Listas Doblemente Enlazadas

Agregar

```
Procedure agregar (var l:listaDoble; n:integer);
Var
 aux:puntero;
Begin
 new (aux);   aux^.info := n;  aux^.ant := nil; aux^.sig := nil;

 if (l.pri = nil) then begin
                       l.pri:= aux;   l.ult:= aux;
                      end
 else begin
        aux^.sig := l.pri;  l.pri^.ant := := aux; l.pri:= aux;
       end;
End;
```

Insertar Ordenado

```
procedure InsertaOrdenado ( l: listaDoble; x: integer);
var
   act, nuevo : puntero;
begin
  new (nuevo);       nuevo^.info := x;
  nuevo^.sig := nil;
  nuevo^.ant :=nil;

  if (l.pri = nil) then
   begin
    l.pri:= nuevo;  l.ult:= nuevo;
   end
  else begin
  act:= l.pri;

  while (act <> nil) and (x > act^.info) do
      act:= act^.sig;
if (act = l.pri) then begin
                       nuevo^.sig:=l.pri;
                       l.pri^.ant:=nuevo;
                       l.pri:= nuevo;
                      end
 else if (act <> nil) then begin
                         nuevo^.ant:=  act^.ant;
                         act^.ant^.sig:=  nuevo;
                         nuevo^.sig:=  act;
                         act^.ant:= nuevo;
                        end
          else begin
               nuevo^.ant:=l.ult;
               l.ult^.sig:=nuevo;
               l.ult:= nuevo;          end;  end;
```

Recursión

Contar letras

```
Program ContarLetras;
 Var K : integer; { contar las letras de una cadena terminada en punto}
Procedure ContarLetras (var cont: integer);
Var ch : char;
  Begin
   read(ch);  {se lee un caracter}
   if ch <> '.' then Begin
      ContarLetras (cont);  {se invoca recursivamente}
      cont := cont + 1;   {y a la vuelta se incrementa Cont}
     End
    else
     cont :=0  {si se leyo' un punto, Cont es 0}
  End;
```

Secuencia de Fibonacci

```
function fib(n : integer) : integer
 begin
   if (n = 0) or (n = 1) then
        fib := 1;
   else
        fib := fib(n-1) + fib(n-2);
 end.
```

Número mayor de un arreglo

```
Procedure Mayor ( v: vector; ini, fin: integer; var max:integer) ;
  begin
     if (ini <= fin) then
     begin
          if (max< v[ini])  then max := v[ini];
        Mayor ( v,ini+1,fin,max) ;
     end;
  end;
```

Búsqueda lineal en un arreglo

```
procedure Busco_Recursivo( a:Elemento; v:Vector; n:Indice;
                        var i:Indice; var ok:Boolean);
   begin
        if (v[i]=a) then ok:= TRUE
                        else begin
                                i:=i+1;
                                if (i>n) then ok:= FALSE
```

```
                else busco_recursivo(a, v, n, i, ok)
                        end
        end;
```

Sumatoria

```
function sumatoria (n:integer):integer;
begin
  if (n = 1) then   {caso base}
    sumatoria := 1
  else
    sumatoria := n + sumatoria (n-1);
  end;
```

Verificar Capicúa

```
function verificarPalindromo (pal: letras; ini, fin: integer):boolean;
Begin
  if (ini = fin) or (fin < ini) then
verificarPalindromo := true
  else
 if (pal[ini] = pal[fin]) then begin
      ini := ini + 1;
      fin := fin - 1;
      verificarPalindromo := verificarPalindromo (pal, ini, fin);
    end
 else
     verificarPalindromo := false;
end;
```

Árboles

Inicializar y Agregar ABO (Árbol Binario Ordenado)

```
Program uno;
Type
  arbol = ^nodo;
  nodo = record
    elem:integer;
    izq:arbol
    der:arbol
  end;

Procedure Agregar ( var A : arbol;  n : integer);         {Recursivo}
Begin
   if A = nil Then begin    { llegué al final de la rama }
            New( A );
                      A^.dato := n;
                      A^.izq  := nil;
                      A^.der  := nil;
                   end
  else
    if n < A^.dato Then Agregar(A^.izq, n)
  else  Agregar(A^.der, dato);
End;

Procedure Inicializar (var a:arbol);
Begin
  a:= nil;
End;

Var
a: arbol; n:integer;
Begin
  inicializar(a);
  read (n);
While n <> 0 do begin
       agregar(a,n);
       Read (n);
End;
  End.
```

Imprimir de menor a mayor

```
Procedure enOrden ( a : arbol );
begin
    if ( a^.izq <> nil ) then enOrden (a^.izq);
    write (a^.dato);
    if ( a^.der <> nil ) then enOrden (a^.der);
 end;
```

Post Orden

```
Procedure postOrden ( a : arbol );
begin
  if ( a<> nil ) then begin
                       postOrden (a^.izq)
                       postOrden (a^.der)
                       write (a^.dato)
                      end;
end;
```

Pre Orden (Orden jerárquico)

```
Procedure preOrden ( a : arbol );
begin
  if ( a<> nil ) then begin
                       write (a^.dato)
                       preOrden (a^.izq)
                       preOrden (a^.der)
                      end;
end;
```

Mínimo

```
Program uno;
Type
 arbol = ^nodo;
 nodo = record
   elem:integer;
   hi:arbol
   hd:arbol
 end;
{Implementar procedimientos}
Var
a,pundato: arbol; n:integer;
Begin
 inicializar(a); cargar(a);
 pundato:= mínimo(a);
 if (pundato <> nil) then
    write(pundato^.elem);
End.
```

Buscar Mínimo (Recursivo)

```
function minimo ( a:arbol): arbol;
 begin
   if a=nil then minimo:=nil
   else if a^.izq = nil then  minimo:= a
```

```
                    else minimo:=minimo(a^.izq)
 end;
```

Buscar Máximo (Recursiva)

```
Function Buscar_Max ( a:arbol): arbol;     {Iterativa}
 begin
        if a <> nil then
                        while (a^.der<> nil) do
                                a:= a^.der;
        Buscar_max := a;
 end;
```

Suma de todos los elementos del árbol

```
Procedure sumar (a: arbol; var s: integer);
Begin
 If (a <> nil) then begin
                s:= s + a^.dato;
                sumar (a^.hi, s);
                sumar (a^.hd, s);
               end;
End.
```

Buscar elemento (Recursiva)

```
Function Buscar (a:arbol; x:elemento): arbol;
 begin
  if (a=nil) then Buscar:=nil
          else if (x= a^.dato) then Buscar:=a
                          else
                             if (x < a^.dato) then
                                          Buscar:=Buscar(a^.izq ,x)
                                    else
                                          Buscar:=Buscar(a^.der ,x)
end;
```

Buscar elemento (Iterativa)

```
Function Buscar ( A : arbol; Dato:itemType): Boolean;
{ Retorna True si Dato es un nodo del árbol, False en caso contrario}

 Var auxi : arbol;
   Begin
        auxi := A;
        while (auxi <> nil) and (auxi^.dato <> Dato) do
      if Dato < auxi^.dato Then auxi := auxi^.izq
                        Else auxi := auxi^.der;
        Buscar := (auxi <> nil );
```

```
End;
```

Borrar Nodo

```
Procedure Borrar ( x:elemento;  var a: arbol;  var ok:boolean);
 Var
   aux : arbol;
 begin
  if a=nil then ok:=false
  else  begin
    if (x<a^.dato) then Borrar(x,a^.izq,ok) {Busco en el subarbol   izquierdo}
    else if (x>a^.dato) then Borrar (x,a^.der,ok) {Busco en el sub.derecho}
                          else begin  {solo hijo a derecha}
                             if a^.izq =nil then begin
                           aux := a;
                                        a := a^.der;
                           dispose (aux);
                                     end
                                       else{solo hijo a izquierda}
                                      if a^.der =nil then begin
                                aux := a;
                                a := a^.izq;
                                dispose (aux);
                               end
                             {2 hijos. Reemplazo con el más pequeño de la derecha}
                                 else begin
                                aux := buscar_Min(a^.der)
                                a^.dato := aux^.dato;
                                          Borrar(a^.dato,a^.der,ok);
                                                          End
                          End
 End
End
```

Informar Pares

```
Procedure informarNumerosPares(a : arbol);
{Proceso que recorre el árbol e informa los números pares}
Begin
        if (a <> nil) then
        begin
                if ( a^.dato mod 2 = 0 ) then
                        writeln(a^.dato);
                informarNumerosPares(a^.hi);
                informarNumerosPares(a^.hd);
        end;
end;
```