R E C U R S I V I D AD

Prof. Nibaldo Rodriguez A.

RECURRENCIA

- Es una función que directa o indirectamente, se hace una llamada a sí misma,
- Pero con instancias (argumentos) más pequeñas; en algún sentido adecuado.

RECURRENCIA

- Un **problema P** se puede resolver conociendo la solución de otro **problema Q** que es del mismo tipo que el **problema P**, pero más pequeño
- Ahora, suponga que puede **resolver Q** a través del **problema R**, el cual es del mismo tipo que **Q y P**, pero de un tamaño menor que ambos.
- Si el **problema R** es tan simple que su solución es directa (no recursiva). Entonces
  - Dado que conocemos la solución de R, procedemos a resolver el **problema Q** y, una vez resuelto **Q**,
  - Se obtiene finalmente la solución del **problema P.**

RECURRENCIA

- Reglas Fundamentales:
  - **Caso Base**:
    - Se debe tener siempre al menos un caso base que pueda resolverse sin recurrencia
    - Puede haber más de un caso base
    - Funciona con el fin de la recursividad
  - **Progreso:**
    - Cualquier llamada recursiva debe progresar hacia un caso base
  - Asumir que toda llamada recursiva interna funciona correctamente.

RECURRENCIA: Observaciones

- La mayor parte de los buenos usos de la recursividad No Agotarán la Memoria del computador y
  - **Sólo son Ligeramente más lentos que la Implementación no recursiva.**
- Pero la recursión siempre nos conduce a Predecir un Código más Compacto.

RECURRENCIA: Observaciones

- Los programas recursivos siempre se pueden implementar de forma **Iterativa con una PILA explicita.**
- El programa será un **poco más rápido**.
- Pero el Código será más **largo y complejo**.
- No Usar Recursión en sustitución de un **BUCLE SIMPLE**.

EJEMPLO: Factorial de un Número

- n!= n*(n-1)!
- 4!=4*3!
- 3!=3*2!
- 2!=2*1!
- 0!=1

```
Int Factotrial(int n) {
        if(n==0) return(1)
        else
                return(n*Factorial)
}
```

EJEMPLO:

- Implementar una Función Recursiva que permita Imprimir por pantalla una Lista enlazada simple en orden Inverso.

```
Void ListRecursiva(Nodo *L) {

    If (L) {
       ListRecursiva(L->sig);
      printf("\n %d",L->d);
    }
}
```

Recorrido InOrden

```
Void Inorden(Arbol *a){

   if (a){
       Inorden(a->izq);
       printf("%d", a->data);
       Inorden(a->der);
    }
}
```

Recorrido PreOrden

```
Void Preorden(Arbol *a){

   if (a){
       printf("%d",a->data);
       Preorden(a->izq);
       Preorden(a->der);
    }
}
```

Recorrido PostOrden

```
Void Postorden(Arbol *a){

   if (a){
      Postorden(a->izq);
      Postorden(a->der)
      printf("%d", a->data);
   }
}
```

EJERCICIOS

- Implementar una Función Recursiva que permita hallar el mínimo de un ABB
- Implementar una Función Recursiva que permita hallar un valor y retornar el nodo del ABB
- Implementar una Función Recursiva para Insertar y Eliminar Datos en un ABB
  Avl * Insert(Avl *T, int d)

•Implementar una Función Recursiva que permita hallar el mínimo de un ABB

```
Arbol FindMin( Arbol T )       {
if( T == NULL )
            return NULL;
elseif( T->Left == NULL )
            return T;
 else
            return FindMin( T->Left );
 }
```

•Implementar una Función Recursiva que permita hallar un valor y retornar el nodo del ABB

```
Arbol Hallar( Arbol *T, int d )  {
if( T == NULL )  return NULL;
if( d < T->d )
          return Find( T->Izq ,d);
elseif( d > T->d )
          return Find(T->Der,d );
else
          return T;
}
```

•Implementar una Función Recursiva para Insertar y Eliminar Datos en un ABB

```
Avl * Insert(Avl *T, int d ) {
if( T == NULL )  T=CreaNodo(d)
else
   if( d < T->d ) {
        T->Left = Insert(T->Izq,d);
   else if( d > T->Element ) {
        T->Right = Insert(T->Der,d);

return T;
}
```

IMPLEMENTACIÓN EN C

- Continuará…