**public** **class** TwoDPosition {

**public** **int** row, col;

**public** TwoDPosition(**int** r, **int** c)

{

row = r;

col = c;

}

**public** **boolean** isLegal(**int** N)

{ **return** row >= 0 && row < N && col >= 0 && col < N; }

}

**public** **class** BasicRecursion {

**public** **static** **boolean** IsPalindrome(String s)

{

**boolean** v;

**int** len = s.length();

**if** (len <= 1)

v = **true**;

**else**

v = (s.charAt(0) == s.charAt(len-1)) && *IsPalindrome*(s.substring(1, len-1));

**return** v;

}

**public** **static** **int** BinomialCoeficient(**int** n, **int** k)

{

**int** v;

**if** (k == 0 || k == n) {

v = 1;

} **else** {

v = *BinomialCoeficient*(n-1, k) + *BinomialCoeficient*(n-1, k-1);

}

**return** v;

}

**public** **static** **int** power(**int** x, **int** y)

{

**if** (y == 0)

**return** 1;

**else**

**return** x \* *power*(x, y-1);

}

**public** **static** String reverse(String str)

{

**if** (str.length() <= 1) {

**return** str;

} **else** {

**int** len = str.length();

**return** "" + str.charAt(len-1) + *reverse*(str.substring(1, len-1)) + str.charAt(0);

}

}

**public** **static** **int** fibonacci(**int** n) // n >= 1

{

**int** v;

**if** (n == 1 || n ==2)

v = 1;

**else**

v = *fibonacci*(n-1) + *fibonacci*(n-2);

**return** v;

}

// Lessen problems of recursive Fibonacci algorithm by using

// memoization.

**private** **static** **long** [] *memoizedFibs*;

**public** **static** **void** initMemoizedFibs(**int** MAXFIB)

{

*memoizedFibs* = **new** **long** [MAXFIB+1];

**for** (**int** i=1; i<=MAXFIB; i++) {

*memoizedFibs*[i] = -1;

}

}

**public** **static** **long** fibMemoized(**int** n)

{

**long** v = *memoizedFibs*[n];

**if** (v == -1) {

**if** (n == 1 || n == 2) {

v = 1;

} **else** {

v = *fibMemoized*(n-1) + *fibMemoized*(n-2);

}

*memoizedFibs*[n] = v;

}

**return** v;

}

**public** **static** **int** gcd(**int** a, **int** b)

{

**if** (b == 0)

**return** a;

**else**

**return** *gcd*(b, a%b);

}

**public** **static** **int** mystery1(**int** a) // a >= 0

{

**int** v;

**if** (a < 10)

v = 1;

**else**

v = 1 + *mystery1*(a/10);

**return** v;

}

**public** **static** **int** mystery2(**int** a)

{

**int** result;

**if** (a == 0) {

result = 1;

} **else** {

result = 0;

**for** (**int** i=0; i<=a; i++) {

result += *mystery1*(i-1);

}

}

**return** result;

}

**public** **static** **void** main(String args[])

{

**int** i;

Scanner input = **new** Scanner(System.***in***);

System.***out***.println(*IsPalindrome*("hannah"));

System.***out***.println(*IsPalindrome*("12345hannah54321"));

System.***out***.println(*IsPalindrome*("12345hanxnah54321"));

System.***out***.println(*IsPalindrome*("12345hanxinah54321"));

System.***out***.println(*IsPalindrome*("shannah"));

System.***out***.println(*IsPalindrome*("hah"));

System.***out***.println(*IsPalindrome*("X"));

System.***out***.println(*IsPalindrome*("Xx"));

System.***out***.println(*IsPalindrome*("XY"));

System.***out***.println(*IsPalindrome*(""));

System.***out***.println("press return");

String line = input.nextLine();

System.***out***.println("Factorial");

**for** (i=0; i<=10; i++)

System.***out***.println("" + i + "! = " + *factorial*(i));

System.***out***.println("press return");

line = input.nextLine();

System.***out***.println("Mystery");

**for** (i=0; i<=15; i++)

System.***out***.println("mystery1(" + i + ") = " + *mystery2*(i));

**for** (i=95; i<=105; i++)

System.***out***.println("mystery1(" + i + ") = " + *mystery2*(i));

System.***out***.println("press return");

line = input.nextLine();

**for** (**int** n=0; n<=15; n++) {

**for** (**int** k=0; k<=n; k++) {

System.***out***.print("(" + n + ' ' + k + ")=" + *BinomialCoeficient*(n, k) + ' ');

}

System.***out***.println();

}

System.***out***.println("press return");

line = input.nextLine();

System.***out***.println("Fibonacci");

**for** (i=1; i<=42; i++) {

System.***out***.println("" + i + " " + *fibonacci*(i) + ' ');

System.***out***.flush();

}

System.***out***.println();

line = line.trim();

input.close();

}

}

**public** **class** AdvancedRecursion {

// Enumeration of all binary strings

**private** **static** **void** binaryStrings(String prefix, **int** len)

{

**if** (len == 0) {

System.***out***.println(prefix);

} **else** {

*binaryStrings*(prefix + '0', len-1);

*binaryStrings*(prefix + '1', len-1);

}

}

**public** **static** **void** binaryStrings(**int** len)

{

*binaryStrings*("", len);

}

**public** **static** **void** anagrams2(String prefix, String remain)

{

**int** len = remain.length();

**if** (len == 0) {

System.***out***.println(prefix);

} **else** {

**for** (**int** i=0; i<len; i++) {

**char** ch = remain.charAt(i);

// eg, remain = "ABCDEF", i = 3 ---> s1 = "ABC" s2 = "EF"

String s1 = remain.substring(0, i);

String s2 = remain.substring(i+1);

*anagrams2*(prefix + ch, s1 + s2);

}

}

}

// This routine is the interface given to users. This routine

// has a parameter list that reflects how users should think

// about using this routine -- they should not have to understand

// how the algorithm works.

**public** **static** **void** anagrams(String str)

{

*anagrams2*("", str);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **void** anagrams2(String prefix, String remain, **int** k)

{

**int** len = remain.length();

**if** (prefix.length() == k) {

System.***out***.println(prefix);

} **else** {

**for** (**int** i=0; i<len; i++) {

**char** ch = remain.charAt(i);

String s1 = remain.substring(0, i);

String s2 = remain.substring(i+1);

*anagrams2*(prefix + ch, s1 + s2, k);

}

}

}

**public** **static** **void** anagrams(String str, **int** len)

{

*anagrams2*("", str, len);

}

// Enumeration of binary strings having a specific number of 1s

**private** **static** **void** ones(String pre, **int** len, **int** bits)

{

**if** (len == 1) {

**if** (bits == 0)

System.***out***.println(pre + '0');

**else** **if** (bits == 1)

System.***out***.println(pre + '1');

} **else** {

*ones*(pre + '0', len-1, bits);

*ones*(pre + '1', len-1, bits-1);

}

}

**public** **static** **void** ones(**int** len, **int** bits)

{

*ones*("", len, bits);

}

**public** **static** **void** deleteBlob(**int** im[][], **int** r, **int** c)

{

**int** N = im.length; // assume square

**if** (r >= 0 && r < N && c >= 0 && c < N && im[r][c] == 1) {

im[r][c] = 0;

*deleteBlob*(im, r-1, c-1);

*deleteBlob*(im, r-1, c);

*deleteBlob*(im, r-1, c+1);

*deleteBlob*(im, r, c-1);

*deleteBlob*(im, r, c);

*deleteBlob*(im, r, c+1);

*deleteBlob*(im, r+1, c-1);

*deleteBlob*(im, r+1, c);

*deleteBlob*(im, r+1, c+1);

}

}

**public** **static** **void** deleteBlobNonRecursive(**int** im[][], **int** r, **int** c)

{

**int** N = im.length;

Stack<TwoDPosition> stk = **new** Stack<>();

stk.push(**new** TwoDPosition(r, c));

**while** (!stk.isEmpty()) {

TwoDPosition pos = stk.pop();

**if** (pos.isLegal(N) && im[pos.row][pos.col] == 1) {

im[pos.row][pos.col] = 0;

stk.push(**new** TwoDPosition(pos.row+1, pos.col+1));

stk.push(**new** TwoDPosition(pos.row, pos.col+1));

stk.push(**new** TwoDPosition(pos.row-1, pos.col+1));

stk.push(**new** TwoDPosition(pos.row+1, pos.col));

//stk.push(new TwoDPosition(pos.row, pos.col));

stk.push(**new** TwoDPosition(pos.row-1, pos.col));

stk.push(**new** TwoDPosition(pos.row+1, pos.col-1));

stk.push(**new** TwoDPosition(pos.row, pos.col-1));

stk.push(**new** TwoDPosition(pos.row-1, pos.col-1));

}

}

}

**public** **static** **void** main(String args[])

{

System.***out***.println("ANAGRAMS");

*anagrams*("ABCD");

System.***out***.println("ANAGRAMS");

*anagrams*("AAB");

System.***out***.println("LENGTH LIMITED ANAGRAMS");

*anagrams*("ABCDEF", 3);

**int** image[][] = {{0,0,0,0,0,1,1},

{0,1,0,0,0,1,1},

{1,0,0,1,0,1,1},

{0,1,0,1,0,0,0},

{0,0,0,1,0,0,0},

{1,0,0,0,1,0,0},

{1,0,0,0,1,0,0}};

**int** r, c;

**final** **int** N = image.length; // assume image is square

**for** (r=0; r<N; r++) {

**for** (c=0; c<N; c++)

System.***out***.print(image[r][c]);

System.***out***.println();

}

System.***out***.println();

*deleteBlob*(image, 1, 1);

// deleteBlobNonRecursive(image, 1, 1);

**for** (r=0; r<N; r++) {

**for** (c=0; c<N; c++)

System.***out***.print(image[r][c]);

System.***out***.println();

}

System.***out***.println();

}

}