Suppose we want to compute the value of $\sum_{i=1}^{n} i$ for a given value of n. A simple for loop for this can be written as,

```
int n = 10;
int sum = 0;
int i = 0;
for( i =1; i <= n; i++)
{
      sum =sum + i;
}
```

we can write above *for* loop also as shown below.

```
for( i =1; i <= n; sum = sum + i++);
```

In this *for* loop last expression of *for* loop will be interpreted as follows after putting brackets :

$$[sum \ = \ [sum \ + \ [i++]]] \, ;$$

This statement has ++ as highest precedence operator. So $i++$ evalutes to i and this evaluated value is added to *sum*. = operator evaluates to $sum + i$.

Above expression has two side effects :

1) i is changed to $i + 1$

2) *sum* is changed to $sum + i$

Above *for* loop also can be written in more compact form.

```
for( sum = 0, i =1; i <= n; sum = sum + i++);
```

, operator is used to seperate multiple statements in loop expressions. Its effect is evaluate first statement then second and so on. , can be used only in first (*initializer expression*) and in last (*counting expression*). However, second expression(loop test) **can not** contain more than one statement. If we want to test multiple conditions then we have to use logical operators(*e.g. or, and*) .

Let us try to use a method for computing the $\sum_{i=1}^{n} i$.

```
int sum_natural( int n)
{
      int sum = 0, i;
      for(i = 1; i <= n; i++)
            sum = sum + i;
      return sum;
}

public static void main (String a[])
{
      System.out.println(sum_natural(10));
}
```

In main method we have one statement. This contains () operator which has highest precedence among all operators. This is used for function calls. We have two function calls in this statement. These functions are

1) System.out.println provided by *Java*

2) sum_natural

Statement will be interpreted as as follows :

$$[System.out.println\ [\ (sum\_natural\,[(10)])]\,;$$

Innermost () tells to evaluate the argument, which is 10. Now use this value 10 to call the function .

Function call $sum\_natural(10)$

Create a box named n and put the evaluated argument value (here 10) in that box. Now function will execute for loop and return a value 55. This value 55 is of type *integer* and println expects a string argument. 55 is implicitaly converted to string "55" .

Can we express that sum as an expression ?

$$\sum_{i=1}^{n} i \;=\; n \;+\; \sum_{i=1}^{n-1} i$$

$$sum(\ n\ ) \;=\; n \;+\; sum\ (n-1);$$

Let us compute value of $sum\ (2)$.

$$\begin{aligned} sum(2) &= 2 \;+\; sum(1) \\ &= 2 \;+\; (1 \;+\; sum(0)\ ) \\ &= 2 \;+\; (1 \;+\; (\ 0 \;+\; sum(-1)\ ) \\ &= \ ... \end{aligned}$$

The problem with above expression is that it never stops. We have to define the function in such a way that it stops when input reaches 0. So this can be defined as,

$$sum(n) \;=\; \begin{cases} n \;+\; sum(\ n-1\ ) & if\ \ n \geq 1 \\ 0 & if\ \ n \leq 0 \end{cases}$$

Now this evaluation will be complete as we evaluate $sum(0)$ as 0.

The method can be written as,

```
int sum_natural( int n)
{
        if(n <= 0)
                return 0;
        else
                return (n + sum_natural(n - 1));
}
```

This approach is called *Recursion.*

If a larger expression can be expressed in terms of smaller expression then we can use ***Recursion*** to solve such expressions. *e.g.* $factorial$

$$factorial(n) = \begin{cases} n * factorial(n-1) & if \;\; n >= 2 \\ 1 & if \;\; n <= 1 \end{cases}$$

Any problem that can be solved with loop can be solved using recursion and vice-versa.

Now consider the *return* statement from method sum_natural,

$$return\ (n + sum\_natural(n-1));$$

after bracketing,

$$return\ ([n + [sum\_natural([n-1])]]);$$

Execution of $sum\_natural(10)$:

we have to evaluate $10 + sum\_natural(9)$. To evaluate this we have to evaluate $sum\_natural(9)$ first. This requires to evaluate $9 + sum\_natural(8)$ which requires evaluation of $sum\_natural(8)$. This evaluation goes till evaluation of $sum\_natural(1)$ which requires evaluation of $1 + sum\_natural(0)$. $sum\_natural(0)$ evaluates to value 0 and now $sum\_natural(1)$ is evaluated to 1. Now evaluation of $sum\_natural(2)$ completes and evaluation completes for 3,4,5,.. . In end we evaluate $sum\_natural(9)$ as 45 and $sum\_natural(10)$ evaluates to 55.

Example of recursion : Find the all factors of a given number n.

```
public class factors
{
        static void doFactors(int f, int n)
        {
                //check if f is a factor of n
                if(n%f == 0)
                {
                        System.out.println(f+"is a factor");
                }
```

```
        if(f<=n)
            doFactors(f+1,n);
    }

    public static void main(String args[])
    {
        int n;
        n = 40;
        doFactors(1,n);
    }
}
```