Ejemplos de tipo de datos abstractos

Lista enlazada

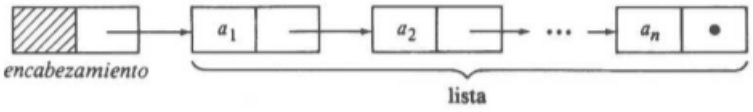

Fig. 2.4. Lista enlazada.

Listas doblemente enlazadas

```
procedure INSERTA ( x: tipo_elemento; p: posicion; var L: LISTA );
    begin
        if p = 0 then begin
            { inserta en la primera posición }
            if mueve ( disponible, L ) then
                ESPACIO [L]. elemento := x
        end
        else { inserta en una posición que no es la primera }
            if mueve (disponible, ESPACIO [p].sig) then
                { celda que ocupará x ya apuntada por ESPACIO [p].sig }
                ESPACIO [ESPACIO [p].sig]. elemento : = x
    end; { INSERTA }

procedure SUPRIME (p: posición; var L: LISTA);
    begin
        if p = 0 then
            mueve (L, disponible)
        else
            mueve (ESPACIO [p].sig, disponible)
    end; { SUPRIME }
```



```
procedure val_inicial;
    { val_inicial enlaza ESPACIO en una lista de celdas disponibles }
    var
        i: integer;
    begin
        for i := long_máx-1 downto 1 do
            ESPACIO [i].sig := i + 1;
        disponible := 1;
        ESPACIO [long_máx] .sig :=0 { marca el final de la lista de celdas dispo-
          nibles}
    end; { val_inicial }
```

Fig. 2.12. Algunos procedimientos para listas enlazadas basadas en cursores.

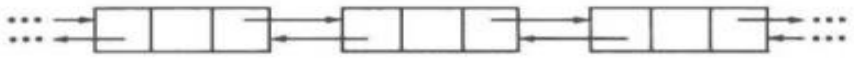

Fig. 2.13. Una lista doblemente enlazada.

Ejemplo con pilas

```
procedure ANULA ( var P: PILA);
    begin
        P.tope := long_máx + 1
    end; { ANULA }

function VACIA ( P: PILA ): boolean;
    begin
        if P.tope > long_máx then
            return (true)
        else
            return (false)
    end; { VACIA }

function TOPE ( var P: PILA ): tipo_elemento;
    begin
        if VACIA (P) then
            error ('la pila está vacía')
        else
            return (P.elementos[P.tope])
    end; { TOPE }

procedure SACA ( var P: PILA );
    begin
        if VACIA(P) then
            error ('la pila está vacía')
        else
            P.tope := P.tope + 1
    end; { SACA }

procedure METE ( x: tipo_elemento; P: PILA);
    begin
        if P.tope := 1 then
            error ('la pila está llena')
        else begin
            P.tope := P. tope – 1;
            P.elementos[P.tope] := x
        end
    end; { METE }
```

Realización circular de colas

```
function suma_uno ( i : integer ): integer;
    begin
        return ((i mod long_máx) + 1)
    end; { suma_uno }

procedure ANULA ( var C: COLA );
    begin
        C. ant := 1;
        C. post := long_máx
    end; { ANULA }

function VACIA ( var C: COLA ) : boolean;
    begin
        if suma_uno(C. post) = C.ant then
            return (true)
        else
            return (false)
    end; { VACIA }

function FRENTE ( var C: COLA ) : tipo_elemento;
    begin
        if VACIA (C) then
            error ('la cola está vacía')
        else
            return (C.elementos[C.ant])
    end; { FRENTE }

procedure PONE_EN_COLA ( x: tipo_elemento; var C: COLA );
    begin
        if suma_uno(suma_uno(C.post)) = C.ant then
            error ('la cola está llena')
        else begin
            C.post := suma_uno(C.post);
            C.elementos[C.post] := x
        end
    end; { PONE_EN_COLA }

procedure QUITA_DE_COLA ( var C: COLA );
    begin
        if VACIA (C) then
            error ('la cola está vacía')
        else
            C.ant := suma_uno(C.ant)
    end; { QUITA_DE_COLA }
```

-Aho, A., Hopcroft, J.,Ullman, Jeffrey. (1998). Estructura de datos y algoritmos (2da edición). México: S.A Alhambra Mexicana