Function:

```
function nombre (parámetros FORMALES): tipo;
type (opcional);
var (opcional);
begin
      …
      nombre:= (Asignación única y obligatoria);
end;
```

Procedure:

```
procedure nombre (parámetros FORMALES x valor o referencia (var));
type (opcional);
var (opcional);
begin
      …
end;
```

Record:

```
type
  identificador = record;
    campo1: tipo de dato;
    campo2: tipo de dato;
    campo3: tipo de dato;
    …
    campoN: tipo de dato;
  end;
{Para leer un record lo mejor es crear un módulo leer}.
```

Array:

```
vector = array [índice (n..N)] of tipo_elementos;
```

Asignación a posición

```
v[posición]:= valor;
```

Lectura y escritura de elementos

```
readln o writeln (v[posición]);
{Para hacer una carga completa de vector de record se modulariza
la lectura}
 for i:= 1 to 100 do begin
      LeerVector (v[i]);
      MostrarVector (v[i]);
 end;
```

Agregar un elemento al final

```
Const
    dimF = 1000;
Type
   vector =  array [ 1..dimF] of tipo_elemento;

procedure AGREGAR (var v: vector; var dimL: integer;
                   elemento: tipo_elemento; var exito: boolean);
begin
exito:= false;
{verificar espacio suficiente}
      if (dimL < dimF) then begin
            exito:= true;
            dimL:= dimL+1; {actualizar cantidad de elementos}
            v [dimL]:= elemento;
      end;
end;
```

Insertar elemento

```
procedure INSERTARPOS (var v:vector; var dimL: integer;
  elemento: tipo_elemento; pos: integer; var exito: boolean);

var i : integer;
begin
exito:= false;
if (dimL < dimF) and ((pos>=1) and (pos<= dimL))then begin
      exito:= true;
      for  i:= dimL downto pos do
      v [ i + 1 ] := v [ i ] ;
      v [pos] := elemento;
      dimL := dimL + 1;
      end;
end;
```

Borrar elemento

```
procedure BorrarPos (var v: vector; var dimL: integer;
pos:posicion;
           var exito: boolean );
var i: integer;
begin
     exito := false;
     if (pos >=1 and pos <= dimL) then begin
          exito := true;
          for i:= pos + 1  to   dimL  do
               v [ i - 1 ]  :=  v [ i ] ;
          dimL := dimL - 1 ;
     end;
end;
```

Borrar elemento determinado

```
procedure BorrarPosModif (var v:vector; var dimL:integer;
pos:Indice);
var i: integer;
begin
     for i:= pos + 1  to   dimL  do
          v [ i - 1 ]  :=  v [ i] ;
     dimL := dimL - 1 ;
end;
```

Búsqueda lineal o secuencial

```
function BuscarPosElem (x:integer; v:vector; dimL: Indice): Indice;
var pos:Indice; exito: boolean;
begin
     pos:=1;
     exito:= false;
     while (pos <= dimL) and (not exito) do
          if (x = v[pos]) then
               exito:= true
          else pos:= pos+1;
          if (exito = false) then
               pos:=0;
     BuscarPosElem := pos;
end;
```

Búsqueda secuencial optimizado

```
function BuscarPosElem (x:integer; v:vector; dimL: Indice): Indice;
var pos:Indice;
begin
      pos:=1;
      while (pos <= dimL) and (x > v[pos]) do
            pos:= pos+1;
      if (pos > dimL) or (x < v[pos]) then
            pos:=0;
      BuscarPosElem := pos;
end;
```

Búsqueda binaria o dicotómica

```
procedure  BusquedaBin ( var v: Vector; var j: Indice;
                              dimL: Indice, x : TipoElem) ;
var pri, ult, medio :  Indice ;
begin
      j :=0 ;
      pri:= 1 ;
      ult:= dimL;
      medio := (pri + ult ) div 2 ;
      while  ( pri < = ult ) and ( x <> v [medio]) do begin
            if ( x < v [ medio ] ) then
                  ult:= medio -1 ;
            else
                  pri:= medio+1 ;
            medio := ( pri + ult ) div 2 ;
      end ;
      if pri < = ult then
            j := medio
      else
            j := 0 ;
end ;
```

Insertar en vector ordenado

```
procedure Insertar (var v:vector; var dimL:Indice; pos: Indice;
                    elem:integer);
var j: indice;
begin
     for j:= dimL downto pos do
          v [ j +1 ] := v [ j ] ;
     v [ pos ] := elem;
     dimL := dimL + 1;
end;
{Se le pasa la posición de la función de búsqueda}
```

Vector contador

```
{se inicializa en 0}
for i:=1 to dimF do
     v2[i]:= 0;
{se le asignan los valores de otro vector}
{pasa a la siguiente pos}
for i:=1 to dimL do begin
v2 [v1[i].campo]:= v2[v1[i].campo] +1;
end;
```

Ordenar vector

```
for i := 1 to dimF do begin
        for j := i+1 to dimF do begin
            if (vector[j] < vector[i]) then begin
                aux := vector[i];
                vector[i] := vector[j];
                vector[j] := aux;
            end;
        end;
    end;
```

Pointer

```
type: tipoPuntero = ^TipoVarApuntada;
{luego se declara una variable}
ej: type:puntAString = ^string;
var puntero : puntAString;
```

Asignaciones

```
Type
      TipoString = string[20];
      Datos = record
            Nombre: TipoString;
            Apellido:TipoString;
            Edad: integer;
            Altura: real;
      end;
      PtrDatos = ^datos;
      PtrReal = ^real;
      PtrString = ^TipoString;
      PtrNil = ^char;
var
      nulo  : PtrNil;
      p     : PtrDatos;
      peso  : PtrReal;
      frase : PtrString;
begin
      New(p); New(peso); New(frase); New (nulo)
      read (P^.nombre);
      read (P^.apellido);
      p^.edad := 30;
      p^.altura := 1.74;
      peso^ := 3;
      frase^ := "La casa de María";
      nulo^ := nil;
```

Dispose

```
dispose (variable);
{la variable apuntada será eliminada, esto debe hacerse antes de
apuntar a otra variable}
```

Lists

```
type
      lista = ^nodo;
      nodo = record
            datos: tipoDato; {contenido}
            sig: lista; {dirección del siguiente nodo}
            end;
var L: lista; {puntero inicial}
```

```
Procedure recorrido ( pri : lista);

Begin
 while (pri <> NIL) do begin
   write (pri^.datos.nom,
          pri^.datos.edad) ;
   pri:= pri^.sig
 end;
end;
```

```
function buscar (pri: lista; x:cadena50):boolean;
Var
   encontre : boolean;

begin
 encontre := false;
 while (pri <> NIL) and (not encontre) do
   if x = pri^.datos.nom  then
                            encontre:= true
                          else
                            pri:= pri^.sig;
   buscar := encontre
End;
```

Agregar adelante

```
procedure agregarAdelante(var l:lista; dato:integer);
var
      nue: lista;
begin
      new(nue);
      nue^.dato := dato;
      nue^.sig := l;
      l := nue;
end;
```

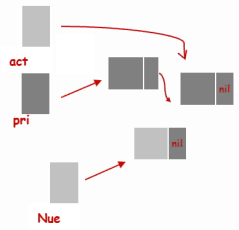
Agregar atrás

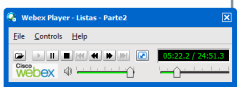
```
// OPCIÓN 1 - recorre la lista para encontrar el último elemento

procedure agregarAtras(var l:lista; dato: integer);
var
      nue, act, ant: lista;
begin
      new(nue);
      nue^.dato := dato;
      ant := l;
      act := l;
      while (act <> nil) do begin
            ant := act;
            act := act^.sig;
      end;
      if (ant = act) then
            l := nue
      else
            ant^.sig := nue;
      nue^.sig := act;
end;
```

```
procedure AgregarAlFinal (var pri: lista; per: persona);
var  act, nue : lista;

begin
 new (nue);
 nue^.datos:= per;
 nue^.sig := NIL;
 if pri <> Nil then begin
                act := pri ;
                while  (act^.sig <> NIL ) do act := act^.sig ;
                act^.sig := nue ;
              end
              else
                pri:= nue;
end;
```





```
// OPCIÓN 2 - con puntero al último elemento
procedure agregarAtras(var l: lista; var ult: lista; dato:
integer);
var
    nue:lista;
begin
     new(nue);
     nue^.dato := dato;
     nue^.sig := nil;
     if (l <> nil) then
          ult^.sig := nue;
     else
          l := nue;
     ult := nue;
end;
```

```
procedure AgregarAlFinal2 (var pri, ult: lista; per: persona);
var  nue : lista;

begin
 new (nue);
 nue^.datos:= per;
 nue^.sig := NIL;
 if pri <> Nil then
                ult^.sig := nue;
              else
                pri := nue;
 ult := nue;
end;
```

Si la lista tiene elementos

Eliminar elemento

```
Procedure BorrarElemento (var pri:lista; nom:cadena50; var exito:
boolean);
var ant, act: lista;
begin
  exito := false;
  act := pri;
  {Recorro mientras no se termine la lista y no encuentre el elemento, si el
elemento seguro existe, no se pregunta por nil}
  while  (act <> NIL)  and (act^.datos.nom <> nom) do begin
       ant := act;
       act := act^.sig
  end;
  if (act <> NIL) then begin
        exito := true;
    if (act = pri) then
      pri := act^.sig;
    else
      ant^.sig:= act^.sig;
    dispose (act);
  end;
```

```
end;
```

Insertar ordenado

```
procedure insertarOrdenado(var L:lista; j:jugador);
var
  nue: lista;
  act, ant: lista; {punteros auxiliares para recorrido}
begin
  new (nue);
  nue^.dato := j;
  act := L; {ubico act y ant al inicio de la lista}
  ant := L;
  while( act <> nil)and(j.altura > act^.dato.altura)do
  begin
    ant := act;
    act:= act^.sig;
  end;
  if (act = ant) then {al inicio o lista vacía}
    L:= nue;
  else {al medio o al final}
    ant^.sig:= nue;
  nue^.sig:= act;
end;
```

```
Procedure InsertarElemento ( var pri: lista; per: persona);
var ant, nue, act: lista;
begin
  new (nue);
  nue^.datos := per;
  act := pri;
  ant := pri;
{Recorro mientras no se termine la lista y no encuentro la posición correcta}
  while (act<>NIL) and (act^.datos.nombre < per.nombre) do begin
      ant := act;
      act := act^.sig ;
  end;
  if (ant = act)  then pri := nue   {el dato va al principio}
                  else  ant^.sig  := nue; {va entre otros dos o al final}
  nue^.sig := act ;
end;
```

Saber si está ordenada

```
function EstaOrdenada(l:lista):boolean; //DE MENOR A MAYOR
var
    ordenada: boolean;
    actual,anterior: integer;
begin
    ordenada:=true;
    actual:=-9999;
    while (l<>nil) and (ordenada) do begin
        anterior:=actual;
        actual:=l^.num;
        if(anterior > actual) then
            ordenada:=false;
        l:=l^.sig;
    end;
    EstaOrdenada:=ordenada;
end;
```