LECCION 11: LISTAS

ESPECIFICACIÓN: es un tipo de dato lineal que contiene una secuencia de elementos $\{a_0, a_1, \ldots, a_{n-1}\}$ especialmente diseñados para realizar las operaciones de inserción, borrado y consulta desde cualquier posición.

OPERACIONES

- Set: modifica un elemento de una posición.
- Get: devuelve un elemento de una posición
- Borrar: borra un elemento de una posición
- Insertar: Inserta un elemento de una posición.
- Size: devuelve el n: de elementos.

NOTA: Cuando tenemos una lista con n elementos las posiciones van desde la posición 0 a la n. Siendo la posición n donde se insertaría un nuevo elemento. Los elementos almacenados están en las posiciones 0 a n-1.

INTERFACE

```
#ifndef  _LISTA_H
#define  _LISTA_H
template <class T>
class Lista{
  private:
     //Implementacion

  public:
    Lista();
    Lista(const Lista<T>& L);
    ~Lista();
     Lista<T>& operator=(const Lista<T>&L);
     T Get(int posicion) const;
     void Set(int posicion, const T&e);
     int Size() const;
     void Insertar(int posicion, const T&e);
     void Borrar(int posicion);

};
#endif.
```

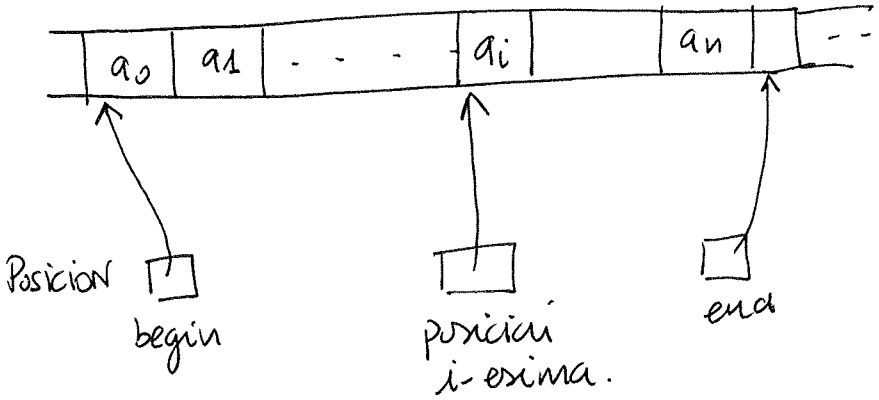
¿Posición?
└ Lugar en la lista donde queremos o bien insertar, consultar, modificar o borrar.

└→ ABSTRAER UN NUEVO T.D.A POSICION sobre UNA LISTA.

LECCION 11: LISTAS

Listas implementadas como vectores

Almacenamos la secuencia de valores, la lista, en un vector y controlamos cada posicion como un puntero al elemento sobre el que queremos operar.




```
#ifndef _LISTA_H
#define _LISTA_H.
 class Lista;
 template <class T>
 class Posicion{
    private:
       T* i;
    public:
       Posicion (): i(0){ }
       Posicion<T> & operator++ (){
                i++;
                return *this;
       }
       Posicion <T> & operator-- (){
                i--;
                return *this;
       }
       T & operator *(){
                return *(i);
       }
```

```
bool operator== (const Posicion<T> &p)
{
      return i==p.i;
}
bool operator != (const Posicion<T>
                  &p ){
      return i!=p.i;
}
friend class Lista;
};
template <class T>
class Lista{
  private:
    T * datos;
    int n; //almacenados
    int reservados;
    void Resize(int tam);
    void Copiar(const Lista<T>& L);
  public: Lista(int tam=10);
    Lista(const Lista<T> & L);
    ~Lista();
    Lista<T> & operator=(const Lista<T>&
    void set(Posicion<T> p, const T& e){
      assert(p.i!=0);
        *p.i = e;
    }
    T get(Posicion<T> p){
        return *p.i;
    }
    Posicion<T> Insertar(Posicion<T> p,
                         const T& e);
    Posicion<T> borrar(Posicion<T> P);
    Posicion<T> begin() const{
        Posicion<T> p;
        p.i = &(datos[0]);
        return p;
    }
```

③ LECCIÓN 11

```
Posicion end() const {
    Posicion p;
    p.i = &(datos[n]);
    return p;
}
};// end de class Lista
```

← #include "Lista.h" // Por ser clase template

// file Lista.cpp

```
template <class T>
Posicion <T> Lista <T> :: Insertar( Posicion <T> p, const T& e) {
    int pos = p.i - datos[0]; // desplazamiento relativo
    if (n==reservados)
        resize(2*reservados);

    Posicion <T> q, aux;
    q=end(); aux=q; --aux;
    p.i=&(datos[pos]); // si se ha hecho resize hacemos que apunte
                       // correctamente
    for (; q!=p; --q, --aux) {
        *q = *aux;
    }                                  — Abre hueco
    *q = e;   n++;
    return q
}
template <class T>
Posicion <T> Lista <T>:: Borrar (Posicion <T> p) {
    Posicion <T> siguiente=p;   ++siguiente;
    for (Posicion <T> q=p; siguiente!=end(); ++q, ++siguiente)
        *q = *siguiente;
    n--;
    int pos = p.i - &(datos[0]);
    if (n < reservados/4) resize(reservados/2);
    return p;
}
```

4

LECCIÓN 11: LISTAS

```
#include "Lista.h"

int main(){
    Lista<int> milista;

    for (int i=0; i<10; i++)
        milista.Insertar(milista.begin(), i);

    //milista tiene 9 8 7 6 5 4 3 2 1 0

    Posicion<int> p;

    for (p = milista.begin(); p!=milista.end(); p++)
        cout << *p;

    //al revés

    Posicion<int> q = --p;
    for ( ; q!=milista.begin(); --q)
        cout << *q;

    cout << *q; // Para imprimir 0.
```

3.