# Recursive Programs

Program 1 – Determine the factorial of a number.

The factorial of a number is: $n! = \prod_{k=1}^{n} k$

5! = 5 x 4 x 3 x 2 x 1 = 120

10! = 10 x 9 x 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 = 3628800

```
class FactorialNonRecursion
{
  public static void main(String[] args){
    int start = 5;
    int ans = 1;
    for(int i = start; i > 0; i--)
    {
      ans = ans*i;
    }
    System.out.println("The result of " + start +"!" + " is: " + ans);
  }
}

class FactorialRecursion
{
  public static void main(String[] args)
  {
    int start = 5;
    int ans = factorial(start);
    System.out.println("The result of " + start +"!" + " is: " + ans);
  }
  public static int factorial(int n){
    if (n == 1){
      return n;
    }
    return n * factorial(n-1);
  }
}
```

Recursive Programs

Program 2 – Printing a list from 0 to 99

```
class PrintNumbersNonRecursion{
  public static void main(String[] args)
  {
    for (int i =0; i< 100; i++){
     System.out.println(i);
    }
  }
}
class PrintNumbersRecursion{
  public static void main(String[] args)
  {
    printNumbers(0);
  }
  public static void printNumbers(int n)
  {
    if(n < 100){
      System.out.println(n);
      printNumbers(n+1);
    }
  }
}
```

# Recursive Programs

**Notes about Recursion**
Recursive functions are common in computer science because they allow programmers to write efficient programs using a minimal amount of code. The downside is that they can create infinite loops, which can cause the program to crash, or worse yet, hang the entire computer system.

Recursion should be used with a function that has a small number of parameters, since the recursion places a new occurrence of the function on the stack, along with those variables. It is quite possible for a function to recall itself upwards of thousands of times. Therefore, you could easily run out of memory in the stack when running your recursive program.

**Practice Problems**

**Directions:** Study the following recursive methods and answer the questions that follow. Do these practice exercises on paper first and then test them on a computer. You'll learn more doing them by hand.

```
int factorial(int n) {
   if (n <= 1)
      return 1;
   else
      return (n * factorial(n-1));
}


int sum1toN(int n) {
   if (n < 1)
      return 0;
   else
      return (n + sum1toN(n-1));
}


int add(int i, int j) { // assumes i >= 0
   if (i == 0)
      return j;
   else
      return add(i-1, j+1);
}

int fib(int n) {  // assumes n >= 0
   if (n <= 1)
      return n;
   else
      return (fib(n-1) + fib(n-2));
}
```

## Recursive Programs

1. Write a recursive method:

   ```
   public static int countA(String s)
   ```

   that counts the number of occurrences of the character 'a' in a string.

2. Write a recursive method:

   ```
   public static String cheerRepeat (String s, int i)
   ```

   that takes as parameters a String s and an integer i and returns a String that has s repeated i times. For example, if the given string is "Go bears! " and the integer is 3 then the return value would be "Go bears! Go bears! Go bears! ". (Note that if the integer is 0, then the empty string "" should be returned.)

3. Write a recursive method:

   ```
   public static String cheerRepeatExpGrowth(String s, int i)
   ```

   that takes as parameters a String s and an integer i and returns a String that has s repeated 2^i times. For example, if the given string is "Go bears! " and the integer is 3 then the return value would be
   "Go bears! Go bears! Go bears! Go bears! Go bears! Go bears! Go bears! Go bears!".
   Do not use multiplication or exponentiation in your algorithm. Just double the length of the string i times.

4. Write a recursive method:

   ```
   public void numbers (string prefix, int levels)
   ```

   The function prints output that consists of the string prefix followed by "section numbers" of the form 1.1., 1.2., 1.3., and so on. The levels argument determines how many levels the section numbers have may. For example, if levels is 2, then the section numbers have the form x.y. If levels is 3, then section numbers have the form x.y.z. The digits permitted in each level are always '1' through '9'. As an example, if prefix is the string "Section" and levels is 2, then the function would start by printing:

   ```
   Section 1.1.
   Section 1.2.
   Section 1.3.
   ```

   and end by printing:

   ```
   Section 9.7.
   Section 9.8.
   Section 9.9.
   ```

5. "Draw Some Images"
   Write the recursive function

```
public static void drawImage(int n)
```

In this question you have to generate a special pattern using recursion. You will see that recursion helps in drawing graphical images that have recursive structure. Consider the following pattern of asterisks and blanks, and a program that can generate such patterns, according to a given parameter n.

| | |
|---|---|
| *<br>* *<br>*<br>**n=2** | *<br>* *<br>*<br>* * * *<br>*<br>* *<br>*<br>**n=4** |
| *<br>* *<br>*<br>* * * *<br>*<br>* *<br>*<br>* * * * * * * *<br>*<br>* *<br>*<br>* * * *<br>*<br>* *<br>*<br>**n=8** | |

The longest line has n asterisks and is a power of two. For example, n might be 2, 4, 8, 16, etc… Between any two adjacent asterisks there is a blank. Check your solution for n not exceeding 32, as the length of a line of the screen is limited.