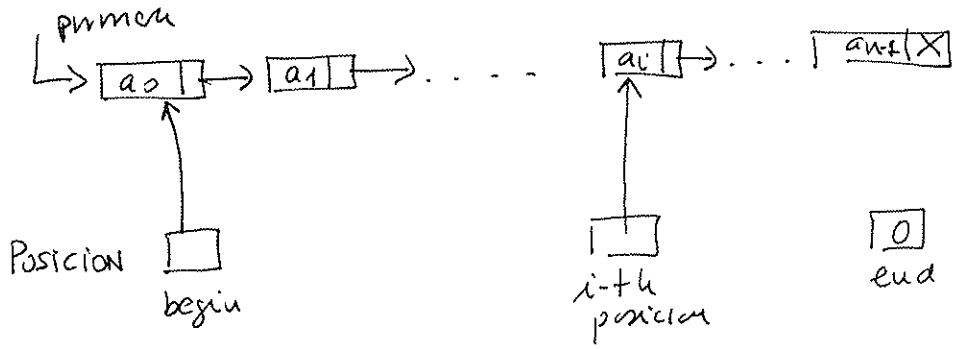
LECCION 12 LISTAS.

_DIFERENTES IMPLEMENTACIONES CON CELDAS ENLAZADAS

1) CELDAS SIMPLES: UN UNICO PUNTERO.



_Problemas

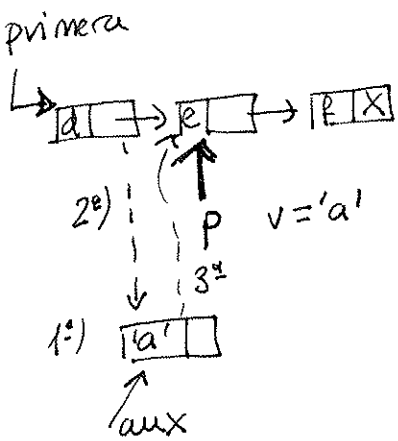
P1 Debemos usar dos punteros para implementar T.D.A Posicion. debido al operator--.

P2 El operator-- gasta O(n).

Ver transparencias 1 y 2.

Ejemplo de Implementacion de Insertar y Borrar.

```
void Lista::Insertar(Posicion p, Tbase v){
    Celda *aux = new Celda; aux->elemento = v;
    if (p == begin()) { //es la primera posicion.
        aux->sig = primera;
        primera = aux;
    }
    else {
        Posicion q = p;        --q; // -- gasta O(n)
        aux->sig = p.punt;
        q.punt->sig = aux;
    }
}
```

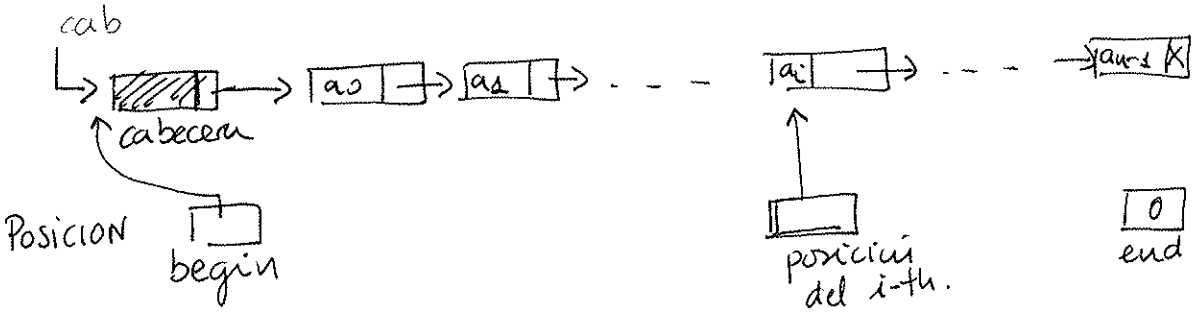


LECCIÓN 12 LISTAS

```
void Lista::Borrar(Posicion p){
    if (p==begin()){
        Celda * aux=primera;
        primera=primera->sig;
        delete aux;
    }
    else{
        Posicion q=p; --q;
        Celda * aux=p.punt;
        q->sig=p->punt->sig;
        delete aux;
    }
}
```

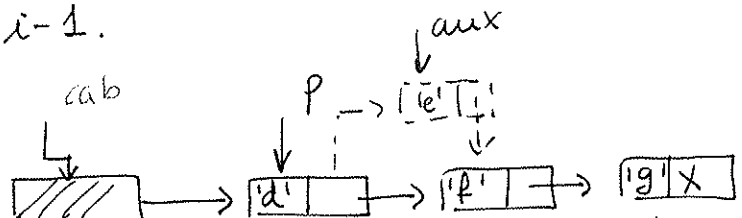
Como primer intento para resolver P1 y P2

2) CELDAS ENLAZADAS CON CABECERA



- Controlamos la posicion del i-th elemento desde una posicion i-1.

Ej:



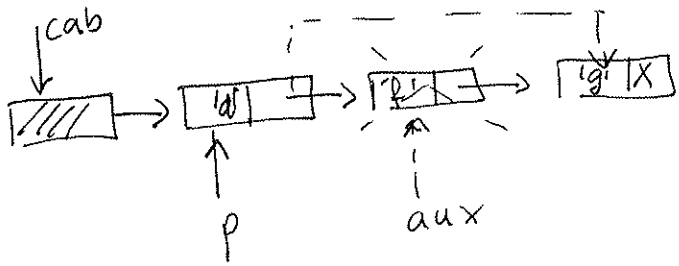
Insertar 'e' desde p → insertarla entre 'd' y 'f'


```
Celda * aux=new Celda;
aux->elemento=v;
aux->sig=p.punt->sig;
p.punt->sig=aux
```

⟹ SENTENCIAS PARA INSERTAR DESDE P.

LECCIÓN 12: LISTAS

Borrar el elemento i-esimo




```
Celda * aux = p.punt->sig;
p.punt->sig = aux->sig;
delete aux;
```

Problemas

P1: Una posicion sigue gastando dos punteros. para poder aplicar el operador -- desde O(end)

//IMPLEMENTACION CELDAS ENLAZADAS CON CABECERA

```
typedef char Tbase;
struct Celda{
   Tbase ele;
   Celda *sig;
};
class Lista;
class Posicion{
   private:
      Celda * punt;
      Celda * primera;
   public:
      Posicion(): punt(0), primera(0){}
      bool operator==(const Posicion &op){
            return punt==op.punt;
      }
      bool operator!=(const Posicion &op){
            return punt!=op.punt;
      }
};
```

LECCION 12: LISTAS

```
Posicion & operator ++(){
    assert (punt != 0);
    punt = punt->sig;
    return *this;
}
Posicion & operator --(){
    if (primera == punt)
        punt = 0;
    else {
        celda * q = primera;
        while (q->sig != punt)
            q = q->sig;
        punt = q;
        return *this;
    }
}
Tbase & operator *() {
    return punt->sig->ele;
}
friend class Lista;
};
}; // end Posicion.
```

LECCION 12: LISTAS

(continuación de la implementación de listas con cabecera)

```
class Lista {
  private:
    Celda * cab;
    void Copiar(const Lista &L);
    void Borrar_All();
  public:
    Lista();
    Lista(const Lista &L);
   ~Lista();
    Lista & operator=(const Lista &L);
    void Insertar(Posicion p, Tbase v);
    void Borrar(Posicion p);
    Tbase Get(Posicion p)const;
    void Set(Posicion p, Tbase v);
    int size()const;
    Posicion begin()const;
    Posicion end()const
};
```

```
void Lista::Copiar(const Lista &L){
    if (L.cab->sig==0){
        cab=new Celda;
        cab->sig=0;
    }
    else{
        cab=new Celda;
        Celda *p=cab, *q=L.cab;
        while(q->sig!=0){
            p->sig=new Celda;
            p->sig->ele=q->sig->ele;
            p=p->sig;
            q=q->sig;
        }
        p->sig=0;
    }
}
```

```
void Lista::Borrar_All(){
    while(cab->sig!=0){
        Celda *aux=cab->sig;
        cab->sig=cab->sig->sig;
        delete aux;
    }
    delete cab;
}
Lista::Lista(){
    cab=new Celda;
    cab->sig=0;
}
Lista::Lista(const Lista &L){
    Copiar(L);
}
Lista::~Lista(){
    Borrar_All();
}
Lista & Lista::operator=(const Lista &L)
{   if(this!=&L){
        Borrar_All();
        Copiar(L);
    }
    return *this;
}
void Lista::Insertar(Posicion p, Tbase v){
    Celda *aux=p.punt->sig;
    p.punt->sig=new Celda;
    p.punt->sig->ele=v;
    p.punt->sig->sig=aux;
}
void Lista::Borrar(Posicion p){
    assert(p.punt->sig!=0);
    Celda *aux=p.punt->sig;
    p.punt->sig=p.punt->sig->sig;
    delete aux;
}
```

LECCION 12 : LISTAS

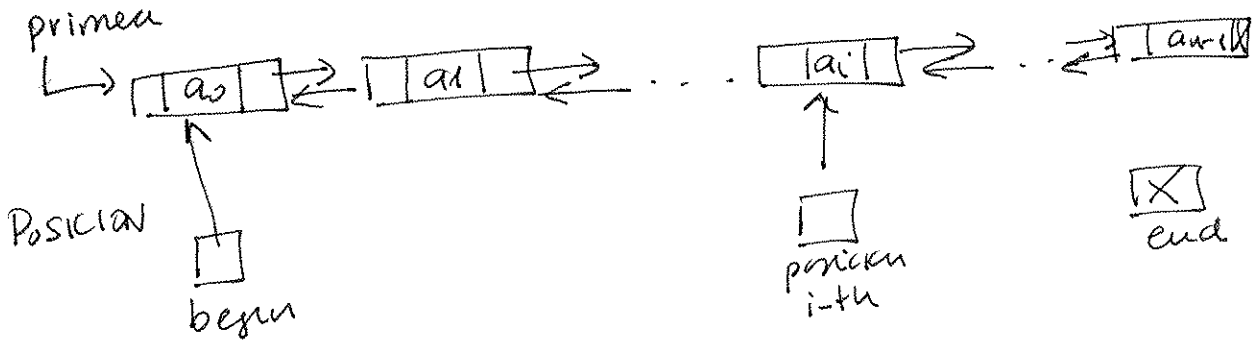
```
Posicion Lista::begin() const{
    Posicion p;
    p.punt = cab;
    p.primera = cab;
    return p;
}
Posicion Lista::end() const{
    Posicion p;
    Celda * aux = cab;
    while (aux->sig != 0){
        aux = aux->sig;
    }
    p.punt = aux;  p.primera = cab;
    return p;
}
```

Problemas asociados con la implementacion de la listas con celdas enlazadas y con cabecera:

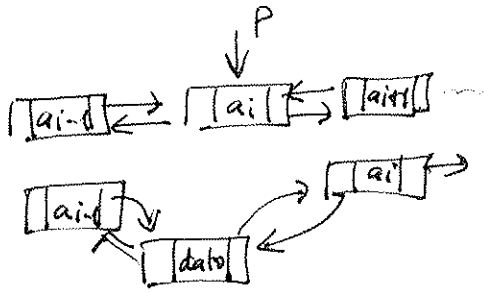
- Dos puntero para posicion para poder implementar --
- La posicion no se refiere al nodo sino al siguiente.

LECCIÓN 12.-

LISTAS: Celdas Doblemente Enlazadas



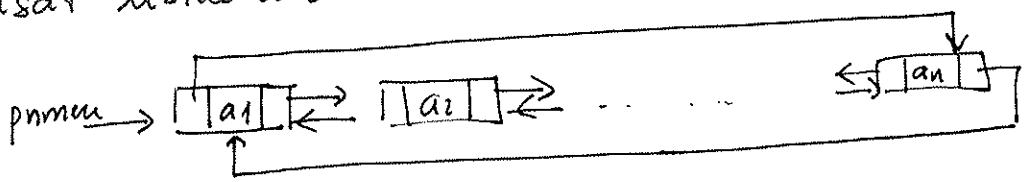

```
struct celda{
    char d;
    Celda *sig;
    Celda *ant;
};
```




```
void Lista::Insercion(Posicion p, char e){
    Celda *aux = new Celda;
    aux->d = e;
    if (p == begin()){
        aux->sig = primera;
        if (primera != 0)
            primera->ant = aux;
        aux->ant = 0;
        primera = aux;
    }
    else{
        aux->sig = p.punt; Posicion q = p;
        q.punt->sig = aux;
        aux->ant = q.punt;
        if (p.punt != 0)
            p.punt->ant = aux;
    }
}
void Lista::Borrar(Posicion p){
    Celda *aux = p.punt;
    if (p == begin()){
        primera = primera->sig;
        if (primera != 0)
            primera->ant = 0;
        delete aux;
    }
    else{
        p.punt->ant->sig = p.punt->sig;
        if (p.punt->sig != 0)
            p.punt->sig->ant = p.punt->ant;
        delete aux;
    }
}
```

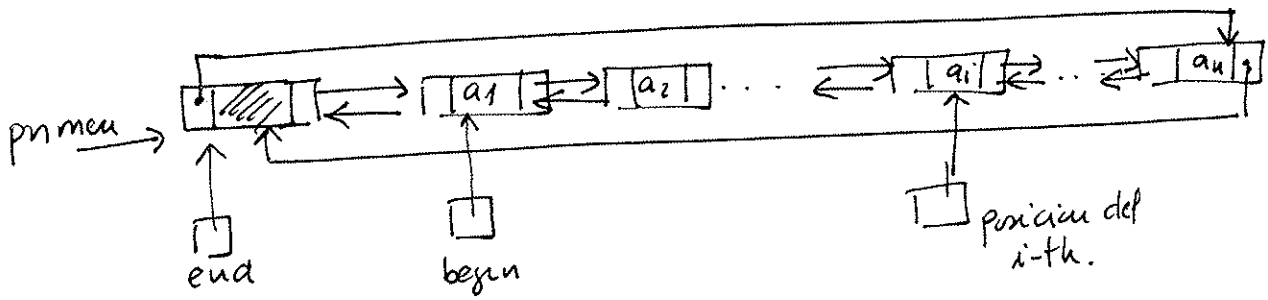
LECCIÓN 12: LISTAS

En las anteriores implementaciones hemos tenido un gran inconveniente para poder realizar la operación -- en la clase Posición. Esto se debe a que para implementar el operador -- necesitamos conocer el comienzo de la lista.
Para poder obtener el -- de la posición fin en tiempo cte podemos usar listas doblemente enlazadas circulares.

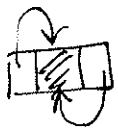


Con esta implementación aún tenemos que saber para cada Posición quién es la primera para detectar que si la sig == primera hemos llegado al fin.

Para que el T.D.A Posición solamente tenga un único puntero usamos CELDAS DOBLEMENTE ENLAZADAS CON CABECERA CIRCULARES



Lista vacía



LISTAS: Celdas ENLAZADAS

```
#ifndef LISTAS1_H
#define LISTAS1_H

typedef char Tbase;
struct Celda{
    Tbase elemento;
    Celda *sig;
};
class Lista;
class Posicion{
  private:
    Celda *punt;
    Celda *primera;
  public:
    Posicion(): punt(0), primera(0){}
    Posicion& operator++(){
        punt = punt->sig;
        return *this;
    }
    Tbase & operator *(){
        return punt->elemento;
    }
    Posicion& operator--(){
        Celda *aux;
        if (primera == punt)
            punt=0;
        else{
            aux=primera;
            while (aux->sig != punt && aux->sig !=0)
                aux=aux->sig
            punt=aux
        }
        return *this;
    }
    bool Operator==(const Posicion &p){
        return p.punt == punt;
    }
    bool operator!=(const Posicion &p){
        return p.punt != punt;
    }
    friend class Lista;
};
```

```
class Lista {
    private:
        Celda *primera;
    public:
        Lista();
        Lista(const Lista &L);
        ~Lista();
        Lista & operator=(const Lista &L);
        int size() const;
        void ~~Insertar~~ Insertar(Posicion p, Tbase v);
        void Borrar(Posicion p);
        Posicion begin() const {
            Posicion p;
            p.primera = primera;
            p.punt = primera;
            return p;
        }
        Posicion end() const {
            Posicion p;
            p.primera = primera;
            p.punt = 0;
            return p;
        }
        Tbase ~~Get()const~~ Get(Posicion p);
        void Set(Posicion p, Tbase v);
};
#endif
```

Listas (uso) (1/2).

Uso de lista

```
bool vacia(const Lista& l)
{
  return l.begin()==l.end();
}
int numero_elementos(const Lista& l)
{
  int n=0;
  for (Posicion p=l.begin(); p! =l.end(); ++p)
    n++;
  return n;
}
void todo_minuscula(Lista &l)
{
  for (Posicion p=l.begin(); p! =l.end(); ++p)
    l.set(p, tolower(l.get(p)));
}
void escribir(const Lista& l)
{
  for (Posicion p=l.begin(); p! =l.end(); ++p)
    cout << l.get(p);
  cout << endl;
}
void escribir_minuscula (Lista l)
{
  todo_minuscula(l);
  escribir(l);
}
```

Uso de lista

```
void borrar_caracter (Lista& l, char c)
{
  Posicion p=l.begin();
  while (p! =l.end())
    if (l.get(p)==c)
      p=l.borrar(p);
    else ++p;
}

Lista al_reves(const Lista& l)
{
  Lista aux;
  for (Posicion p=l.begin(); p! =l.end(); ++p)
    aux.insertar(aux.begin(),l.get(p));
  return aux;
}

Posicion localizar(const Lista& l, char c)
{
  for (Posicion p=l.begin(); p! =l.end(); ++p)
    if (l.get(p)==c) return p;
  return l.end();
}
```

Listas (uso) (2/2).

Uso de lista

```
int numero_elementos(const Lista& l);
void borrar_caracter (Lista& l, char c);
void todo_minuscula(Lista &l);

bool palindromo (const Lista& l)
{
  Lista laux(l);
  int n= numero_elementos(l);
  if (n<2)
     return true;

  borrar_caracter(laux,' ');
  todo_minuscula(laux);

  Posicion p1,p2;
  p1=laux.begin();
  p2=laux.end(); --p2;
  for (int i=0;i<n/2;i++) {
     if (laux.get(p1)! =laux.get(p2))
        return false;
     ++p1;--p2;
  }
  return true;
}
```

Note que *no sería válido* mover las posiciones *"mientras"* la primera sea *"menor"* que la segunda.

Uso de lista

```
int main()
{ char dato;
  Lista l;

  cout << endl<< "Escriba una frase" << endl;
  while ((dato=cin.get())! ='\n')
     l.insertar(l.end(),dato);

  cout << endl<< "La frase introducida es:"<<endl;
  escribir (l);
  cout << endl<< "La frase en minúscula:"<<endl;
  escribir_minuscula (l);
  if (localizar(l,' ')==l.end())
     cout << endl << "La frase no tiene espacios."<<endl;
  else {
     cout << endl<< "La frase sin espacios:"<<endl;
     Lista aux(l);
     borrar_caracter(aux,' ');
     escribir (aux);
  }
  cout << endl<< "La frase al revés:"<<endl;
  escribir (al_reves(l));
  if (palindromo(l))
        cout <<endl<<"Es un palíndromo"<< endl;
  else cout<<endl<<"No es un palíndromo"<<endl;

  return 0;
}
```

Listas (celdas doblemente enlazadas con cabecera circulares) (1/2).

lista.h

```
typedef char Tbase;
struct CeldaLista{
   Tbase elemento;
   CeldaLista *anterior;
   CeldaLista *siguiente;
};
class Lista;                          // Declaración adelantada
class Posicion {
     CeldaLista *puntero;
   public:
     Posicion(): puntero(0) {}
     //Posicion(const Posicion& p);
     //~Posicion();
     //Posicion& operator= (const Posicion& p);
     Posicion& operator++() {
       puntero= puntero->siguiente;
       return *this;
     }
     Posicion& operator--() {
       puntero= puntero->anterior;
       return *this;
     }
     bool operator==(const Posicion& p)
     { return puntero==p.puntero; }
     bool operator! =(const Posicion& p)
     { return puntero! =p.puntero; }
     friend class Lista;
};
```

Tbase &operator*(){ return puntero->elemento; }

lista.h

```
class Lista{
     CeldaLista *cab;
   public:
     Lista();
     Lista(const Lista& l);
     ~Lista();
     Lista& operator= (const Lista& l);

     void set (Posicion p, Tbase e) {
       p.puntero->elemento= e;
     }
     Tbase get (Posicion p) const {
       return p.puntero->elemento;
     }
     Posicion insertar(Posicion p, Tbase e);
     Posicion borrar(Posicion p);

     Posicion begin() const {
       Posicion p;
       p.puntero= cab->siguiente;
       return p;
     }
     Posicion end() const {
       Posicion p;
       p.puntero= cab;
       return p;
     }
};
```

Listas (celdas doblemente enlazadas con cabecera circulares) (2/2).

lista.cpp

```
Lista::Lista() {
  cab= new CeldaLista;
  cab→siguiente= cab;
  cab→anterior=cab;
}
Lista::Lista(const Lista& l){
  cab= new CeldaLista;
  cab→siguiente= cab;
  cab→anterior=cab;

  CeldaLista *p= l.cab→siguiente;
  while (p! =l.cab) {
    CeldaLista *q;
    q= new CeldaLista;
    q→elemento=p→elemento;
    q→anterior= cab→anterior;
    cab→anterior→siguiente= q;
    cab→anterior= q;
    q→siguiente =cab;
    p= p→siguiente;
  }
}
Lista::~Lista() {
  while (begin()! =end())
    borrar(begin());
  delete cab;
}
```

lista.cpp

```
Lista& Lista::operator= (const Lista& l){
  Lista aux(l);
  CeldaLista *p;
  p=this→cab;
  this→cab= aux.cab;
  aux.cab=p;
  return *this;
}
Posicion Lista::insertar(Posicion p, Tbase e) {
  CeldaLista *q= new CeldaLista;
  q→anterior= p.puntero→anterior;
  q→siguiente=p.puntero;
  p.puntero→anterior= q;
  q→anterior→siguiente= q;
  q→elemento= e;
  p.puntero=q;
  return p;
}
Posicion Lista::borrar(Posicion p){
  assert (p! =end());
  CeldaLista *q= p.puntero;
  q→anterior→siguiente= q→siguiente;
  q→siguiente→anterior= q→anterior;
  p.puntero=q→siguiente;
  delete q;
  return p;
}
```