APRENDIENDO A USAR LISTAS EN:

Pascal

LISTA

Coleccion de nodos, donde cada nodo tiene un elemento y en que direccion de memoria se encuentra el siguiente nodo.

Cada nodo de la lista se representa con un puntero, que apunta a un dato (elemento de la lista) y a una direccion (donde se ubica el siguiente elemento de la lista).

Toda lista tiene un elemento inicial.

Declaracion:

```
type
    nombreTipoLista = ^nombreNodo;
    nombreNodo = record
        elem : tipoElemento;
        sig : nombreTipoLista;
    end;
```

Siempre antes de operar con una variable de lista hay que inicializar en nil, en el caso que la primera operacion sea asignarle el valor de otro puntero no es necesario.

Importante: antes de acceder a un nodo de lista se tiene que validar que la direccion sea valida.

Acceso:

```
{ Acceso al elemento del nodo }
variableLista^.elem
{ Acceso al siguiente nodo }
variableLista^.sig
```

Carga

Cuando se desconoce la cantidad de elementos, se carga con un while para que verifique una condicion de carga.

```
type
    tElemento = record
        { ... }
    end;
    tLista = ^tNodo;
    tNodo = record
procedure cargarLista(var lista: tLista);
var
    leido: tElemento;
begin
    leerElemento(leido);
    while(leido.campoCorte <> VALOR_CORTE) do begin
        metodoCarga(lista, { otros argumentos, } leido);
        leerElemento(leido);
    end;
end;
```

Metodos de carga

```
{ Agregar al inicio }
procedure agregarAdelante(var lInicio: tLista; elemento: tElemento);
var
    lNueva: tLista;
begin
    new(lNueva);
    nue^.elem := elemento;
    nue^.sig := inicio;
end;

{ Agregar al final }
procedure agregarAtras(var lInicio, lFinal: tLista; elemento: tElemento);
var
    lNueva: tLista;
begin
    new(lNueva);
    nue^.elem := elemento;
    nue^.sig := nil;
    if (lInicio = nil) then
        lInicio := lNueva;
    else
        lFinal^.sig := lNueva;
    lFinal := lNueva;
end;
```

Insertar ordenado

```
{ Insertar ordenado }
procedure insertarOrdenar(var lista: tLista; elemento: tElemento);
var
    lNueva, lActual, lAnterior: tLista;
begin
    new(lNueva);
    nue^elem := elemento;
    lActual := lista;
    while(lActual <> nil) and (elemento.campoOrden > lActual^elem.campoOrden) do
    begin
        lAnterior := lActual;
        lActual := lActual^.sig
    end;
    if (lActual = lAnterior) then
        lista := lNueva
    else
        lAnterior^.sig := lNueva;
    lNueva^.sig := lActual;
end;
```

Busqueda

```
function buscar(lista: tLista; valorBuscado: tipo): boolean;
begin
    while(lista <> nil) and (lista^.elem.campoValor <> valorBuscado) do
        lista := lista^.sig
    buscar := lista <> nil;
end;
```

Eliminacion

```
procedure eliminar(var lista: tLista; valorAlEliminar: tipo);
var
    lActual, lAnterior; tLista;
begin
    lActual := lista;
    lAnterior := lista;
    while((lActual <> nil) and (lActual^.elem.campoValor <> valorEliminar)) do
    begin
        lAnterior := lActual;
        lActual := lActual^.sig;
    end;
    if (lActual <> nil) then begin
        if (lActual = lista) then
            lista := lActual^.sig;
        else
            lAnterior^.sig := lActual^.sig;
        dispose(lActual);
    end;
end;
```