N°
Fecha: / /

Luis Ulián Peñeda Choque

UNIDAD II TDA Lista

2.1.1 Definición del TDA Lista

En general, una lista es una secuencia de elementos de forma $a_1, \ldots a_n$ donde $n \geq 0$ y cada elemento a_i es de tipo genérico. El tamaño de la lista es vacía. Para todas las listas menos esta última, el primer es a_1 y el último elemento es a_n. Se dice que a_{i+1} es sucesor de a_i ($i < n$) y que a_{i-1} es proceso de a_i ($i > 1$). Por lo tanto, los elementos puede estar ordenado en función de posición (i en el caso de a_i).

A diferencia de los conjuntos puede haber elementos repetidos en la lista y a diferencia de las matrices y registro, el número de elementos de la lista no es fijo, es variable por lo tanto, no es limitado desde el principio.

En una maquina de información cada elemento suele denominarse nodo (celda o caja) que puede contener uno o más punteros.
Las lista admiten una serie de operaciones que podemos agrupar en:

Operaciones de construcción (crear).
Operaciones de posición (f.n. primero, siguiente, anterior)
Operaciones de consultas (vacía, longitud, recupera) y finalmente
Operaciones de modificación (modifica, suprime, inserta)
que se detallara e implementara más adelante

Los TDA-lista se puede realizar mediante memoria estática (vectores o arrays) o mediante asignación de memoria dinamica con punteros y mediante el uso de un modelo simulado de memoria dinámica con TDA Memoria

$$L = \langle a_1, a_2, \ldots a_n \rangle$$

Donde a_1 es el primer elemento y a_n es un elemento. $\forall a_i \in L$

2.1.2 Especificación del TDA Lista

Partiendo de lo establecido en la unidad 1. Especificación informal tenemos lo siguiente:
Elementos que conforman la estructura

TDA Lista (Valores Todos los números Enteros, Operaciones crea, fin primero, siguiente: recupera, longitud, insertar_primero, suprime, modifica
Elemento que conforman la estructura

N°________________ Tema: ________________
Fecha: ____/____/____

Luchino Pencoca Choque

DESCRIPCIÓN DE LAS OPERACIONES

Crear (L: Lista)
Utilidad Sirve para iniciar la lista
Entrada: Lista L
Salida Ninguna
Precondición Ninguna
Poscondición Lista L inicializada vacía

Fin (L: lista) devuelve (Dirección)
Utilidad Retorna la dirección donde se encuentra el último elemento de la lista L
Entrada Lista L
Salida Dirección
Precondición Lista no vacía
Poscondición Ninguna

Primero (L: lista) devuelve (Dirección)
Utilidad Retorna la dirección donde se encuentra el primer elemento de la lista L
Entrada Lista L
Salida Dirección
Precondición Lista no vacía
Poscondición Ninguna

Siguiente (L: lista; P: Dirección) devuelve (Dirección)
Utilidad Devuelve la dirección que ocupa el elemento sucesor del elemento que ocupa la dirección P en la lista L
Entrada Lista L
Salida Dirección
Precondición: Lista no vacía y P no es la dirección del último elemento
Poscondición: Ninguna

anterior (L: lista; P: Dirección) devuelve (Dirección)
Utilidad Devuelve la dirección que ocupa el elemento predecesor del elemento que ocupa la dirección P en la lista L
Entrada Lista L
Salida Dirección
Precondición Lista no vacía y P no es la dirección del último elemento
Poscondición Ninguna

Vacía (L: Lista) devuelve (entero)
Utilidad: Devuelve la longitud de la lista L / Cantidad de elementos
Entrada Lista L
Salida Número
Precondición Ninguna
Poscondición Ninguna

Tema: Lista Tercera Clase

N°
Fecha: / /

Recorre (L: Lista; P: Dirección) devuelve (E: Elemento)
utilidad: Devuelve en E el elemento que ocupa la dirección P en la lista L
Entrada: Lista L
Salida: Elemento
Precondición: La lista L es no vacía. La dirección P es la dirección de un elemento de la lista L
Poscondición: Ninguna

Longitud (L: Lista) devuelve (q: Entero)
Utilidad: Devuelve la longitud de la lista L / cantidad de elementos
Entrada: Lista L
Salidad: Numero
Precondición: Ninguna
Poscondición: Ninguna

Insertar (L: Lista; P: dirección; E: Elemento)
utilidad: Inserta el elemento E en la lista L como predecesor del elemento que ocupa la dirección P en la lista. Si P es la dirección Fin de la lista L entonces el elemento E pasa a ser el penultimo elemento de la lista tras la operación de inserción. El valor P, así como el de cualquier otro caso instancia del tipo de dato dirección existe antes de la operación de inserción, queda indefinido tras ejecutarse la operación
Entrada: Lista L
Salida: Ninguna
Precondición: La dirección P es la dirección de un elemento de lista L o bien la dirección de la lista
Poscondición: Lista L incrementada en un elemento más E

Insertar_Primero (L: Lista; G: Elemento)
utilidad: Insertar elemento E en la lista L como predecesor del elemento que ocupa el primer lugar en la lista
Entrada: Lista L
Salida: Ninguna
Precondición: La dirección
Poscondición: Lista L incrementada en un elemento más E

Insertar_ultimo (L: Lista, E: Elemento)
Utilidad: Insertar el elemento E en la lista L como último elemento de la misma, el elemento E ocupara el último lugar en la lista
Entrada: Lista L
Salida: Ninguno
Precondición: Ninguna
Poscondición: Lista L incrementada en el elemento más E

Nº _____ Tema _____

Fecha ___/___/___

Luthian Rincon Chayre

Suprime (L: lista; P: dirección)
Utilidad Elimina de la lista L el elemento que ocupa la dirección P. El valor de P, así como el de cualquier otra copia o instancia del tipo de datos dirección existente ante la operación de eliminación quedan indefinidos tras ejecutarse la operación.
Entrada Lista L
Salida Ninguna
Precondición La lista L es no vacía, la dirección P es la dirección de un elemento de lista L
Poscondición: Lista L decrementa en un elemento más, el elemento E de la dirección ya no pertenece a la lista L

Modifica (L: lista; P: dirección; E: tipoelem)
Utilidad Modificar el elemento que ocupa la dirección P de la lista L, cambiando por un nuevo elemento E
Entrada Lista L
Salida Ninguna
Precondición La lista L es no vacía. La dirección P es la dirección de un elemento de lista L
Poscondición Lista L modificada

2.1.3 Aplicaciones con Lista
En esta sección se puede apreciar que ya estamos en condiciones para plantear algoritmos usando el TDA Lista, abstrayéndonos de la forma como esta implementado

Ejem 1: Implementar un algoritmo que busque un elemento en la lista L y retorne la Dirección donde se encuentre, caso contrario Nulo

```
Dirección buscar (Lista L, Tipo Elemento elemento)
Inicio
    Si (L.vacia())
    entonces retornar nulo;
    caso Contrario
    Inicio
    P= L.primero()
    mientras P<>Nulo
     Inicio
     e = L.recuperar(P)
     si e = elemento entonces retornar P
     P = L.siguiente(P)
    Fin
    Fin
      retorna nulo;
Fin
```

Tema: _____ N°: _____

Fecha: ___/___/___

Revisión Tercera Clase

Ejemplo 2 Implementar un algoritmo que muestre el contenido de una lista

```
Inicio
  si (L.vacia()=verdadero)
  entonces retornar // termina proceso
  Caso contrario
  inicio
    P=L.Primero()
    mientras p<>nulo
    inicio
      e=L.recuperar(p)
      mostrar (e," ")
      p=L.siguiente(p))
    fin
    Fin
  fin
```

2.1.4 Implementación del TDA Lista

La implementación de los TDA pueden variar sin embargo el comportamiento del TDA lista no cambiara por lo tanto que se plantea en este capítulo dos formas de implementación sin que esto signifique que son las únicas formas

2.1.4.1 Implementación con vector

Para la implementación del TDA Lista se utilizara un vector y un atributo denominado longitud, donde el vector sera el que contendrá los elementos y longitud guardara la cantidad de elementos contenidos en la lista, se hace notar que los indices que nos permiten manipular el vector se constituyen en direcciones para efectos de la implementación

Definición de clase

Constante max=100

Nulo=0

Tipo de dato:

crear()

Dirección fin()

Dirección primero()

Dirección siguiente(dirección)

Dirección Anterior(dirección)

Boolean vacia()

TipoElemento recupera(dirección)

entero longitud()

WINNER

N° ______ Tema: ______

Fecha: ___ / ___ / ___

Luis Iñigo Pereira Choque

Insertar (dirección, elemento)
insertar_primero (elemento)
insertar_ultimo (elemento)
suprime (dirección)
modificar (dirección, elemento)
fin definición clase

Implementación de clase Lista
Lista crear()
inicio
    longitud = 0;
fin
    https//youtu.be/ksghCv-i6yg

Dirección Lista fin()
inicio
    si no vacio() entonces retornar longitud
        caso contrario // llamar a error

Dirección Lista primero()
inicio
    si no vacia() entonces
        retornar 1 // Retorna 1 por que en esa dirección esta el primero
        caso contrario
            // llamar a excepción ListaVacia
    fin

Dirección Lista siguiente (P dirección)
inicio
    si no vacia() entonces
        retornar 1 // retorna 1 por que en esa dirección esta el primero
        caso contrario
            // llamar a excepción ListaVacia
fin

Dirección Lista siguiente (P dirección)
inicio
    si vacia() entonces // llamar a excepción ListaVacia
    Caso Contrario
    Inicio
        si P = longitud entonces // llamar excepción DirecciónErr
        caso contrario
            retornar (P+1)
fin
fin

Tema: N°

Lista Ligada Pendiente Chequeo Fecha: / /

```
Dirección: Lista.anterior (P Dirección)
inicio
    Si vacio() entonces // llamar a excepción ListaVacia
    Caso contrario
    inicio
    Si p = 1 entonces // llamar excepción DireccionPrimerElem
                 caso contrario
                 retornar (p-1)
    fin
    fin

Boolean Lista.vacia()
    inicio
        retornar (longitud = 0)
    fin

TipoElemento Lista.recuperar(P Dirección)
    inicio
      Si vacia() entonces // llamar a excepción ListaVacia
       Caso contrario
       inicio
          Si no (p >= 1 y p <= longitud)
          entonces
          // llamar a excepción Dm
          Caso contrario
                retornar elementos[p]
          fin
      fin

  Entero Lista.longitud()
  inicio
      retornar Longitud
  fin

Lista.insertar (P Dirección, elemento TipoElemento)
Inicio
    Si longitud = max
    entonces // llamar a excepción ListaLlena
    Si no (p >= 1 y p <= longitud)
        entonces // llamar a excepción Direcc. Entro.
    Para cada i = (longitud + 1) descendente hasta (p + 1)
    inicio
          elementos[i] = elementos[i-1]
    fin
          elementos[p] = elemento
          longitud = longitud + 1
    fin
```

Nº _______ Tema: _______
Fecha: ___ / ___ / ___ Luis Lizaro Porcona Choque

lista insertar_primero (elemento TipoElemento)
Inicio
si longitud = max
entonces // llamar a excepcion lista llena
para cada i (longitud +1) descontando hasta 2
inicio
elemento [i] = elementos [i-1]
fin
elemento [1] = elemento
longitud ← longitud + 1
fin

lista insertar_ultimo (elemento TipoElemento)
Inicio
si longitud = max
entonces // llamar la excepción
caso contrario
longitud = longitud +1
elementos [longitud] = elemento
fin

lista suprime (p Dirección)
Inicio
si longitud = 0
entonces // llamar a excepción lista vacía
si no (p >= 1 y p <= longitud)
entonces // llamar a excepción direccionErr
para cada i = p hasta (longitud - 1)
inicio
elemento [i] = elemento [i+1]
fin
longitud ← longitud - 1
fin

lista modifica (p Dirección, elemento TipoElemento)
inicio
si longitud = 0
entonces // llamar a exception lista vacía
si no (p >= 1 y p <= longitud)
entonces // llamar a excepción direccionErr
elemento [p] = elementos
fin

Unidad Tercer Corte

2.1.4.2 Implementación con simulación de memoria
(usando la clase CsMemoria)

Esta forma de implementación es netamente académica en virtud a lo que busca es ver mejor conceptos sobre los punteros, para ello se entiende que se usara como memoria nuestra clase Cs memoria implementada en la unidad uno

Definida la clase Lista
Tipo de dato
Nodo
elemento Tipo de Dato
Sig Puntero a Nodo (valor entero para esta implementación)
fin
Dirección Puntero a Nodo (valor entero para esta implementación)

Clase Lista
Atributos
Ptr Elementos Dirección
longitud de tipo Entero
Metodos
crear()
Dirección Fin()
Dirección primero()
Dirección siguiente (dirección)
Dirección anterior (dirección)
booleano vacio()
Tipo Elemento recupera (dirección)
entero longitud()
insertar (dirección, elemento)
insertar_primero(elemento)
insertar_ultimo(elemento)
suprimir(dirección)
modificar (dirección, elemento)
Fin definición de clase

Implementación de clase lista utilizando Simulador de Memoria Cs memoria
Lista.crear()
Inicio
longitud=0 ptr.Elementos=nulo
fin

Dirección Lista.Fin()
Inicio
si vacio() entonces // llamar a excepción lista vacia
caso contrario
inicio
X=Ptr.Elemento
mientras X <> nulo
Ptr.Fin = X
X = M.obtener_dato(X, '→sig')
fin mientras
retornar Ptr.Fin
fin
fin

Lunes 20 Tercera Clase Lunes

```
lista.insertar(P Direccion, E TipoElemento)
Inicio // x tendria direccion de memoria si existe espacio
  x = M.new_espacio ('elemento, sig')
  si x <> nulo entonces
  inicio
     m.poner_dato (x, '->elemento', E), m.poner_dato (x'->sig', Null)
  si vacia() entonces PtrElementos = x, longitud = 1
    caso contrario inicio
    longitude = longitude +1
    si P = primero() entonces m.poner_dato(x,'->sig', P)
        PtrElementos = x
      caso contrario
       ant = anterior (P)
       m.poner_dato(ant,'->sig', x)
       m.poner_dato (x,'->sig', P)
    fin
  fin
     caso contrario // llamar a excepcion existe espacio memoria
  fin

lista.insertar_primero (E TipoElemento)
  Inicio // x tendria direccion de memoria si existe espacio
     x = M.New_espacio ('elemento, sig')
     si x <> nulo entonces
     inicio
        m.poner_dato (x, '->elemento', E)
        m.poner_dato (x, '->sig', PtrElementos)
        longitud = longitud +1
        PtrElementos = x
     fin
        caso contrario // llamar a excepcion existe espacio memoria
  fin

lista.insertar_ultimo (E TipoElemento)
  Inicio // x tendria direccion de memoria si existe espacio
     x = M.New_espacio ('elemento, sig')
     si x <> nulo entonces
       inicio
          m.poner_dato (x, '->elemento', E)
          m.poner_dato (x, 'sig', Null)
        si longitud <> 0 entonces m.poner_dato(Fin(), '->sig', x)
           caso contrario PtrElementos = x
       longitud = longitud +1
       fin
          caso contrario // llamar a excepcion existe espacio memoria

Direccion Lista.primero()
    Inicio
        si no vacia() entonces
          retornar PtrElementos // apunta en mero elemento
          caso contrario
                // llamar a excepcion ListaVacia
        fin
```

Luis Ishiro Pereira Chagoc

```
Direccion Lista.siguiente (p direccion)
Inicio
  si vacio() entonces // llamar a excepcion ListaVacia
  caso contrario
  inicio
    si p = fin() entonces // llamar excepcion DireccionErr
      caso contrario
      retornar (m.obtener_dato(p, '->sig'))
  fin
  fin

Direccion lista.anterior (p direccion)
  inicio
    si vacia() entonces // llamar a excepcion ListaVacia
    caso contrario
    inicio
      si p = primero() entonces // llamar excepcion DireccionPrimeroErr
    caso contrario
      INICIO
      x = Ptr Elemento
      ant = nulo
      mientras x <> Nulo
      inicio
        si x = p entonces retornar ant
        ant = x
          x = m.obtener_dato(x, '->sig')
      fin
    fin
  fin
  fin

Booleano Lista vacia()
  inicio
      retornar (longitud > 0)
  fin

TipoElemento Lista recuperar (p direccion)
Inicio
    si vacia() entonces // llamar a excepcion ListaVacia
      caso contrario retornar m.obtener_dato(p, '->Elemento')
  fin

Entero lista.longitud()
Inicio
    retornar logitud
  fin
```

Lunes Tercera Chequeo

```
Lista_suprimir (CP Dirección)
Inicio
    Si longitud = 0 entonces // llamar a excepción listavacia
        Si p = ptrelementos entonces // es el primer nodo
        Inicio
            x = ptrelementos
            ptrelementos = M.obtener_dato (ptrelementos, '→sig')
            // Liberar espacio de memoria x
        Fin
        Caso contrario
        Inicio
            ant = anterior (p)
            poner_dato (ant, '→sig', siguiente (p))
            // Liberar espacio de memoria p
        Fin
        longitud = longitud - 1
Fin

Lista modificar (CP Dirección, elemento tipoelemento)
Inicio
    Si vacia () entonces // llamar a excepción listavacia
        M.poner_dato (p, '→elemento', elemento)
Fin
```

2.1.4.2 Implementación con punteros

En esta última forma de implementación planteada lo que se resalta son los cambios que tienen que hacerse al código de la implementación con el Simulador de memoria considerado que ahora se está trabajando con punteros, así como se tiene de color rojo los cambios fundamentales en los algoritmos y a vuelo que dando por resueltas las definiciones formales en C++

Definiendo la clase Lista

```
Tipo de dato
        Nodo
            elemento Tipo Elemento
            Sig Puntero a Nodo
        Fin
        Direccion Puntero a Nodo

clase Lista
    Atributos
        PtrElementos Dirección
        longitud de tipo Entero
    Metodos
        Crear ()
        Direccion Fin ()
        Direccion Primero ()
```

Gestión Pericos Ehaye

T

Dirección primero ()
Dirección siguiente (Dirección)
Dirección anterior (Dirección)
booleano vacia ()
Tipo elemento recupera (Dirección)
Entero longitud ()
inserta (Dirección, elemento)
inserta_primero (elemento)
inserta_ultimo (elemento)
Suprime (dirección)
modifica (dirección, elemento)
Fin definición clase

Implementación clase lista utilizando punteros
Lista_crear ()
    inicio
        longitud = 0 ptrElementos = Nulo
    Fin

Dirección Lista_primero ()
    inicio
        si ¬ vacia () entonces
            retornar ptrElementos // porque en esa dirección
                // esta el primer elemento
        caso contrario
            // llamar a excepción ListaVacia
    Fin

Dirección Lista_fin ()
  inicio
      si vacia () entonces // llamar a excepción listaVacia
        caso contrario
        inicio
          x = ptrElementos
          mientras x <> nulo
          ptrFin = x
            x = x → sig // con simulador m_obtener_dato (x, '→sig')
          fin mientras x=>
            retornar ptrFin
            fin
        Fin

Lista insertar (PDirección, E tipoElemento)
inicio // x tendrá dirección de memoria si existe espacio
  x = new nodo // con simulador M. New - espacio ('elemento, sig')
    si x<>nulo entonces
      inicio
        x → elemento = E // con simulador m_poner_dato (x'→elemento', E)
        x → sig = Nulo // M. poner_dato (x'→sig', Nulo)
    Si vacia() entonces ptrElementos = x, longitud = 1
      caso contrario inicio
        longitud = longitud + 1
        si P = primero() entonces x → sig = P //

Luis Ángel Pérez Chayer

```
while (x != Null)
    if (x == d)
      return ant;
    ant = x;
    x = x->Sig;
  }
    return Null;
  }
}

bool ListaP :: vacio (){
    return long == 0;
}
int ListaP :: recupera (direccion d){
    if (vacia())
      cout << "Error: Lista vacia";
    else
      return d -> elemento;
}
int ListaP :: longitud (){
    return long;
}
void ListaP :: inserta (direccionP d, int e){
Nodo L * x = New Nodo L
  if (x != Null){
    x -> elemento = e;
    x -> Sig = NULL;
    if (vacia()){
      ptr->elemento = x
      long = 1;
  }
  else {
      long ++;
      if (d == primero()){
        x -> Sig = d;
          ptr->elemento = x;
    } else{
          direccionP ant = anterior (d)
        ant -> Sig = x;
          x -> Sig = d;
      }
    }
      else
        cout << "Error: No existe espacio de memoria".
}
```

Luciano Porcel Chaparro

```
Void ListaP::inserta_primero(int e){
    NodoL * X = New NodoL;
    if (X != NULL){
      X -> elemento = e;
      X -> sig = ptr_elementos;
      ptr_elementos = X;
       long ++;
   }
   else
    cout << "Error existe espacio de memoria";
}

Void ListaP::inserta_ultimo(int e){
    NodoL * X = new NodoL
      if (X != NULL){
      X -> elemento = e;
       X -> sig = NULL;
       long ++;
        if (vacio())
          pts_elementos = X;
       else
          Fin() -> sig = X;
    }
      else
          cout << "error: No existe espacio de memoria"
   }

Void ListaP::suprimir(direccionP d){
   if (long == 0){
         cout << "Error: Lista vacia";
         return;
    }
   if (d == ptr_elementos){
   direccionP X = ptr_elementos;
   ptr_elementos = ptr_elementos -> sig;
     delete(X);
  } else {
         direccionP ant = anterior(d);
         ant -> sig = d -> sig;
         delete(d)
    }
         long--;
}
Void ListaP::modificar(direccionP d, int e){
      if (vacia())
         cout << "Error: Lista vacia";
      else
         d -> elemento = e;
}
```

Excepción Persona Cheque

```
P ptrElemento; = x
Caso contrario
   ant = anterior(P)
   ant → sig = x   // con simulador m.poner_dato(ant,'→sig',x)
   x → sig = P     // con simulador m.poner_dato(x,'→sig',x)
Fin
   Fin
   Caso contrario // llamar a excepción existe espacio memoria
Fin

lista.inserta_primero (E TipoElemento)
Inicio // x tendrá dirección de memoria si existe espacio
x = New Nodo
Si x<> nulo entonces
   Inicio
      x → elemento = E      // con simulador m.poner_dato(x,'→elemento',E)
      x → sig = ptrElemento // con simulador m.poner_dato(x,'→sig',Elemento)
      longitud = longitud + 1
      ptrElemento = x
   Fin
   Caso contrario // llamar a excepción existe espacio memoria
Fin

Booleano Lista vacia()
   Inicio
      retornar (longitud = 0) // o (ptrElemento = nulo)
   Fin

lista inserta_ultimo (E TipoElemento)
   Inicio // x tendrá dirección de memoria si existe espacio
   x = New Nodo // con simulador New - espacio (elemento, sig)
   Si x<> nulo entonces
   Inicio
      x → elemento = E // con simulador m.poner_dato(x,'→elemento',E)
      x → sig = Null   // con simulador m.poner_dato(x,'→sig',Null)
      si longitud <> 0 entonces
            final() → sig = x // con simulador m.poner_dato(final()
                              → sig', x)
      Caso contrario ptrElemento = x
   longitud = longitud + 1
   Fin
   Caso contrario // llamar a excepción existe espacio memoria
   Fin

Dirección lista retorna (P dirección)
Inicio
   Si vacia() entonces // llamar a retornar Lista Vacia
   caso contrario
   Inicio
Si P = primero() entonces // llamar excepción DireccionPrimeroEn
   caso contrario
   Inicio
```

Luciana Poccori Choque

```
si p = primero() entonces // llamar excepcion DireccionPrimeroErr
  caso contrario
  inicio
     x = ptrElementos
     ant = nulo
     mientras x <> Nulo
     inicio
        si x = p entonces retornar ant
     ant = x
        x = x -> sig // con simulador m.obtener_dato(x, '->sig')
     fin
     fin
  fin
fin

TipoElemento lista recupera (P direccion)
  inicio
    si vacio () entonces // llamar a excepcion ListaVacia
     caso contrario retornar P -> elemento // con simulador m.obtener
              _dato (P, '-> Elemento')
  fin
      end

entero lista longitud()
     inicio
        retornar longitud
     fin

lista modifica (P Direccion, elemento tipo elemento)
inicio
   si vacia() entonces // llamar a excepcion lista vacia

         P -> elemento = elemento // con simulador m.poner_dato
         (P, '-> elemento', elemento)
   fin

lista suprime (P Direccion)
   inicio
      si longitud = 0 entonces // llamar a excepcion listavacia
      si P = ptrelementos entonces // es el primer nodo
      inicio
           x = ptrelementos
           P trelementos = Ptrelementos -> siguiente // con simulador
                              M.obtener_dato (Ptrelementos, '->sig')
        delete x // con simulador m.delete_espacio(x)
      fin
       caso contrario
        inicio
        ant = anterior (P)
        ant -> sig = siguiente (P) // con
```

Gestión Ponce de León

TOP

```
Fin longitud = longitud - 1
Fin

lista modificar(P Direccion, elemento tipo elemento)
vacio
    si vacia() entonces // llamar a excepción lista vacia
    P -> elemento = elemento // consumidor me pone dato
                        (P, -> elemento, elemento)
Fin

    lista con punteros

ULista.PH

# ifndef ULista_PH
# define ULista_PH

# include <iostream>
# include <string>
# include "funciones.h"

Using namespace std;
struct NodoL{
    int elemento;
    NodoL * sig;
};
    Typedef NodoL* direccionP;

class ListaP{
private:
        direccionP ptr_elemento
        int longg;

Public:
        ListaP()
        direccionP primero();
        direccionP siguiente(direccionP d);
        direccionP anterior(direccionP d);
        bool vacio();
        int recupere(direccionP d);
        int longitud();
        void inserta(direccionP d, int e);
        void inserta_primero(int e);
        void inserta_ultimo(int e);
        void suprime(direccionP d);
        void modifica(direccionP d, int e);
    string to_str();
};
# endif
```

Luchiño Pancasa Choque

```
# pragma hdrstop
# include "ULista P.h"
# pragma package (smart_init)

Lista P:: Lista P () {
    ptr_elementos = Null;
    long = 0;
}
direccionP lista P :: fin () {
  if (vacia ())
      cout << " Error : Lista vacia "
  else {
    direccionP X = ptr_elementos;
    direccionP ptr_fin;
    while (X != Null) {
      ptr_fin = X;
      X = X -> sig;
    }
      return ptr_fin;
  }
}
    direccionP Lista P : : primero() {
  if (! vacia ())
        return ptr_elementos;
    else
      cout << " Error : Lista vacio";
}

direccionP Lista P : siguiente (direccionP) {
   if (vacio ())
      cout << " Error : Lista vacia"
   else {
        if (d == fin ())
      cout << " Error direccion erroneo",
      else
           returno d -> sig ;
   }
}

direccionP Lista P : : anterior (direccionP d) {
   if (vacio ())
        cout << " Error : Lista vacio"
    else {
        if (d == primero ())
           cout << " Error : direccion erronea"
   else {
       direccionP x = ptr_elementos,
       direccionP ant = Null;
```