ENUMERATIVO: {*identificador* = (*valor1,valor2,valor3,etc*);}

SUBRANGO: {*identificador* = *valorInicial .. valorFinal*;}

CONJUNTO: {*identificador* = SET OF *tipodedato*;}

DIV: retorna la parte entera.
{*resto*:=*num* DIV 10} → retorna la descomposición del num, todos menos el ultimo digito.

MOD: retorna el último digito.
{*digito*:=*num* MOD 10} → retorna el ultimo digito de num.
{((*num* MOD 2) = 0)} → para saber si num es par.
{((*num* MOD 2) <> 0)} → para saber si num es impar.

REGISTROS:

```
{TYPE
   Nombre-registro = RECORD
        nombreCampo = tipodato;
        nombreCampo = tipodato;
        …
        END;
VAR
  nombreVariable:nombreRegistro;}
```

ARREGLOS:

Declaración:

```
{TYPE
    nombreArreglo = ARRAY [rango] OF tipodato;
  VAR
    nombreVariable:nombreArreglo;}
```

Asignación:

```
{TYPE
    numeros = ARRAY [1 .. 7] OF integer;
VAR
    VN:numeros;
BEGIN
    VN[3]:=7;
    VN[6]:= 10;
END.}
```

```
{BEGIN
   for i:=1 to 7 do begin
        read(num);
        VN[i]:=num;
END.}
```

Inicializar:

```
{PROCEDURE inicializar (var vec:num);
    BEGIN
       For i:=1 to dimF do
            Vec[i]:=0;
    END;}
```

AGREGAR al final:

```
{PROCEDURE agregarAlFinal (var vec:números; var dimL:integer; var OK:boolean;
                                                                num:integer);
  BEGIN
    OK:=false;
    If ((dimL + 1) <= dimF) then begin
        Vec[dimL + 1]:=num;
        dimL:=dimL + 1;
        OK:=true;
        End;}
```

INSERTAR en una POS dada: {PROCEDURE insertar (var vec:números; var dimL:integer; var OK:boolean;
num,pos:integer);

```
BEGIN
  OK:=false;
  If ((dimL + 1) <= dimF) and (pos>1) and (pos<=dimL)) then begin
        For i:=dimL downto pos do
          Vec[i + 1]:= vec[i];
        Vec[pos]:=num;
        dimL:=dimL + 1;
        OK:=true;
        End;}
```

BORRAR elem de una POS dada: {PROCEDURE borrar (var vec:números; var dimL:integer; var OK:boolean;
pos:integer);

```
BEGIN
  OK:=false;
  If (pos>1) and (pos<=dimL) then begin
        For i:=pos to (dimL – 1) do
          Vec[i]:= vec[i + 1];
        dimL:=dimL - 1;
        OK:=true;
        End;}
```

BUSCAR arreglo sin orden: {FUNCTION buscar (var vec:números; dimL,num:integer):boolean;

```
VAR
   OK:boolean; pos:integer;
BEGIN
  OK:=false;
  Pos:=1;
  while (pos <= dimL) and (OK = false) do begin
        if (vec[pos] = num) then
          OK:=true;
          else
            pos:= pos + 1;
        end;
  buscar:= OK;
END;}
```

BUSCAR arreglo ordenado: {FUNCTION buscar (var vec:números; dimL,num:integer):boolean;

```
VAR
   pos:integer;
BEGIN
  Pos:=1;
  while (pos <= dimL) and (num < vec[pos]) do
        pos:=pos + 1;
   if (pos<=dimL) and (vec[pos] = num) then
          buscar:=true;
          else
            buscar:=false;
 END;}
```

BUSCAR dicotómica: {PROCEDURE buscar (var vec:números; dimL,bus:integer; var OK:boolean);

```
VAR
   Pri,ult,medio:integer;
BEGIN
  OK:=false;
  Pri:=1;
  Utl:=dimL;
  Medio:=(pri + ult) DIV 2;
  while (pri <= ult) and (bus <> vec[medio]) do begin
        if (bus < vec[medio]) then
           ult:= medio - 1;
           else
              pri:= medio + 1;
        medio:= (pri + ult) DIV 2;
        end;
  if (pri <= ult) and (bus = vec[medio]) then
        OK:=true;
END;}
```

BUSCAR arreglo sin orden: {FUNCTION buscar (var vec:números; dimL,num:integer):boolean;

```
VAR
   OK:boolean; pos:integer;
BEGIN
  OK:=false;
  Pos:=1;
  while (pos <= dimL) and (OK = false) do begin
        if (vec[pos] = num) then
           OK:=true;
           else
              pos:= pos + 1;
        end;
  buscar:= OK;
END;}
```

ORDENAR: {PROCEDURE ordenar (var vec:vector; dimL:integer);

```
VAR
  i, j, p:integer;
  ítem:tipoElem;
BEGIN
  For i:= 1 to dimL – 1 do begin
      P:= i;
      For j:= i + 1 to dimL – 1 do begin
            if (vec[j] < vec[p]) then
                  p:=j;
            ítem:=vec[p];
            vec[p]:= vec[i]:
            v[i]:= ítem;
            end;
  END;}
```

LISTA ENLAZADA

Declaración: {TYPE
lista = ^nodo*;*
nodo = RECORD
dato:tipodedato;
sig:lista;
end;
VAR
Lis:lista*;*}

Recorrido: {PROCEDURE recorrido (aux:lista);
BEGIN
While (aux <> nil) do begin
Write(aux^.dato);
Aux:=aux^.sig;
End;
END;}

Crear lista: {BEGIN
…….
lis:=NIL;
…….
END;}

BUSCAR elemento: {FUNCTION buscar (aux:lista; elem:string):boolean;
VAR
OK:boolean;
BEGIN
OK:=false;
while (aux <> nil) and (OK = false) do begin
if (elem = aux^.dato) then
OK:=true;
else
aux:=aux^.sig;
buscar:= OK;
END;}

AGREGAR al principio: {PROCEDURE agregarAdelante (var lis:lista; num:integer);
VAR
Aux:lista;
BEGIN
New(aux);
Aux^.dato:=num;
Aux^.sig:=lis;
Lis:=aux;
END;}

AGREGAR al final: {PROCEDURE agregarAlFinal (var lis:lista; num:integer);

```
VAR
   Act,ult,Aux:lista;
BEGIN
    New(aux);
    Aux^.dato:=num;
    Aux^.sig:=lis;
    If (lis <> nil) then begin
          Act:=lis;
          While (act <> nil) do begin
                Ult:=act;
                Act:=act^.sig;
                End;
          Ult^.sig:=aux
          End
          Else
                Lis:=aux;
END;}
```

INSERTAR elemento: {PROCEDURE insertarNodo (var lis:lista; num:integer);

```
VAR
   Act,ant,Aux:lista;
BEGIN
    New(aux);
    Aux^.dato:=num;
    Act:=lis;
    Ant:=lis;
    While (act <> nil) and (act^.dato < num) do begin
          Ant:=act;
          Act:=act^.sig;
          End;
     if (ant = act) then begin
          lis:=aux;
          else
               ant^.sig:=aux
     aux^.sig:=act;
END;}
```

BORRAR elemento: {PROCEDURE borrarElemento (var lis:lista; num:integer);

```
VAR
   Act,ant:lista;
BEGIN
    Act:=lis;
    Ant:=lis;
    While (act <> nil) and (act^.dato <> num) do begin
          Ant:=act;
          Act:=act^.sig;
          End;
     if (ant <> nil) then begin
          if (act = lis) then
               lis:=act^.sig;
               else
                    ant^.sig:=act^.sig;
           dispose(act);
           end;
```

LISTA CIRCULAR

AGREGAR elem al principio: {PROCEDURE agregarAdelante (var lis:listacircular; num:integer);

```
VAR
   Aux,aux2:lista;
BEGIN
    New(aux);
    Aux^.dato:=num;
    If (lis = nil) then begin
          Aux^.sig:=lis;
          Else
               Aux2:=lis;
               While (aux2^.sig <> lis) do
                    Aux2:=aux2^.sig;
               Aux2^.sig:=aux;
               Aux^.sig:=lis;
               End;
END;}
```

BUSCAR elemento: {FUNCTION buscar (lis:listacircular; elem:integer):boolean;

```
VAR
   Ult,act:listacircular;
   OK:boolean;
BEGIN
  OK:=false;
  If (lis <> nil) then begin
          Ult:=lis;
          Act:=lis;
          while (act^.dato <> elem) and (ult^.sig <> lis) do begin
               ult:=act;
               act:=act^.sig;
               end;
           if (act^.sig = elem) then
              OK:=true;
            end;
    buscar:= OK;
END;}
```

BORRAR elemento: {PROCEDURE borrar (var lis:listacircular; elem:integer; var OK:boolean);

```
VAR
  Act,ult:listacircular;
BEGIN
   OK:false;
   If (lis <> nil) then begin
      Act:=lis;
      ult:=lis;
      While (ult^.sig <> lis) and (act^.dato <> num) do begin
         ult:=act;
         Act:=act^.sig;
         End;
    if (act^.dato = elem) then begin
         OK:=true;
         if (ult = act) then
            while (ult^.sig <> lis) do
                 ult:=ult^.sig;
          ult^.sig:=lis^.sig;
          lis:=lis^.sig;
          else
               ult^.sig:=act^.sig;
        dispose(act);
        end;
END;}
```

ARBOLES

Declaración: {TYPE

```
   arbol = ^nodo;
   nodo = RECORD
            hijoIzq:arbol;
            elem:tipodato;
            hijoDer:árbol;
            end;
VAR
  A:arbol;}
```

Creación: {BEGIN

```
        ….
        A:=nil;
        ….
END.
```

INSERTAR (recursivo): {PROCEDURE insertar (var a:árbol; dato:tipodato);

```
BEGIN
   If (a = nil) then begin
         New(a);
         Aux^.elem:=dato;
         A^.hijoIzq:=nil;
         A^.hijoDer:=nil;
         End
         Else
             If (dato < a^.elem) then
                 Insertar (a^.hijoIzq, dato);
                 Else
                      Insertar (a^.hijoDer, dato);
  END;}
```

RECORRIDO enOrden: {PROCEDURE enOrden (a:árbol);

```
BEGIN
   If (a <> nil) then begin
      enOrden(a^.hijoIzq);
      Write(a^.elem);
      enOrden(a^.hijoDer);
      end;
 END;}
```

RECORRIDO postOrden: {PROCEDURE postOrden (a:árbol);

```
BEGIN
   If (a <> nil) then begin
      postOrden(a^.hijoIzq);
      postOrden(a^.hijoDer);
      Write(a^.elem);
      end;
 END;}
```

RECORRIDO postOrden: {PROCEDURE preOrden (a:árbol);

```
BEGIN
   If (a <> nil) then begin
      Write(a^.elem);
      preOrden(a^.hijoIzq);
      preOrden(a^.hijoDer);
      end;
 END;}
```

BUSCAR elemento: {FUNCTION buscar (a:Arbol; dato:integer):arbol;

```
BEGIN
  If (a = nil) then
        Buscar:=nil;
        Else
          If( dato = a^.elem) then
             Buscar:=a;
             Else
                If (dato < a^.elem) then
                   Buscar:= buscar (a^.hijoIzq, elem);
                   Else
                      Buscar:=buscar (a^.hijoDer, elem);
END;}
```

BUSCAR minimo: {FUNCTION buscarMIN (a:Arbol):arbol;

```
BEGIN
  If (a = nil) then
        BuscarMIN:=nil;
        Else
          If (a^.hijoIzq = nil) then
             BuscarMIN:=a;
             Else
                BuscarMIN:=buscarMIN (a^.hijoIzq);
END;}
```

BUSCAR maximo: {FUNCTION buscarMAX (a:Arbol):arbol;

```
BEGIN
  If (a <> nil) then
        While (a^.hijoDer <> nil) do
            A:=a^.hijoDer;
   buscarMAX:=  a;
END;}
```

BORRAR elemento: {PROCEDURE borrar (x:integer; var a:árbol; var OK:boolean);

```
VAR
   Aux:árbol;
BEGIN
   If (a = nil) then
      OK:=false;
      Else
         If (x < a^.elem) then
            Borrar (x, a^.hijoIzq, OK);
            Else
               If (x > a^.elem) then
                  Borrar (x, a^.hijoDer, OK);
                  Else
                    OK:=true;
                    If (a^.hijoIzq = nil) then begin
                       Aux:=a;
                       A:=a^.hijoDer;
                       Dispose(aux);
                       End
                       Else
                            If (a^.hijoDer = nil) then begin
                               Aux:=a;
                               a:=a^.hijoIzq;
                               dispose(aux);
                               end
                               else
                                  aux:=buscarMIN(a^.hijoder);
                                  a^.elem:=aux^.elem;
                                  borrar(a^.elem, a^.hijoDer, OK);
   END;}
```