Recursion - I

RECURSION

Recursion is a powerful problem-solving strategy. Solves large problems by reducing them to smaller problems of the **same form.** The subproblems have the same form as the original problem.

To illustrate the basic idea, imagine you have been appointed as funding coordinator for a large charitable organization. Your job is to raise one million dollars in contributions to meet the expenses of the organization. If somebody can write out a check for the entire amount, your job is easy. On the other hand, it may not be easy to locate persons who would be willing to donate one million dollars.

However people don't mind donating smaller amounts. If lets say $100 is a good enough amount, than all you have to do is to call 10,000 friends to complete the task. Well, it is going to be a tall order for you, but you know that your organization has a reasonable supply of volunteers across the country.

You may start off by finding 10 dedicated supporters in different parts of the country and appoint them regional coordinators. So each person now has to raise $100,000. It is simpler than raising one million, but hardly qualifies to be easy.

Maybe they can adopt the same strategy, i.e. delegate parts of the job to 10 volunteers each within their region and asking each to raise $10,000. The delegation process can continue until there are volunteers who have to go around raising donations of $100 each from individual donors.

The following structure is a psuedocode for the problem: Ask funding coordinator to collect fund

```
void collect (int fund)
{
        if ( fund <=100) {
                contact individual donor.
        } else {
                find 10 volunteers.
                Get each volunteer to collect fund/10 dollars
                Pool the money raised by the volunteers.  }
}
```

Notice that the basic nature of the problem remains the same, i.e. collect n dollars, where value of n is smaller each time it is called. To solve the problem you can invoke the same function again.

Having a function to call itself is the key idea of **recursion** in the context of programming. A structure providing a template for writing recursive functions is as follows:

```
If  (test for a simple case) {
                Compute simple solution without recursion

} else  {
                break the problem into similar subproblems
                solve these by calling  function recursively.
                Reassemble the solution to the subproblems
```

Recursion is an example of divide-and-conquer problem solving strategy. It can be used as an alternative to iterative problem solving strategy.

Example : Let us consider the Factorial Function

n! = n * (n-1) * (n-2) * … * 2 * 1

0! = 1

Iterative solution:

```
int fact(int n)
{
      int p, j;
      p = 1;
      for ( j=n; j>=1; j--)
      p = p* j;
      return ( p );
}
```

Recursive definition:

In the recursive implementation there is no loop. We make use of an important mathematical property of factorials. Each factorial is related to factorial of the next smaller integer :

*n! = n * (n-1)!*