Stacks and Recursion

- Recursion is a strange kind of loop formed by a function calling itself (directly or indirectly), often with different actual parameters with each recursive call
- We avoid infinite recursion by having some special terminating condition

```
int factorial ( int n )  // precondition: n >= 0
{
    if (n <= 1) return 1;
    else return (n * factorial(n-1) );
}
```

jmsamaniego@up1b.edu.ph (revised 2009)

Iteration = Recursion

- Any iterative function can be implemented recursively, and any recursive function can be implemented iteratively, but one version is often more natural than the other
- Many older languages do not offer recursion, e.g., assembly languages, COBOL, classic FORTRAN, classic BASIC
- Some programming languages have recursion as their main sequence control structure, e.g., LISP/Scheme

jmsamaniego@up1b.edu.ph (revised 2009)

Computing x^n = x*x*x...*x (n times)

jmsamaniego@up1b.edu.ph (revised 2009)

Iterative version

```
float iraise ( float x, int n )
{
    float   r = 1.0;
    int j;
    for ( j=0; j<n; j++) {
        r = r * x;
    }
    return r;
}
```

Recursive version

```
float rraise ( float x, int n )
{
    if ( n == 0 )
        return  1.0;
    else
        return  x * rraise(x, n-1);
}

// note: more efficient
    algorithms are possible
```

Recursive algorithms are natural for recursive structures

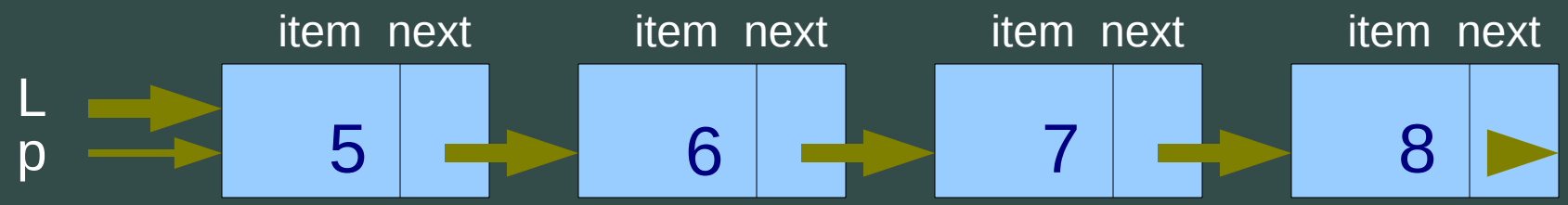
- To print all the elements in a linked-list

Iterative version

```
float iprintall ( node  *L )
{
    node *p;
    for (p=L; p!=NULL; p=p->next)  {
      printf(" %d ", p->item);
    }
}
```

Recursive version

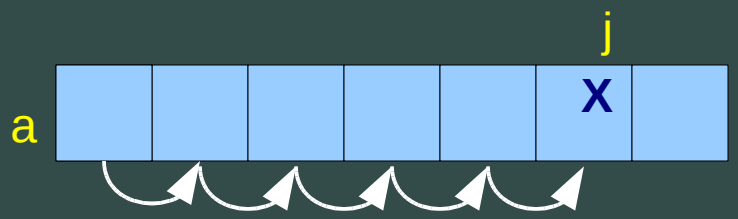
```
float rprintall ( node  *p )
{
   if (p != NULL)  {
     printf(" %d ", p->item);
     rprintall ( p->next );
   }
}
```



Towards a better search algorithm

- Given an array a[ ] of n unique items, and a target item x, find the position or index of x in the array; if x is absent in the array, return -1
- Recall the sequential search algorithm may require searching every item in the array

```
int  seqsearch ( int n, int a[ ], int x )
{
    int j;
    for (j=0; j<n; j++) {
        if ( a[j] == x )  return j;  // x found at position j
    }
    return -1;   // x not found in any of the positions
}
```



jmsamaniego@up1b.edu.ph (revised 2009)

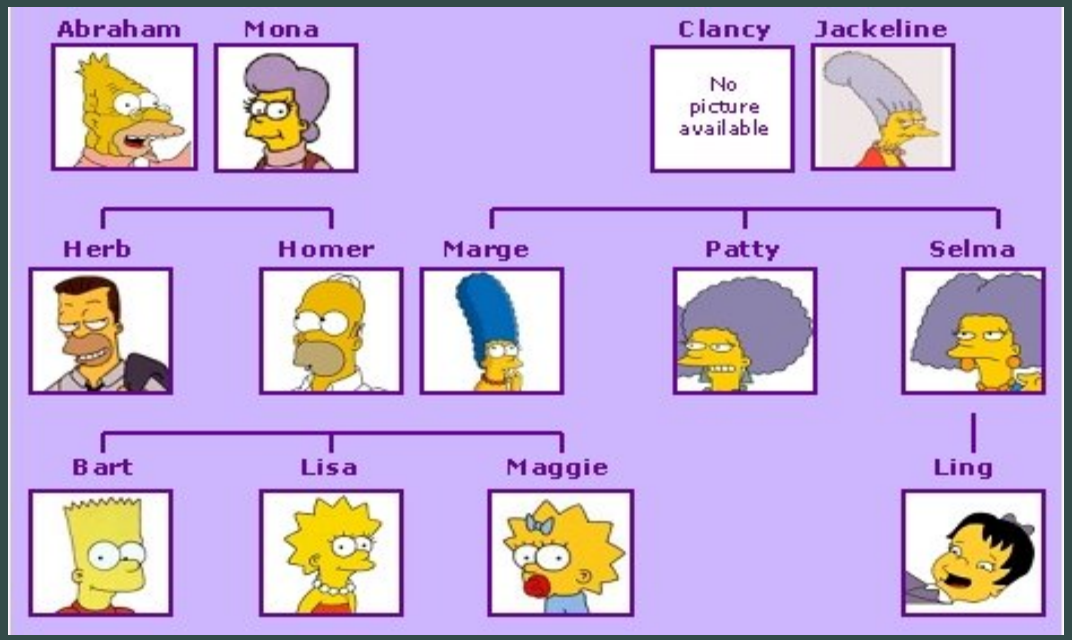
If the array is sorted, use binary search

- Idea: Check the middle position; if x is at the middle position, then we are done, else we search in the left half or in the right half, depending on whether x is less or greater than the middle value

```
int  binarysearch( int low, int high, int a[ ], int x )
{   // precondition: array a[ ] is in ascending order
    if (low > high) return -1; // x is absent
    else {
        int  mid = (low+high)/2;
        if ( x == a[mid] ) return mid;  // found x
        else if ( x < a[mid] ) return binarysearch(low,mid-1, a, x);
        else /* x > a[mid] */ return binarysearch(mid+1,high, a, x);
} // note: only about log n probes are necessary to search for x
```

jmsamaniego@up1b.edu.ph (revised 2009)

Trees as recursive structures



jmsamaniego@up1b.edu.ph (revised 2009)