return gcd(num2,num1-num2);  }  else  {  return gcd(num2,num1);  }  } |
| --- |

1. Write a method that implements this definition of *square numbers*:

square(1) = 1

square(N) = square(N-1) + 2N – 1

Aside: where did this crazy definition of *square numbers* come from?

Easy: this is from simple algebra:

(N-1)2 = N2 - 2N + 1

rearrange to get:

N2 = (N-1)2 + 2N – 1

| /\*  \* int square(int inputN)  \*  \* This method uses the definition of square numbers, finding the square of a number recursively  \*  \* inputN -> the base, which it is multiplied by itself, or inputN^2  \*  \* int -> returns int, the the square of the inputN  \*  \*/  public static int square(int inputN)  {  if (inputN == 1)  {  return 1;  }  else  {  return square(inputN-1)+2\*inputN-1;  }  } |
| --- |

1. A *prime number* is an integer that cannot be divided by any integer other than one and itself. For example, 7 is a prime number because its only divisors are 1 and 7. The integer 8 is not a prime number because its divisors are 1, 2, 4 and 8.

Another way to define prime is

prime (N) = prime (N, N-1)

prime (N, 1) = true

prime (N, D) = false if D divides N, prime (N, D-1) otherwise

For example,

prime (4) = prime (4, 3)

prime (4, 3) = prime (4, 2)

prime (4, 2) = false

Another example,

prime(7) = prime(7,6)

prime(7,6) = prime(7,5)

prime(7,5) = prime(7,4)

prime(7,4) = prime(7,3)

prime(7,3) = prime(7,2)

prime(7,1) = true

Translate the math-like definition of prime into two Java methods that return boolean. Use the **%** operator to test divisibility. Put your method into a class, write a testing class, and test your program.

| /\*  \* Program name: PrimeNumberRecursion.java  \*  \* By: Lucas Chow (Last edited: 2022-)  \*  \* ICS4U1: Recursion  \*  \* This class uses recursion to find the  \*  \*  \*/  public class PrimeNumberRecursion{  //using method overloading to run both of these methods      /\*  \* \*\*This method is technically not recursion, but it has the same name of the following method  \*  \* boolean prime(int inputN)  \*  \* The first step to finding the prime, identifying the first number.  \*  \* inputN -> the number to check if it is prime  \*  \* This method is boolean to see if true/false if the number is prime.  \*  \* Returns prime(inputN, inputN-1), (Different Method, same name, using method overloading)  \*  \*  \*/  public static boolean prime(int inputN)  {  return prime(inputN, inputN-1);  }    /\*  \* prime(int inputN, int inputDivisor)  \*  \* inputN -> the number to check if it's prime  \*  \* inputDivisior -> the number to divide into inputN  \*  \* if the second parameter divides into first, the input is not prime.  \*  \* This method calls itself until inputDivisior = 1, therfore the number is prime  \*  \*  \*/  public static boolean prime(int inputN, int inputDivisor)  {  if (inputDivisor == 1)  {  return true;  }  else  {  if (inputN%inputDivisor == 0)  {  return false;  }  else  {  return prime(inputN, inputDivisor-1);  }  }  }    public static void main(String[] args)  {  //This number is NOT prime  System.out.println("Is 480 prime: "+prime(480));    //This number IS prime  System.out.println("Is 4919 prime: "+prime(4919));  }  } |
| --- |