Tad Pila

Implementación vectorial estática (pilavec0.h)

```
1  #ifndef PILA_VEC0_H
2  #define PILA_VEC0_H

4  class Pila {
5  public:
6     typedef int tElemento; // por ejemplo
7     Pila();
8     bool vacia() const;
9     bool llena() const; // Requerida por la implementación
10    const tElemento& tope() const;
11    void pop();
12    void push(const tElemento& x);
13 private:
14    static const int Lmax = 100; // Longitud máxima de una pila
15    tElemento elementos[Lmax]; // vector de elementos
16    int tope_; // posición del tope
17 };

19 #endif // PILA_VEC0_H
```

Implementación vectorial pseudoestática (pilavec1.h)

```
1  #ifndef PILA_VEC1_H
2  #define PILA_VEC1_H
3  class Pila {
4  public:
5     typedef int tElemento; // por ejemplo
6     explicit Pila(unsigned TamaMax); // constructor
7     Pila(const Pila& P); // ctor. de copia
8     Pila& operator =(const Pila& P); // asignación entre pilas
9     bool vacia() const;
10    bool llena() const; // Requerida por la implementación
11    const tElemento& tope() const;
12    void pop();
13    void push(const tElemento& x);
14    ~Pila(); // destructor
15 private:
16    tElemento *elementos; // vector de elementos
17    int Lmax; // tamaño del vector
18    int tope_; // posición del tope
19 };
20 #endif // PILA_VEC1_H
```

Implementación genérica mediante celdas enlazadas

```
1  #ifndef PILA_ENLA_H
2  #define PILA_ENLA_H
3  #include <cassert>

5  template <typename T>
6  class Pila {
7  public:
8     Pila(); // constructor
9     Pila(const Pila<T>& P); // ctor. de copia
10    Pila<T>& operator =(const Pila<T>& P); // asignación
11    bool vacia() const;
12    const T& tope() const;
13    void pop();
14    void push(const T& x);
15    ~Pila(); // destructor
```

Implementación genérica mediante celdas enlazadas

```
16 private:
17    struct nodo {
18       T elto;
19       nodo* sig;
20       nodo(const T& e, nodo* p = nullptr): elto(e), sig(p) {}
21    };

23    nodo* tope_;

25    void copiar(const Pila<T>& P);
26 };
```

Tad Cola

Implementación vectorial pseudoestática

```
1  #ifndef COLA_VEC_H
2  #define COLA_VEC_H
3  #include <cassert>

5  template <typename T> class Cola {
6  public:
7     explicit Cola(size_t TamaMax); // Constructor, req. ctor. T()
8     Cola(const Cola<T>& C); // Ctor. de copia, requiere ctor. T()
9     Cola<T>& operator =(const Cola<T>& C); // Asig., req. ctor. T()
10    bool vacia() const;
11    bool llena() const; // Requerida por la implementación
12    const T& frente() const;
13    void pop();
14    void push(const T& x);
15    ~Cola(); // Destructor
16 private:
17    T *elementos; // Vector de elementos
18    size_t Lmax; // Tamaño del vector
19    size_t n_eltos; // Tamaño de la cola
20 };
```

Implementación vectorial circular

```
1  #ifndef COLA_CIR_H
2  #define COLA_CIR_H
3  #include <cassert>

5  template <typename T> class Cola {
6  public:
7     explicit Cola(size_t TamaMax); // Constructor, req. ctor. T()
8     Cola(const Cola<T>& C); // Ctor. de copia, requiere ctor. T()
9     Cola<T>& operator =(const Cola<T>& C); // Asig., req. ctor. T()
10    bool vacia() const;
11    bool llena() const; // Requerida por la implementación
12    const T& frente() const;
13    void pop();
14    void push(const T& x);
15    ~Cola(); // Destructor
16 private:
17    T *elementos; // Vector de elementos
18    size_t Lmax; // Tamaño del vector
19    size_t inicio, fin; // Posiciones de los extremos de la cola
20 };
```

Implementación mediante una estructura enlazada

```
1  #ifndef COLA_ENLA_H
2  #define COLA_ENLA_H
3  #include <cassert>

5  template <typename T> class Cola {
6  public:
7     Cola(); // Constructor
8     Cola(const Cola<T>& C); // Ctor. de copia
9     Cola<T>& operator =(const Cola<T>& C); // Asignación de colas
10    bool vacia() const;
11    const T& frente() const;
12    void pop();
13    void push(const T& x);
14    ~Cola(); // Destructor
```

```
15 private:
16    struct nodo {
17       T elto;
18       nodo* sig;
19       nodo(const T& e, nodo* p = nullptr): elto(e), sig(p) {}
20    };

22    nodo *inicio, *fin; // Extremos de la cola

24    void copiar(const Cola<T>& C);
25 };
```

Tad Lista

Implementación vectorial pseudoestática

```
16 template <typename T>
17 class Lista {
18 public:
19    typedef size_t posicion; // Posición de un elemento
20    explicit Lista(size_t TamaMax); // Constructor, req. ctor. T()
21    Lista(const Lista<T>& L); // Ctor. de copia, requiere ctor. T()
22    Lista<T>& operator =(const Lista<T>& L);// Asig, req. ctor. T()
23    void insertar(const T& x, posicion p);
24    void eliminar(posicion p);
25    const T& elemento(posicion p) const;// Lec. elto. en Lista const
26    T& elemento(posicion p); // Lec/Esc elto. en Lista no-const
27    posicion buscar(const T& x) const;// Req. operador == para 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
```

```
33 private:
34    T *elementos; // Vector de elementos
35    size_t Lmax; // Tamaño del vector
36    size_t n; // Longitud de la lista
37 };
```

Implementación mediante una estructura enlazada

```
1  template <typename T> class Lista {
2     struct nodo; // Declaración adelantada privada
3  public:
4     typedef nodo* posicion; // Posición de un elemento
5     Lista(); // Constructor
6     void insertar(const T& x, posicion& p);
7     void eliminar(posicion& p);
8     // ......
9  private:
10    struct nodo {
11       T elto;
12       nodo* sig;
13       nodo(T e, nodo* p = nullptr): elto(e), sig(p) {}
14    };

16    nodo* L; // lista enlazada de nodos
17 };
```

Implementación con una estructura enlazada con cabecera

```
1  #ifndef LISTA_ENLA_H
2  #define LISTA_ENLA_H
3  #include <cassert>

5  template <typename T> class Lista {
6     struct nodo; // Declaración adelantada privada
7  public:
8     typedef nodo* posicion; // Posición de un elemento
9     Lista(); // Constructor, requiere ctor. T()
10    Lista(const Lista<T>& Lis); // Ctor. de copia, requiere ctor. T()
11    Lista<T>& operator =(const Lista<T>& Lis); // Asig. de listas
12    void insertar(const T& x, posicion p);
13    void eliminar(posicion p);
14    const T& elemento(posicion p) const;// Lec. elto. en Lista const
15    T& elemento(posicion p); // Lec/Esc elto. en Lista no-const
```

```
16    posicion buscar(const T& x) const;// Req. operador == para T
17    posicion siguiente(posicion p) const;
18    posicion anterior(posicion p) const;
19    posicion primera() const;
20    posicion fin() const; // Posición después del último
21    ~Lista(); // Destructor
22 private:
23    struct nodo {
24       T elto;
25       nodo* sig;
26       nodo(const T& e, nodo* p = nullptr): elto(e), sig(p) {}
27    };

29    nodo* L; // lista enlazada de nodos

31    void copiar(const Lista<T>& Lis);
32 };
```

Impl. con una estructura doblemente enlazada con cabecera

```
1  #ifndef LISTA_DOBLE_H
2  #define LISTA_DOBLE_H
3  #include <cassert>

5  template <typename T> class Lista {
6     struct nodo; // Declaración adelantada privada
7  public:
8     typedef nodo* posicion; // Posición de un elemento
9     Lista(); // Constructor, requiere ctor. T()
10    Lista(const Lista<T>& Lis); // Ctor. de copia, req. ctor. T()
11    Lista<T>& operator =(const Lista<T>& Lis); // Asig. de listas
12    void insertar(const T& x, posicion p);
13    void eliminar(posicion p);
14    const T& elemento(posicion p) const;// Lec. elto. en Lista const
15    T& elemento(posicion p); // Lec/Esc elto. en Lista no-const
```

```
16    posicion buscar(const T& x) const;// Req. operador == para T
17    posicion siguiente(posicion p) const;
18    posicion anterior(posicion p) const;
19    posicion primera() const;
20    posicion fin() const; // Posición después del último
21    ~Lista(); // Destructor
22 private:
23    struct nodo {
24       T elto;
25       nodo *ant, *sig;
26       nodo(const T& e, nodo* a = nullptr, nodo* s = nullptr):
27          elto(e), ant(a), sig(s) {}
28    };

30    nodo* L; // lista doblemente enlazada de nodos

32    void copiar(const Lista<T>& Lis);
33 };
```