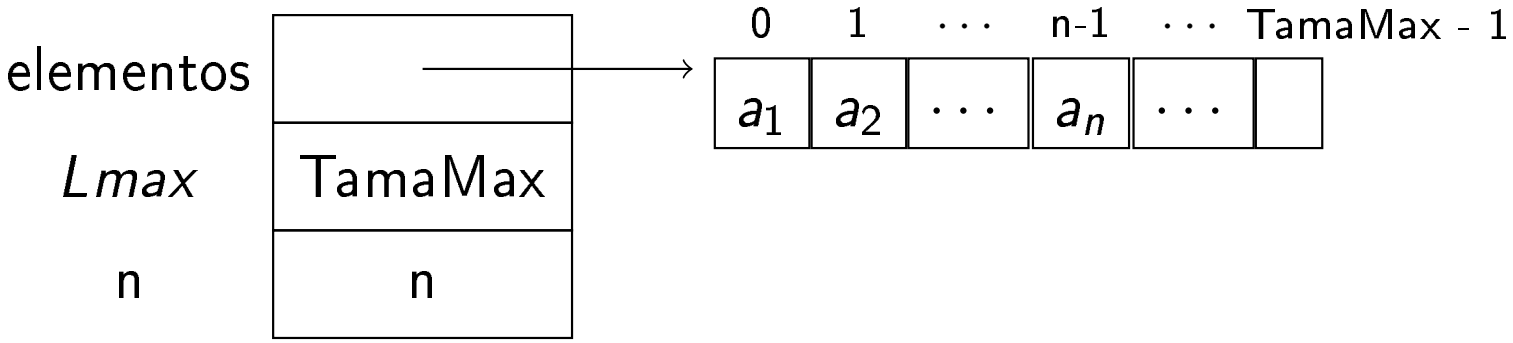
Implementación vectorial pseudoestática



Implementación vectorial pseudoestática

```
1  // listavec.h
2  //
3  // clase Lista genérica: vector pseudo-estático (en
4  // memoria dinámica) cuyo tamaño (parámetro de entrada
5  // del constructor) puede ser distinto para cada
6  // objeto de la clase Lista.
7  // Las variables externas de tipo posición, posteriores
8  // a la posición p en la que se realiza una inserción
9  // o eliminación, no cambian, pero sí los elementos
10 // que se encuentran en dichas posiciones.

12 #ifndef LISTA_VEC_H
13 #define LISTA_VEC_H
14 #include <cassert>
```

Implementación vectorial pseudoestática

```
16 template <typename T>
17 class Lista {
18 public:
19    typedef int posicion; // posición de un elto
20    explicit Lista(size_t TamaMax); // constructor
21    Lista(const Lista<T>& l); // ctor. de copia
22    Lista<T>& operator =(const Lista<T>& l); // asignación entre
          listas
23    void insertar(const T& x, posicion p);
24    void eliminar(posicion p);
25    const T& elemento(posicion p) const; // acceso a elto, lectura
26    T& elemento(posicion p); // acceso a elto, lectura/escritura
27    posicion buscar(const T& x) const; // requiere operador ==
          para el tipo T
28    posicion siguiente(posicion p) const;
29    posicion anterior(posicion p) const;
30    posicion primera() const;
31    posicion fin() const; // posición después del último
32    ~Lista(); // destructor
```

Implementación vectorial pseudoestática

```
33 private:
34    T *elementos; // vector de elementos
35    int Lmax; // tamaño del vector
36    int n; // longitud de la lista
37 };

39 // clase Lista genérica: vector pseudo-estático.
40 // Una lista de longitud n se almacena en celdas
41 // consecutivas del vector, desde 0 hasta n-1.
42 // La posición de un elemento es el índice de la celda
43 // en que se almacena.
44 //
45 // Implementación de operaciones

47 template <typename T>
48 inline Lista<T>::Lista(size_t TamaMax) :
49    elementos(new T[TamaMax]),
50    Lmax(TamaMax),
51    n(0)
52 {}
```

Implementación vectorial pseudoestática

```
54 template <typename T>
55 Lista<T>::Lista(const Lista<T>& l) :
56    elementos(new T[l.Lmax]),
57    Lmax(l.Lmax),
58    n(l.n)
59 {
60    for (Lista<T>::posicion p = 0; p < n; p++) // copiar el vector
61       elementos[p] = l.elementos[p];
62 }
```

Implementación vectorial pseudoestática

```
64 template <typename T>
65 Lista<T>& Lista<T>::operator =(const Lista<T>& l)
66 {
67    if (this != &l) { // evitar autoasignación
68       // Destruir el vector y crear uno nuevo si es necesario
69       if (Lmax != l.Lmax) {
70          delete[] elementos;
71          Lmax = l.Lmax;
72          elementos = new T[Lmax];
73       }
74       // Copiar el vector
75       n = l.n;
76       for (Lista<T>::posicion p = 0; p < n; p++)
77          elementos[p] = l.elementos[p];
78    }
79    return *this;
80 }
```

Implementación vectorial pseudoestática

```
82  template <typename T>
83  void Lista<T>::insertar(const T& x, Lista<T>::posicion p)
84  {
85     assert(p >= 0 && p <= n); // posición válida
86     assert(n < Lmax); // lista no llena
87     for (Lista<T>::posicion q = n; q > p; q--)
88        // desplazar los eltos. en p, p+1, ...
89        elementos[q] = elementos[q-1]; // a la siguiente posición
90     elementos[p] = x;
91     n++;
92  }
93  template <typename T>
94  void Lista<T>::eliminar(Lista<T>::posicion p)
95  {
96     assert(p >= 0 && p < n); // posición válida
97     n--;
98     for (Lista<T>::posicion q = p; q < n; q++)
99        //desplazar los eltos. en p+1, p+2, ...
100       elementos[q] = elementos[q+1]; // a la posición anterior
101 }
```

Implementación vectorial pseudoestática

```
103 template <typename T> inline
104 const T& Lista<T>::elemento(Lista<T>::posicion p) const
105 {
106    assert(p >= 0 && p < n); // posición válida
107    return elementos[p];
108 }

110 template <typename T>
111 inline T& Lista<T>::elemento(Lista<T>::posicion p)
112 {
113    assert(p >= 0 && p < n); // posición válida
114    return elementos[p];
115 }
```

Implementación vectorial pseudoestática

```
117 template <typename T>
118 typename Lista<T>::posicion Lista<T>::buscar(const T& x)
        const
119 {
120    Lista<T>::posicion q = 0;
121    bool encontrado = false;
122    while (q < n && !encontrado)
123       if (elementos[q] == x)
124          encontrado = true;
125       else q++;
126    return q;
127 }

129 template <typename T>
130 inline typename Lista<T>::posicion Lista<T>::siguiente(Lista<
        T>::posicion p) const
131 {
132    assert(p >= 0 && p < n); // posición válida
133    return p+1;
134 }
```

Implementación vectorial pseudoestática

```
136 template <typename T>
137 inline typename Lista<T>::posicion Lista<T>::anterior(Lista<T
        >::posicion p) const
138 {
139    assert(p > 0 && p <= n); // posición válida
140    return p-1;
141 }
142 template <typename T>
143 inline typename Lista<T>::posicion Lista<T>::primera() const
144 { return 0; }

146 template <typename T>
147 inline typename Lista<T>::posicion Lista<T>::fin() const
148 { return n; }

150 template <typename T>
151 inline Lista<T>::~Lista()
152 { delete[] elementos; }

154 #endif // LISTA_VEC_H
```