ED – 24/10/2017

- P1 -

Debemos usar dos punteros en Posicion para poder implementar el operator –

```
class Posicion{
	private:
		Celda *punt;
		Celda *primera;
}
```

- P2 -

Eficiencia del operator -- O(n)

void Lista::Insertar(Posicion p, Tbase, v){ //typedef Tbase char; -> Cada vez que ponga TBase, significa char.

```
	Celda *aux = new celda;
	aux->ele = v;

	if(p == begin()){
		aux->sig = primera;
		primera = aux;
	}

	else{
		Posicion q = p;
		--q;
		aux->sig = p.punt;
	}

}
```

Posicion begin(){

```
        Posicion p;
        p.punt = primera;
        p.primera = primera;
        return p;
}

Posicion end(){
        Posicion p;
        p.punt = 0;
        p.primera = primera;
}

void Lista::Borrar(Posicion p, Tbase, v){
        if(p == begin()){
                Celda *aux = primera;
                primera = primera -> sig;
                delete aux;
        }

        else{
                Posicion q = p;
                --q;
                Celda *aux = p.punt;
                q->sdig = p.punt->sig;
                delete aux;

        }
}
```

ITERADORES

T.D.A Posicion

Proceder

1) Inicia el iterador. Pej: Inicializarlo al principio del contador a traves del metodo begin del contenedor.

2) Se debe saber como avanzar - usando el operator++ del iterador

3) Debemos poder acceder al elemento del contenedor usando el iterador - A través del operator* del iterador.

4) Saber Cuando terminar de iterar sobre los elementos del contenedor - A traves del metodo end() del contenedor.

```
class Lista{
private:
	Celda *primera;
public:
	Lista();
	Lista(const Lista &L);
	~Lista();
	Lista operator=(const Lista &L);

	class iterator{
		private:
			Celda * punto;
		public:
			iterator():punt(0){};
			bool operator(const iterator i){
				return i.punt == punt;
			}

			bool operator!= (const iterator &i){
				return i.punt != punt;
			}
			char & operator*(){
				return *punt->ele;
			}
			iterator & operator++(){
					punt = punt->sig;
					return *this;
			}
			iterator & operator--(){
				punt = punt ->ant;
				return *this;
			}
```

```
                friend class Lista;
        }//Class iterator

        void Set(iterator p, char v){};
        char Get(iterator p);
        void Insertar(iterator p, char c);
        void Borrar(iterator p);
        iterator Begin(){
                iretator i;
                i.punt = primera -> sig;
                retunr i;
        }
        iterator end(){
                iterator i;
                i.punt = primera;
                return i;
        }

}//CLASS LISTA



int main(){

        Lista miLista;

        for(int oi=0; i<5; i++){
                miLista.Insertar(miLista.begin(), (char) i);
        }

        Lista::iterator it;

        for(it = miLista.begin(); it!=miLista.end(); ++it){
                cout<<*it;
                *it = 'a';
        }
}
```

Los ITERADORES van a ir dentro de la clase contenedora

```
#include "Lista.h"

void Imprimir(const Lista &l){

	Lista::iterator it; //Al ser constante la lista como argumento, iterator hay que pasarlo como const_iterator
                                                // pero hay que hace otra clase de const_iterator (Declarada abajo)

	for(it = l.begin(); it != l.end(); ++it){
		cout<<*it;
	}
}

int main(){
	Lista l1;

	for(char c == 'a'; c<= 'z'; ++c){
		li.Insertar(l1.end(), c);
	}

	Imprimir(li);
}


class const_iterator{
private:
	const Celda *punt;
public:
	const_iterator():punt(0){}
	bool operator==(const const_iterator &i){
		return punt == i.punt;
	}

	bool operator!=(const const_iterator &i){
		return punt != i.punt;
	}
```

```
        const char & operator*(){
                return punt->ele;
        }

        const_iterator begin(){
                const_iterator i;
                i.punt = primera->sig;
                return i;
        }

        const_iterator end(){
                const_iterator i;
                i.punt = primera;
                return i;
        }

        const_iterator &operator++(){
                punt = punt -> sig;
                return *this;
        }

        const_iterator &operator--(){
                punt = punt->ant;
                return *this;
        }

        friend class Lista;
}//Cklass const_iterator
```

Ejemplo nos piden crear un iterador sobre un vector dinamico que itere sobre los elementos pares.

```
bclass VD{
private:
        int *datos;
        int n;
        int reservados;
```

```
public:
        .
        .
        .
        class iterator{
        private:
                int *punt;
                int *final;
        public:
                bool operator == (const iterator &i){
                        return i.punt == punt;
                }

                bool operator!=(Const iterator &i){
                        return i.punt != punt;
                }

                int &operator *()
                        return *punt;
                }

                iterator &operator++(){
                        while((*punt)%2!=0 && finak !=punt)
                                        ++punt;
                        return *this;
                }

                iterator beign()
                iterator i;
                i.punt = &(datos[0]);
                i.final = &(&datos[n]);
                if(datos[0]%2!=0)
                        ++i;
                return i;
        }}
```