**RECURSION**

**Recursive Definition**

A **recursive definition** of a set always consists of three distinct clauses:

1. The **basis clause** (or simply basis) of the definition establishes that certain objects are in the set. This part of the definition specifies the "seeds" of the set from which the elements of the set are generated using the methods given in the inductive clause. The set of elements specified here is called **basis** of the set being defined.
2. The **inductive clause** (or simply induction) of the definition establishes the ways in which elements of the set can be combined to produce new elements of the set. The inductive clause always asserts that if objects are elements of the set, then they can be combined in certain specified ways to create other objects. Let us call the objects used to create a new object the **parents** of the new object, and the new object is their **child** .
3. The **extremal clause** asserts that unless an object can be shown to be a member of the set by applying the basis and inductive clauses a finite number of times, the object is not a member of the set.   
   The set you are trying to define recursively is **the set that satisfies those three clauses.**

Examples of Recursive Definition of Set

**Example 1.** Definition of the Set of Natural Numbers tex2html_wrap_inline51  
  
The set ***N*** is the set that satisfies the following three clauses:   
  
**Basis Clause:** tex2html_wrap_inline53  
**Inductive Clause:** For any element ***x*** in tex2html_wrap_inline51, ***x* + *1*** is in tex2html_wrap_inline51.   
**Extremal Clause:** Nothing is in tex2html_wrap_inline51unless it is obtained from the Basis and Inductive Clauses.   
  
The basis for this set ***N*** is **{ *0* } .** The ***x* + *1*** in the Inductive Clause is the child of ***x***, and ***x*** is the parent of ***x* + *1***.   
Following this definition, the set of natural numbers ***N*** can be obtained as follows:   
First by the Basis Clause,   ***0*** is put into ***N***.   
Then by the Inductive Clause, since ***0*** is in ***N***,  ***0 + 1 (= 1)*** is in ***N***. ***0*** is the parent of ***1***, and ***1*** is the child of ***0***.   
Then by the Inductive Clause again,   ***1 + 1 (= 2)*** is in ***N***. ***1*** is the parent of ***2***, and ***2*** is the child of ***1***.   
Proceeding in this manner all the "natural numbers" are put into ***N***.   
  
**Example 2.** Definition of the Set of Nonnegative Even Numbers tex2html_wrap_inline65  
  
The set ***NE*** is the set that satisfies the following three clauses:   
  
**Basis Clause:** tex2html_wrap_inline67  
**Inductive Clause:** For any element ***x*** in tex2html_wrap_inline65, ***x* + *2*** is in tex2html_wrap_inline65.   
**Extremal Clause:** Nothing is in tex2html_wrap_inline65unless it is obtained from the Basis and Inductive Clauses.

## Recursive Definition of Function

Some functions can also be defined recursively.   
  
**Condition:** The domain of the function you wish to define recursively must be a set defined recursively.   
**How to define function recursively:** First the values of the function for the basic elements of the domain are specified. Then the value of the function at an element, say ***x***, of the domain is defined using its value at the parent(s) of the element ***x***.   
  
A few examples are given below.   
They are all on functions from integer to integer except the last one.   
  
**Example 1:** The function ***f(n*) = *n!*** for natural numbers ***n*** can be defined recursively as follows:   
  
The function ***f*** is the function that satisfies the following two clauses:   
  
**Basis Clause:** ***f(0) = 0! = 1***   
**Inductive Clause:** For all natural number ***n*,  *f(n+1) = (n+1) f(n)*.**   
  
Note that here Extremal Clause is not necessary, because the set of natural numbers can be defined recursively and that has the extremal clause in it. So there is no chance of other elements to come into the function being defined.   
  
Using this definition, ***3!*** can be found as follows:   
Since ***0 ! = 1***,   ***1 ! = 1 \* 0 ! = 1 \* 1 = 1* ,**   
Hence ***2 ! = 2 \* 1 ! = 2 \* 1 = 2* .**   
Hence ***3 ! = 3 \* 2 ! = 3 \* 2 \* 1 = 6* .**   
  
**Example 2:** The function ***f(n) = 2n + 1*** for natural numbers ***n*** can be defined recursively as follows:   
  
The function ***f*** is the function that satisfies the following two clauses:   
  
**Basis Clause:** ***f(0) = 1***   
**Inductive Clause:** For all natural number ***n*,  *f(n+1) = f(n) + 2* .**   
See above for the extremal clause.   
  
**Example 3:** The function ***f(n)* = *2n*** for natural numbers ***n*** can be defined recursively as follows:   
  
The function ***f*** is the function that satisfies the following two clauses:   
  
**Basis Clause:** ***f(0) = 1***   
**Inductive Clause:** For all natural number ***n*,  *f(n+1) = 2 f(n)* .**   
See Example 5 for the extremal clause.

## Recursive Algorithm

A **recursive algorithm** is an algorithm which calls itself with "smaller (or simpler)" input values, and which obtains the result for the current input by applying simple operations to the returned value for the smaller (or simpler) input. More generally if a problem can be solved utilizing solutions to smaller versions of the same problem, and the smaller versions reduce to easily solvable cases, then one can use a recursive algorithm to solve that problem. For example, the elements of a recursively defined set, or the value of a recursively defined function can be obtained by a recursive algorithm.   
  
If a set or a function is defined recursively, then a recursive algorithm to compute its members or values mirrors the definition. Initial steps of the recursive algorithm correspond to the basis clause of the recursive definition and they identify the basis elements. They are then followed by steps corresponding to the inductive clause, which reduce the computation for an element of one generation to that of elements of the immediately preceding generation.   
  
In general, recursive computer programs require more memory and computation compared with iterative algorithms, but they are simpler and for many cases a natural way of thinking about the problem.   
  
**Example 1:** Algorithm for finding the ***k***-th even natural number   
Note here that this can be solved very easily by simply outputting ***2*\*(*k - 1*)** for a given ***k* .** The purpose here, however, is to illustrate the basic idea of recursion rather than solving the problem.   
  
**Algorithm 1:   Even(**positive integer ***k*)**   
**Input: *k*** , a positive integer   
**Output: *k***-th even natural number (the first even being ***0***)   
  
**Algorithm:**   
**if *k* = *1*,** **then** return ***0*;**   
**else** return **Even(*k-1*) + *2* .**   
  
Here the computation of **Even(*k*)** is reduced to that of **Even** for a smaller input value, that is **Even(*k-1*). Even(*k*)** eventually becomes **Even(*1*)** which is ***0*** by the first line. For example, to compute **Even(*3*)**, **Algorithm** **Even(*k*)** is called with ***k* = *2***. In the computation of **Even(*2*)**, **Algorithm** **Even(*k*)** is called with ***k* = *1*.** Since **Even(*1*) = *0, 0*** is returned for the computation of **Even(*2*)**, and **Even(*2*)** = **Even(*1*) + *2* = *2*** is obtained. This value ***2*** for **Even(*2*)** is now returned to the computation of **Even(*3*)**, and **Even(*3*) = Even(*2*) + *2* = *4*** is obtained.

**Example 2:** Algorithm for computing the ***k***-th power of ***2***   
  
**Algorithm 2   Power\_of\_2(**natural number **k)**   
**Input: *k*** , a natural number   
**Output: *k***-th power of ***2***   
  
**Algorithm:**   
**if** ***k* = *0***, **then** return ***1***;   
**else** return **2\*Power\_of\_2(*k - 1*) .**   
  
The next example does not have any corresponding recursive definition. It shows a recursive way of solving a problem.

**Example 3:** Recursive Algorithm for Sequential Search   
  
**Algorithm 3   SeqSearch(*L, i, j, x*)**   
  
**Input:** ***L*** is an array, ***i*** and ***j*** are positive integers, ***i*** **http://www.cs.odu.edu/%7Etoida/nerzic/content/symbols_sets/leq.gif*j***, and ***x*** is the key to be searched for in ***L***.   
**Output:** If ***x*** is in ***L*** between indexes ***i*** and ***j***, then output its index, else output ***0***.   
  
**Algorithm:**   
**if** ***i*** **http://www.cs.odu.edu/%7Etoida/nerzic/content/symbols_sets/leq.gif*j* ,** **then**   
{   
   **if** ***L*(*i*) = *x***, **then** return ***i* ;**   
   **else** return **SeqSearch(*L, i+1, j, x*)**   
}   
**else** return ***0*.**

**Exercises**

**1.** Find   f(1),  f(2),  and   f(3),   if   f(n) is defined recursively by   f(0) = 2   and   for n = 0, 1, 2, ...

    a)  f(n + 1) = f(n) + 2.

    b)   f(n + 1) = 3f(n).

    c)   f(n + 1) = 2f(n).

**2.** Find  f(2),  f(3),  and   f(4),   if   f(n) is defined recursively by   f(0) = 1, and  for n= 1, 2,...

    a) f(n + 1) = f(n) + 3f(n - 1).

    b) f(n + 1) = f(n)2 f(n - 1).

**3.** Let F be the function such that **F(n**) is the sum of the first **n** even positive integers.  Give a recursive definition of **F(n)**.

**4.** Give a recursive algorithm for computing **nx** whenever **n** is a positive integer and x is an integer.

**5.** Give a recursive algorithm for finding the sum of the first **n** odd positive integers.

**6**. Fibonacci series:

Where n0 = 0, and n1= 1

You have to find the nth number from the fibonacci series.

Find the basic clause and indicative clause and after that develop the program.

7. Use your previous binary search pseudo-code and implement it.

8. Implement Merge sort

