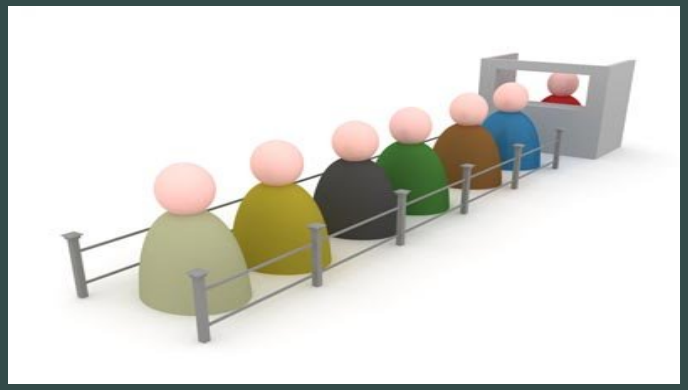
Dynamic Data Structures

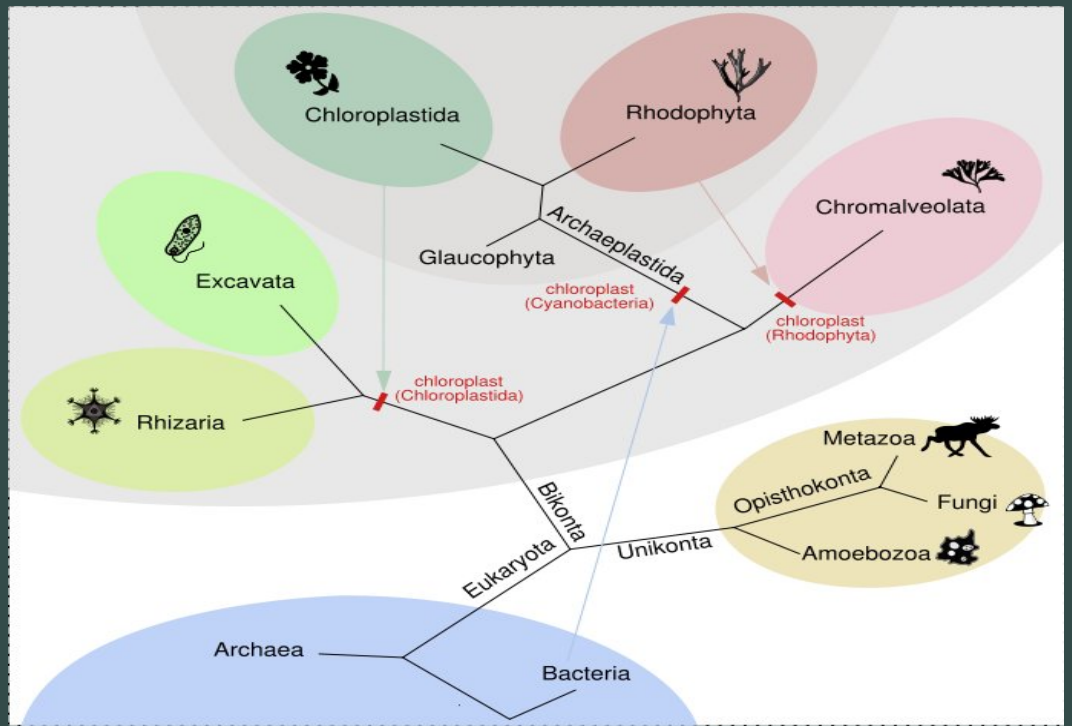
Data structures that frequently change their size or shape

- Stacks, queues, dynamic lists
- Trees and other hierarchical structures
- Graphs and other network-like structures

jmsamaniego@up1b.edu.ph (revised 2009)



www.ibiza-spotlight.com



commons.wikimedia.org

Queues

- First-in-first-out policy
- Additions at the rear only, deletions at the front only
- Can use a simple array to represent the items in a queue, but additions/deletions will occasionally require lots of data moves – think of people moving through a queue at a cafeteria
- A “circular array” is another representation – think of a guard handing out service numbers in a bank
- Used in job queues, printer queues, etc

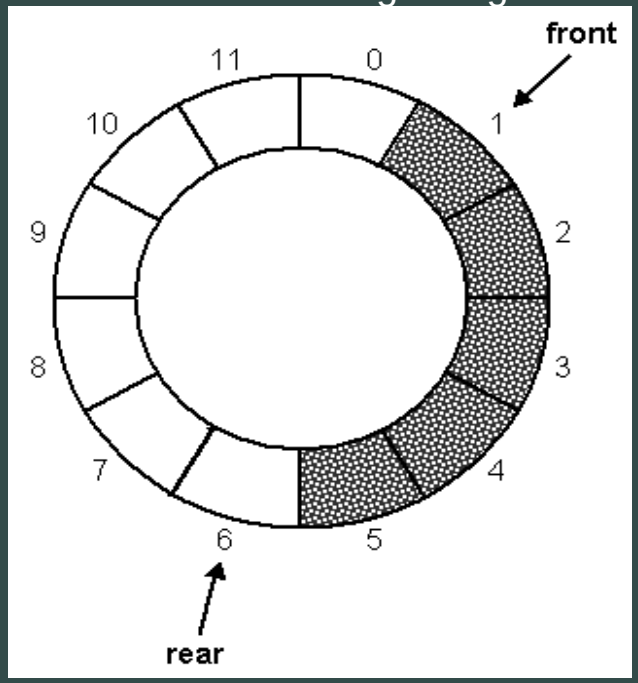
rear

jmsamaniego@up1b.edu.ph (revised 2009)

Circular array representation for a queue

```
typedef ..... item;
typedef struct {
    item   ca[MAXSIZE];
    int    front, rear;  // 0..MAXSIZE-1
} queue;
void add_to_queue( queue *q, item x )  {
    if ( front == rear )
        printf("error: queue is full");
    else {
        q->ca [ q->rear ] = x;   // add new item at rear position
        q->rear = (q->rear + 1) % MAXSIZE;
    }
}
```

www.ontologos.org/OML/



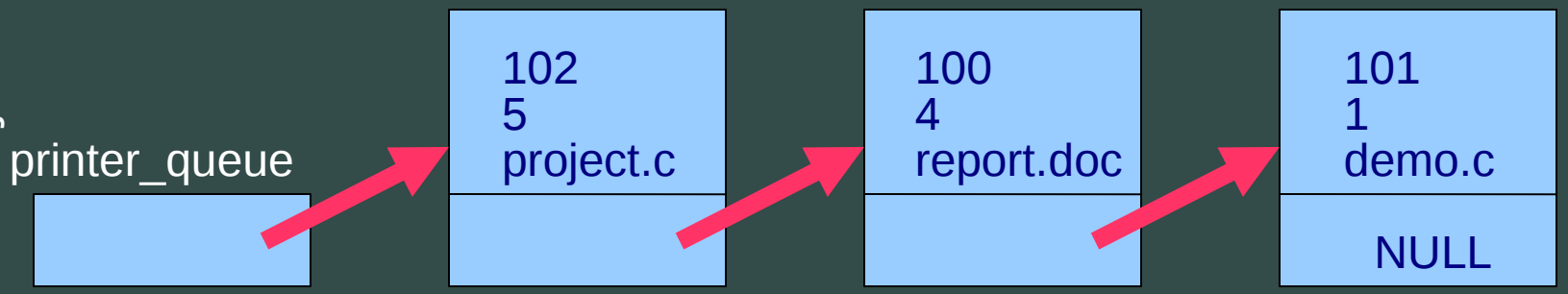
Dynamic linked lists

- Many queues are really dynamic lists that allow additions/deletions at arbitrary positions
- Examples:
  - a document in a print queue can be prematurely canceled without being printed
  - a high-priority job can be put near the front of a job queue
  - a text editor allows insertion and deletion of lines anywhere in the text

jmsamaniego@up1b.edu.ph (revised 2009)

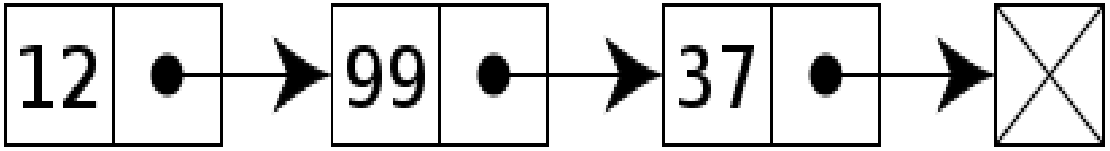
A linked list

```
struct  node  {   // a recursive data structure
  int     job_ID;
  int     priority;
  char  filename[80];
  struct  node  * next;  // “next” points to an identical structure
};
typedef   struct  node    task;
task    * printer_queue;
```

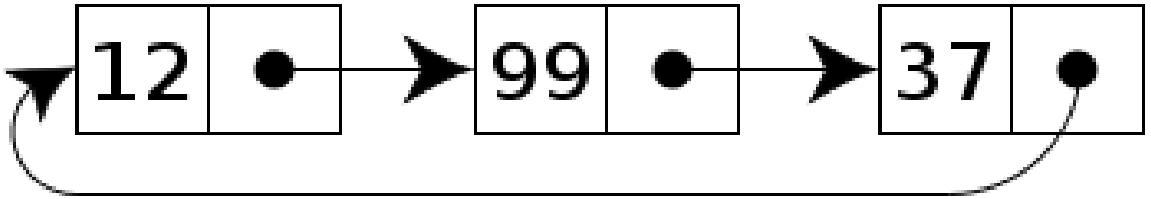


jmsamaniego@up1b.edu.ph (revised 2009)

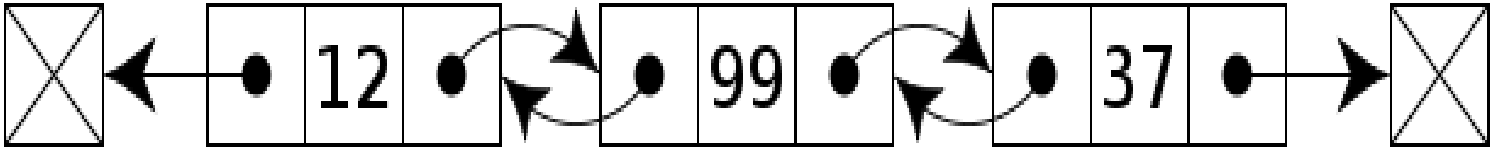
Common types of linked lists



Singly-linked: each node points to the NEXT node



Circular singly-linked: last node points back to the first node

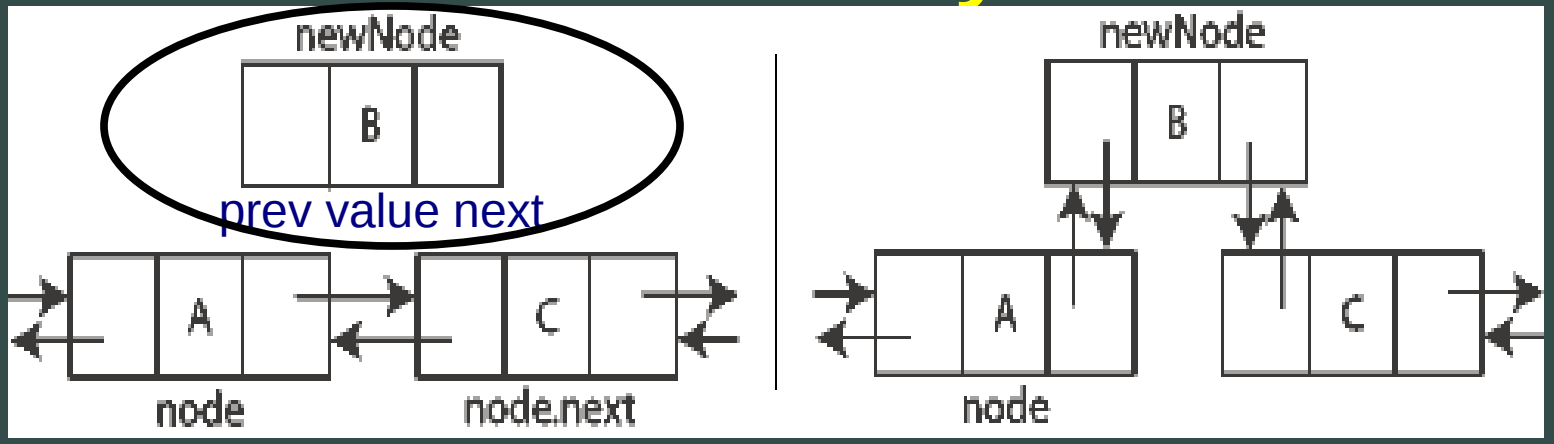


Doubly-linked: each node points to
the NEXT and PREVIOUS nodes

Images from Wikipedia

jmsamaniego@up1b.edu.ph (revised 2009)

Insertion in a doubly-linked list



before

after

To insert a new node between node and node -> next:

0. newnode = malloc (sizeof(node));
1. newnode -> value = 'B';
2. newnode -> prev **=** node;
3. newnode -> next **=** node -> next;
4. node -> next -> prev **=** newnode;
5. node -> next **=** newnode;

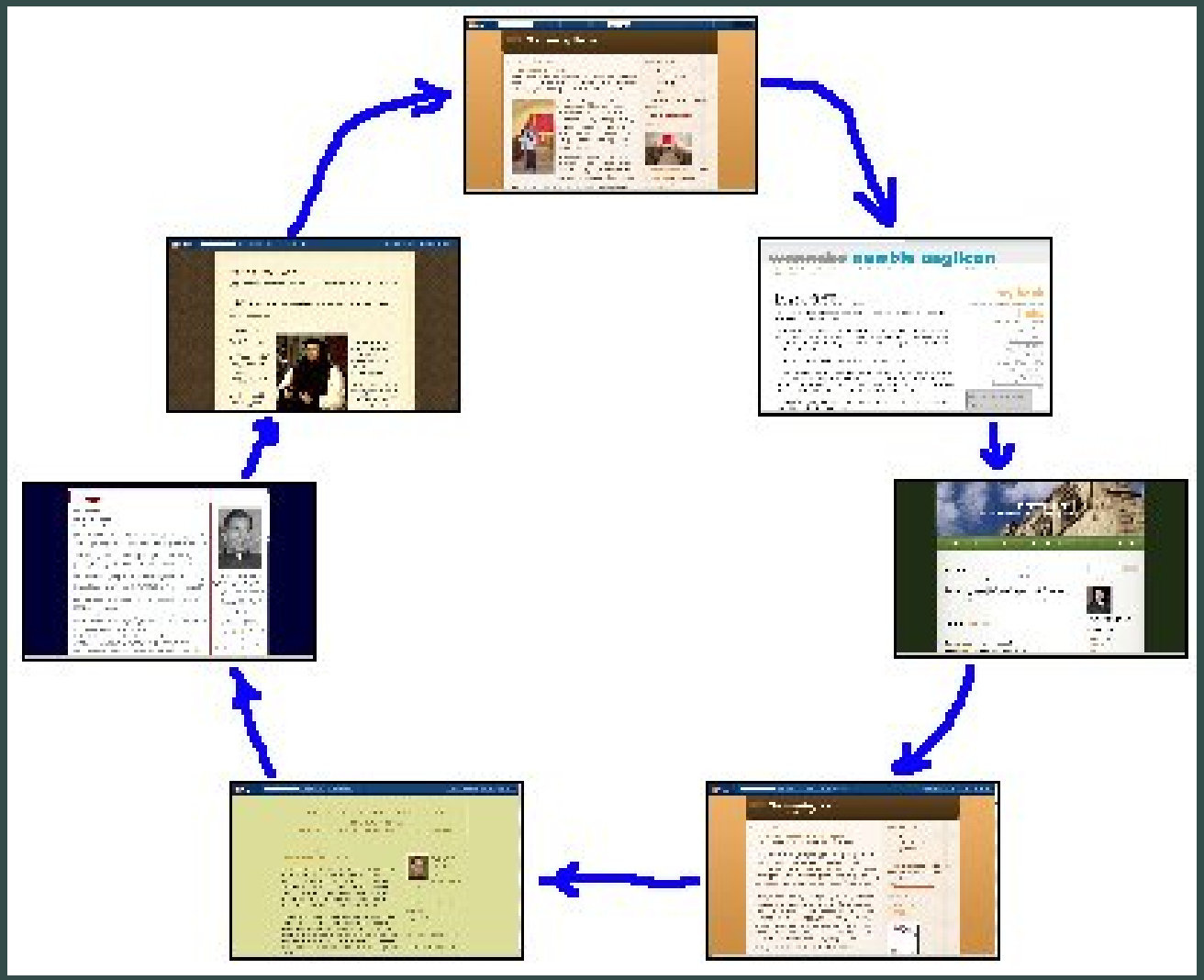
jmsamaniego@up1b.edu.ph (revised 2009)

jmsamaniego@up1b.edu.ph (revised 2009)

A circular doubly-linked list

The Gossips, by Norman Rockwell

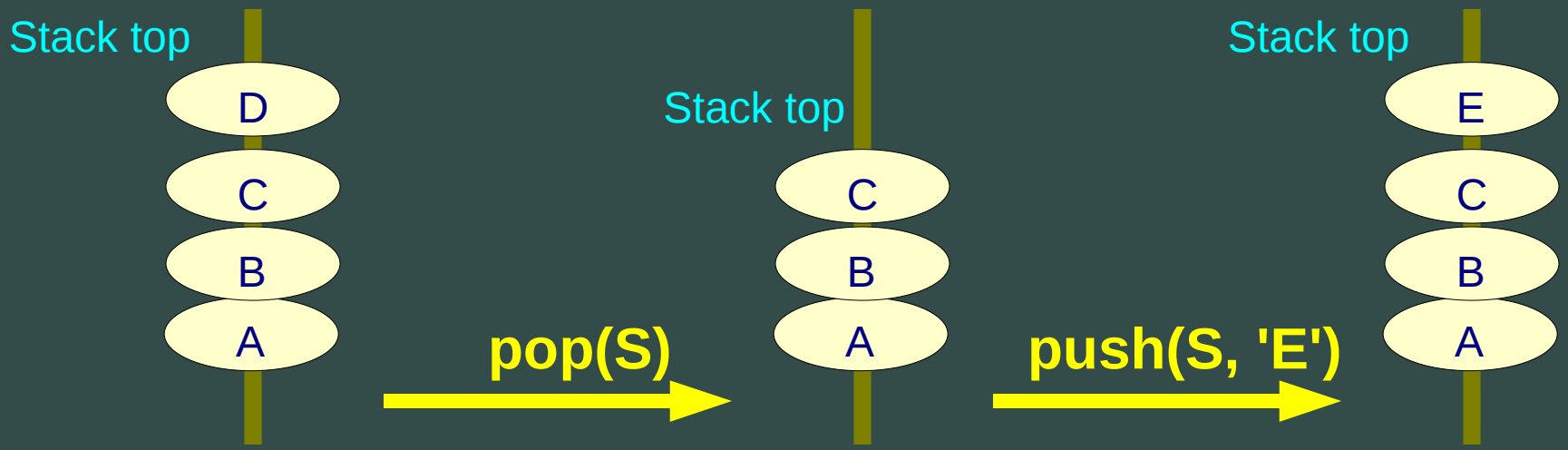
Web rings as circular linked lists



jmsamaniego@up1b.edu.ph (revised 2009)

Stacks

- Stacks are similar to queues but are even more restrictive: additions and deletions are done only at one end called the stack top
- Additions are also called PUSHes, deletions are also called POPs
- This implies a **last-in-first-out** policy

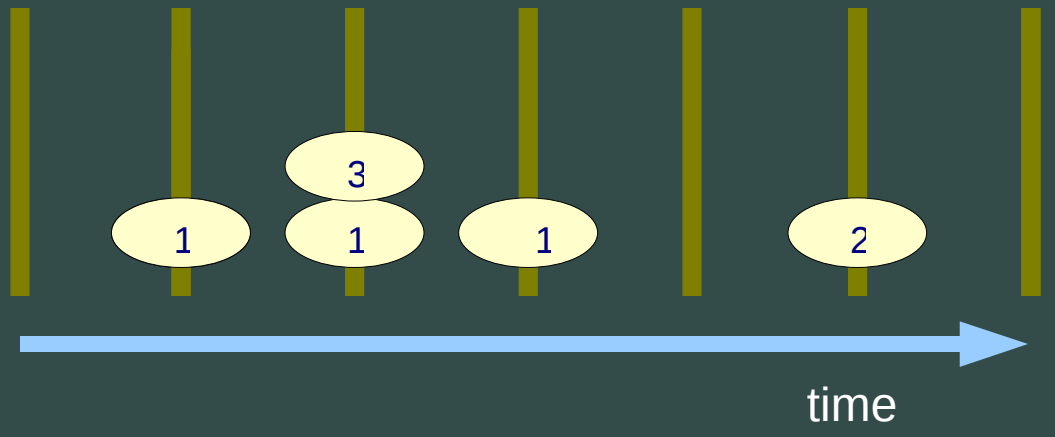


jmsamaniego@up1b.edu.ph (revised 2009)

Stacks and function calls

- Stacks are used when a function calls another function, which may in turn, call yet another function, etc ...
- the return addresses need to be stored in reverse order, and a stack is an ideal data structure for this

```
main()  {
    function_foo();  // 1 (return address)
    function_bar();  // 2
}
function_foo()  {
    function_bar();  // 3
    return;
}
function_bar()  {
}
```



jmsamaniego@up1b.edu.ph (revised 2009)

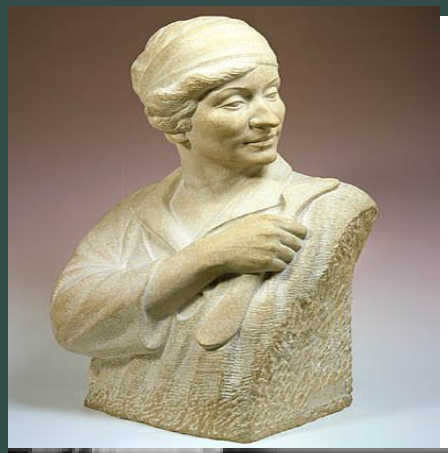
Recursive functions: functions that call themselves

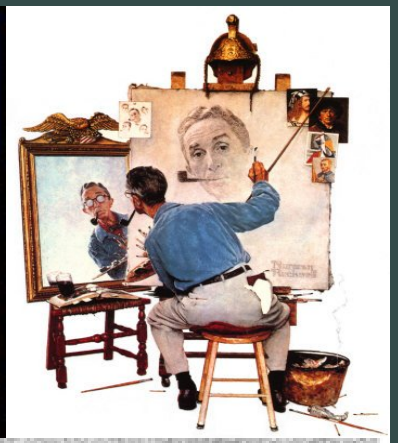
```
int factorial( int n )      // a recursive function definition
{
    if (n==0) return 1;                 // 0! = 1, by definition
    else return n*factorial(n-1);   // n! = n*(n-1)!, for n>0
}
```

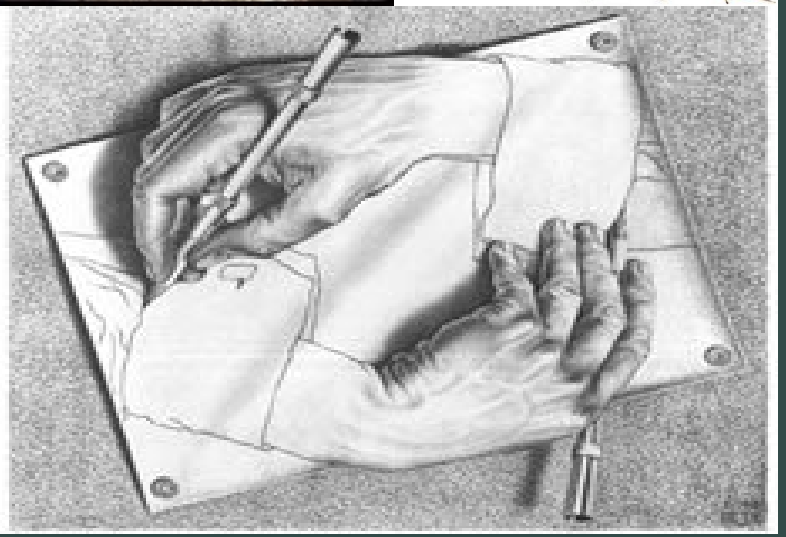
factorial(4) = 4*factorial(4-1)
    where factorial(3) = 3*factorial(3-1)
        where factorial(2) = 2*factorial(2-1)
            where factorial(1) = 1*factorial(1-1)
                where factorial(0) = 1

jmsamaniego@up1b.edu.ph (revised 2009)

Recursion in art – figures that “make themselves”





Images from: ?, Lang-udan, a Romblon carver, Norman Rockwell, ?, MC Escher

jmsamaniego@up1b.edu.ph (revised 2009)

More recursion in art

jmsamaniego@up1b.edu.ph (revised 2009)

MC Escher's strange loops

jmsamaniego@up1b.edu.ph (revised 2009)