Análisis de Algoritmos y Estructuras de Datos

Tema 6: Tipo Abstracto de Datos Cola

Mª Teresa García Horcajadas José Fidel Argudo Argudo
Antonio García Domínguez Francisco Palomo Lozano

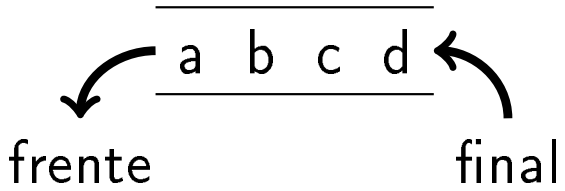
UCA
Universidad de Cádiz

Versión 3.0

Índice

Definición de Cola

- Una cola es una secuencia de elementos en la que las operaciones se realizan por los extremos:
  - Las eliminaciones se realizan por el extremo llamado inicio, frente o principio de la cola.
  - Los nuevos elementos son añadidos por el otro extremo, llamado fondo o final de la cola.
- El primer elemento añadido a una cola es el primero en salir de ella, por lo que también es conocida como estructura FIFO: *First Input First Output*.



Especificación del TAD *Cola*

Definición:

Una cola es una secuencia de elementos de un mismo tipo T, en la cual se pueden añadir elementos sólo por un extremo, al que llamaremos fin, y eliminar por el otro, al que llamaremos inicio o frente.

Operaciones:

Cola()

Postcondiciones: Crea una cola vacía.

bool vacia() const

Postcondiciones: Devuelve `true` si la cola está vacía.

size_t tama() const

Postcondiciones: Devuelve el número de elementos que contiene la cola.

Especificación del TAD *Cola*

const T& frente() const

Precondiciones: La cola no está vacía.

Postcondiciones: Devuelve el elemento del inicio de la cola.

void pop()

Precondiciones: La cola no está vacía.

Postcondiciones: Elimina el elemento del inicio de la cola y el siguiente, si existe, se convierte en el nuevo inicio.

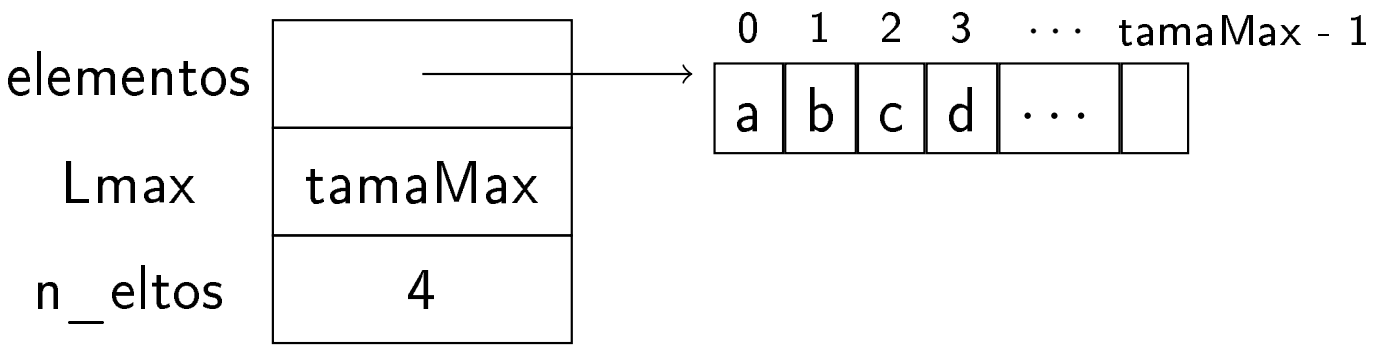
void push(const T& x)

Postcondiciones: Inserta el elemento x al final de la cola, y si estaba vacía, éste es el nuevo inicio.

Implementación vectorial pseudoestática

Implementación vectorial pseudoestática

Capacidad de la cola definida por el usuario del TAD mediante un parámetro del constructor.



Implementación vectorial pseudoestática (colavec0.h)

```
1  template <typename T> class Cola {
2  public:
3    // ...
4  private:
5    T* elementos;      // Vector de elementos
6    size_t Lmax,       // Tamaño del vector
7           n_eltos;    // Tamaño de la cola
8  };

10 template <typename T>
11 inline Cola<T>::Cola(size_t tamaMax) :
12    elementos(new T[tamaMax]),
13    Lmax(tamaMax),
14    n_eltos(0)
15 {}

17 template <typename T>
18 inline bool Cola<T>::vacia() const
19 {
20    return n_eltos == 0;
21 }
```

Implementación vectorial pseudoestática (colavec0.h)

```
23 template <typename T>
24 inline size_t Cola<T>::tama() const
25 {
26    return n_eltos;
27 }

29 template <typename T>
30 inline size_t Cola<T>::tamaMax() const
31 {
32    return Lmax;
33 }

35 template <typename T>
36 inline const T& Cola<T>::frente() const
37 {
38    assert(!vacia());
39    return elementos[0];
40 }
```

Implementación vectorial pseudoestática (colavec0.h)

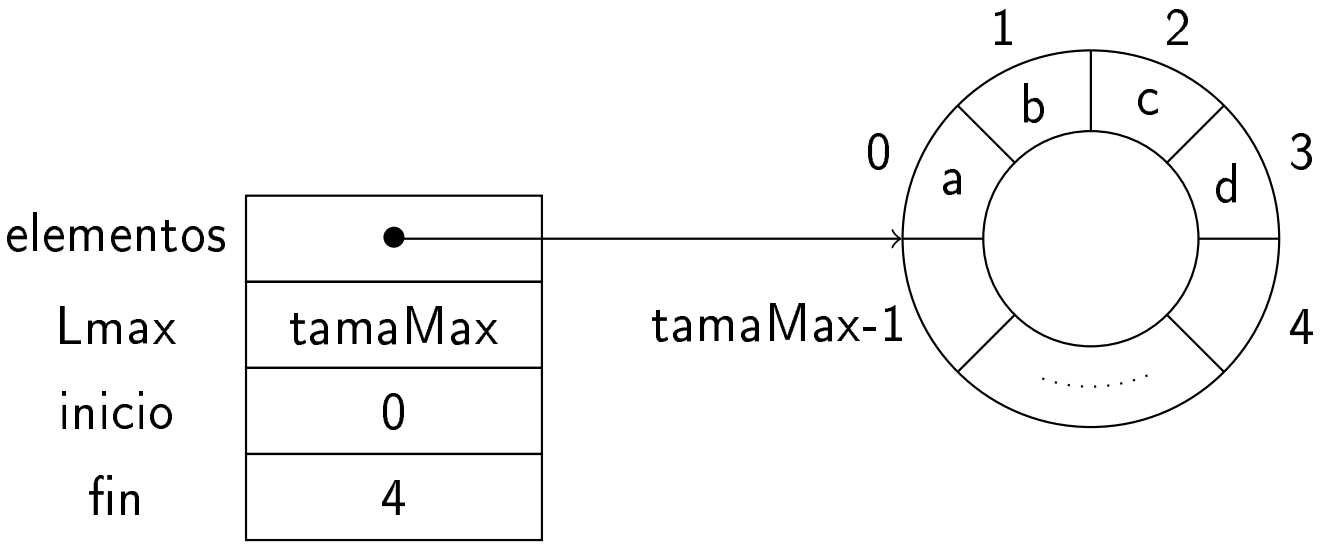
```
42 template <typename T>
43 void Cola<T>::pop()
44 {
45    assert(!vacia());
46    for (size_t i = 1; i < n_eltos; ++i)
47       elementos[i-1] = elementos[i];
48    --n_eltos;
49 }

51 template <typename T>
52 inline void Cola<T>::push(const T& x)
53 {
54    assert(n_eltos < Lmax);
55    elementos[n_eltos] = x;
56    ++n_eltos;
57 }
```

Implementación vectorial circular: esquema general

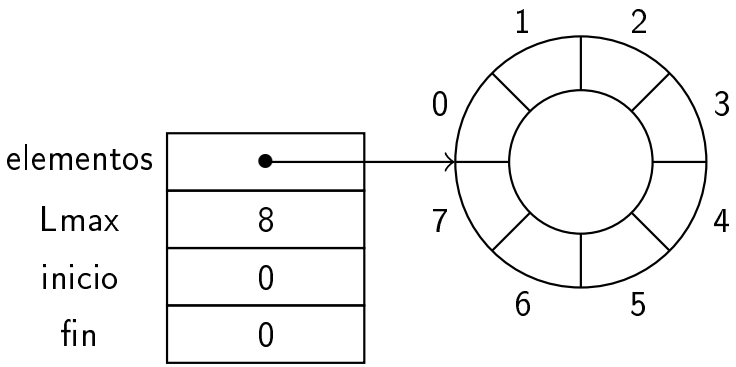
Modificación de la representación del TAD Cola

La eficiencia de la extracción, $\Theta(n)$, se puede mejorar haciendo que el frente no permanezca en una posición fija, sino que avance con cada extracción, al igual que el final con cada inserción. Sin embargo, para que esta idea funcione se debe considerar que el vector es circular desde un punto de vista lógico.

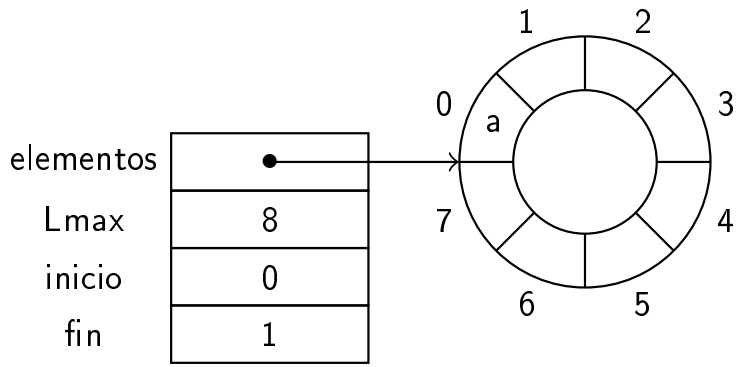


Implementación vectorial circular

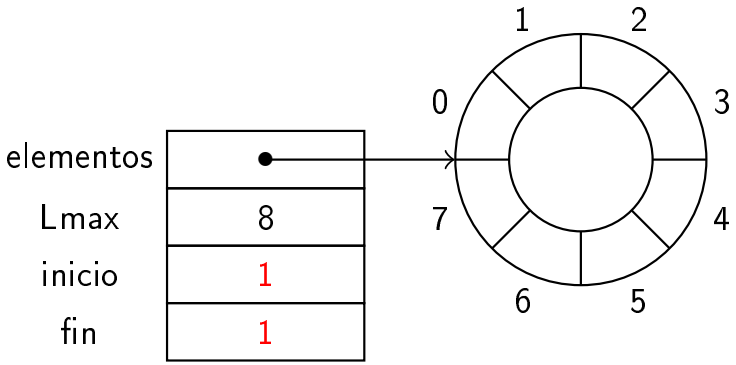
Indistinguibilidad cola vacía/llena



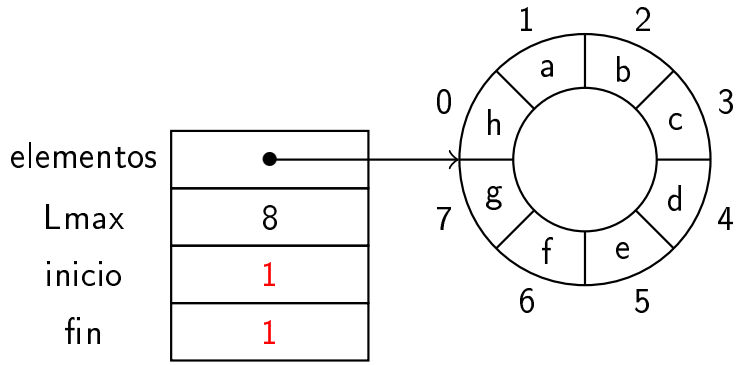
(a) Cola inicial (vacía)



(b) Inserción de un elemento



(c) Extracción (vacía)

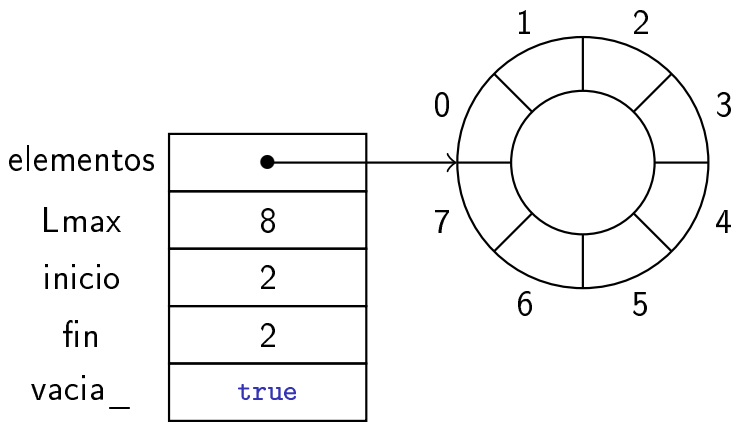


(d) Inserción de *Lmax* eltos. (llena)

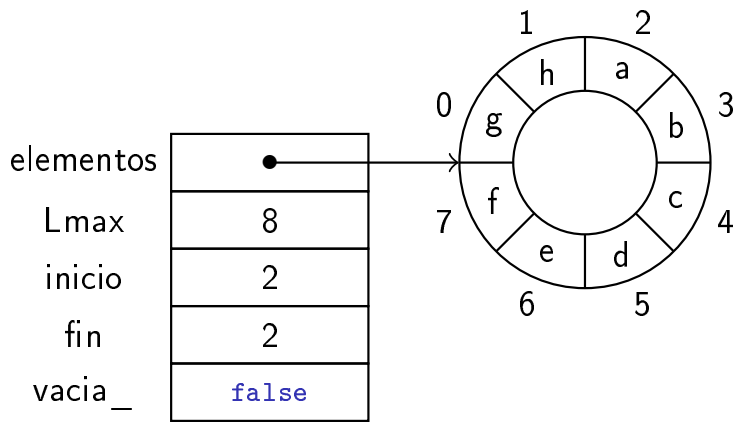
Implementación vectorial circular

Distinción cola vacía/llena

1. Atributo adicional en la clase cola de tipo `bool`, que indique vacía/no-vacía o llena/no-llena.



(e) Vacía



(f) Llena

Implementación vectorial circular (colavec1.h)

```
1  template <typename T> class Cola {
2  public:
3     // ...
4  private:
5     T* elementos;      // Vector de elementos
6     size_t Lmax,       // Tamaño del vector
7            inicio,     // Posición del frente
8            fin;        // Posición siguiente a la del último
9     bool vacia_;       // Estado de la pila
10 };

12 template <typename T>
13 inline Cola<T>::Cola(size_t tamaMax) :
14    elementos(new T[tamaMax]),
15    Lmax(tamaMax),
16    inicio(0),
17    fin(0),
18    vacia_(true)
19 {}
```

Implementación vectorial circular (colavec1.h)

```
21 template <typename T>
22 inline bool Cola<T>::vacia() const
23 {
24    return vacia_;
25 }

27 template <typename T>
28 inline size_t Cola<T>::tama() const
29 {
30    if (inicio == fin && !vacia_) // Cola llena
31       return Lmax;
32    // Operando con signo:
33    //    return (fin - inicio) % Lmax;
34    // Operando sin signo hay que evitar desbordamientos:
35    else if (inicio <= fin)
36       return fin - inicio; // nº celdas ocupadas
37    else
38       return Lmax - (inicio - fin); // Lmax - nº celdas libres
39 }
```

Implementación vectorial circular (colavec1.h)

```
41 template <typename T>
42 inline size_t Cola<T>::tamaMax() const
43 {
44     return Lmax;
45 }

47 template <typename T>
48 inline const T& Cola<T>::frente() const
49 {
50     assert(!vacia_);
51     return elementos[inicio];
52 }
```

Implementación vectorial circular (colavec1.h)

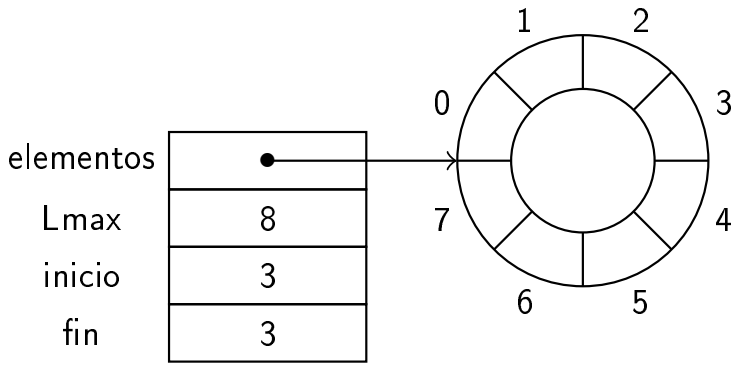
```
54 template <typename T>
55 inline void Cola<T>::pop()
56 {
57     assert(!vacia_);
58     inicio = (inicio + 1) % Lmax;
59     if (inicio == fin) vacia_ = true;
60 }

62 template <typename T>
63 inline void Cola<T>::push(const T& x)
64 {
65     assert(inicio != fin || vacia_);    // Cola no llena
66     if (vacia_) vacia_ = false;
67     elementos[fin] = x;
68     fin = (fin + 1) % Lmax;
69 }
```

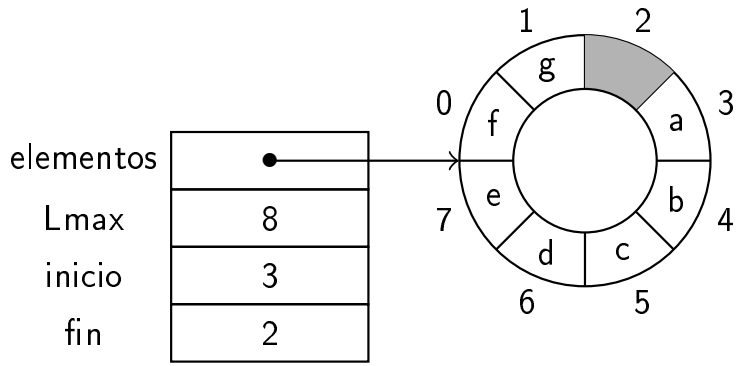
Implementación vectorial circular

Distinción cola vacía/llena

2. Conservar al menos una posición libre en el vector.



(g) Vacía



(h) Llena

Implementación vectorial circular (colavec2.h)

```
1  template <typename T> class Cola {
2  public:
3     // ...
4  private:
5     T* elementos;      // Vector de elementos
6     size_t Lmax,       // Tamaño del vector
7            inicio,     // Posición del frente
8            fin;        // Posición siguiente a la del último
9    };

11 template <typename T>
12 inline Cola<T>::Cola(size_t tamaMax) :
13    elementos(new T[tamaMax + 1]), // +1 para detectar cola llena
14    Lmax(tamaMax + 1),
15    inicio(0),
16    fin(0)
17 {}
```

Implementación vectorial circular (colavec2.h)

```
19 template <typename T>
20 inline bool Cola<T>::vacia() const
21 {
22    return inicio == fin;
23 }

25 template <typename T>
26 inline size_t Cola<T>::tama() const
27 {
28    // Operando con signo:
29    //    return (fin - inicio) % Lmax;
30    // Operando sin signo hay que evitar desbordamientos:
31    if (inicio <= fin)
32       return fin - inicio; // nº celdas ocupadas
33    else
34       return Lmax - (inicio - fin); // Lmax - nº celdas libres
35 }
```

Implementación vectorial circular (colavec2.h)

```
37 template <typename T>
38 inline size_t Cola<T>::tamaMax() const
39 {
40    return Lmax - 1;
41 }

43 template <typename T>
44 inline const T& Cola<T>::frente() const
45 {
46    assert(!vacia());
47    return elementos[inicio];
48 }
```

Implementación vectorial circular (colavec2.h)

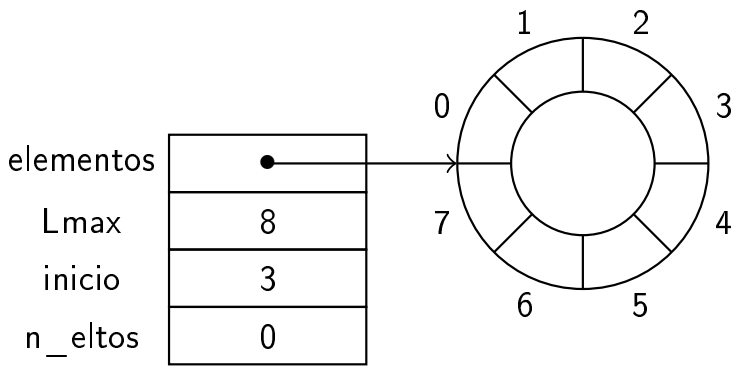
```
50 template <typename T>
51 inline void Cola<T>::pop()
52 {
53     assert(!vacia());
54     inicio = (inicio + 1) % Lmax;
55 }

57 template <typename T>
58 inline void Cola<T>::push(const T& x)
59 {
60     assert(inicio != (fin + 1) % Lmax);    // Cola no llena
61     elementos[fin] = x;
62     fin = (fin + 1) % Lmax;
63 }
```

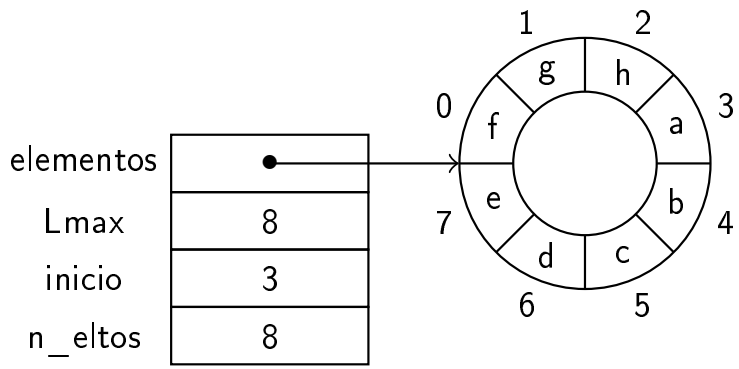
Implementación vectorial circular

Distinción cola vacía/llena

3. Sustituir la posición posterior a la del último, `fin`, por un contador de elementos.



(i) Vacía



(j) Llena

Implementación vectorial circular (colavec.h)

```
1  #ifndef COLA_VEC_H
2  #define COLA_VEC_H
3  #include <cstddef>    // size_t
4  #include <cassert>

6  template <typename T> class Cola {
7  public:
8     explicit Cola(size_t tamaMax);    // Constructor, req. ctor. T()
9     bool vacia() const;
10    size_t tama() const;
11    size_t tamaMax() const;           // Requerida por la implementación
12    const T& frente() const;
13    void pop();
14    void push(const T& x);
15    Cola(const Cola& C);              // Ctor. de copia, req. ctor. T()
16    Cola& operator =(const Cola& C); // Asig., req. ctor. T()
17    ~Cola();                          // Destructor
18 private:
19    T* elementos;      // Vector de elementos
20    size_t Lmax,       // Tamaño del vector
21           inicio,     // Posición del frente
22           n_eltos;    // Tamaño de la cola
23 };
```

Implementación vectorial circular (colavec.h)

```
25 template <typename T>
26 inline Cola<T>::Cola(size_t tamaMax) :
27    elementos(new T[tamaMax]),
28    Lmax(tamaMax),
29    inicio(0),
30    n_eltos(0)
31 {}

33 template <typename T>
34 inline bool Cola<T>::vacia() const
35 {
36    return n_eltos == 0;
37 }

39 template <typename T>
40 inline size_t Cola<T>::tama() const
41 {
42    return n_eltos;
43 }
```

Implementación vectorial circular (colavec.h)

```
45 template <typename T>
46 inline size_t Cola<T>::tamaMax() const
47 {
48    return Lmax;
49 }

51 template <typename T>
52 inline const T& Cola<T>::frente() const
53 {
54    assert(!vacia());
55    return elementos[inicio];
56 }

58 template <typename T>
59 inline void Cola<T>::pop()
60 {
61    assert(!vacia());
62    inicio = (inicio + 1) % Lmax;
63    --n_eltos;
64 }
```

Implementación vectorial circular (colavec.h)

```
66 template <typename T>
67 inline void Cola<T>::push(const T& x)
68 {
69    assert(n_eltos < Lmax);
70    elementos[(inicio + n_eltos) % Lmax] = x;
71    ++n_eltos;
72 }

74 // Constructor de copia
75 template <typename T>
76 Cola<T>::Cola(const Cola& C) :
77    elementos(new T[C.Lmax]),
78    Lmax(C.Lmax),
79    inicio(0),
80    n_eltos(C.n_eltos)
81 {
82    for (size_t i = 0; i < n_eltos; ++i)    // Copiar elementos
83       elementos[i] = C.elementos[(C.inicio + i) % Lmax];
84 }
```

Implementación vectorial circular (colavec.h)

```
86  // Asignación entre colas
87  template <typename T>
88  Cola<T>& Cola<T>::operator =(const Cola& C)
89  {
90      if (this != &C) {    // Evitar autoasignación
91          // Destruir el vector y crear uno nuevo si es necesario
92          if (Lmax != C.Lmax) {
93              T* p = elementos;
94              elementos = new T[C.Lmax]; // Si new falla, *this no cambia.
95              Lmax = C.Lmax;
96              delete[] p;
97          }
98          inicio = 0;
99          n_eltos = C.n_eltos;
100         for (size_t i = 0; i < n_eltos; ++i)   // Copiar elementos
101             elementos[i] = C.elementos[(C.inicio + i) % Lmax];
102     }
103     return *this;
104 }
```

Implementación vectorial circular (colavec.h)

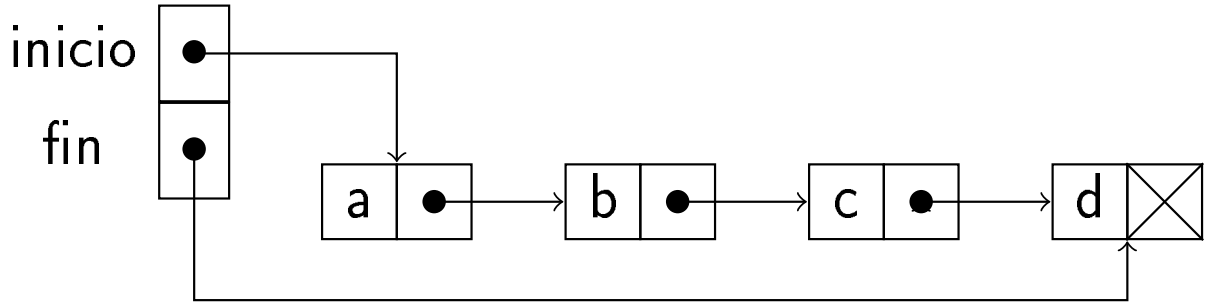
```
106 // Destructor
107 template <typename T>
108 inline Cola<T>::~Cola()
109 {
110    delete[] elementos;
111 }

113 #endif // COLA_VEC_H
```

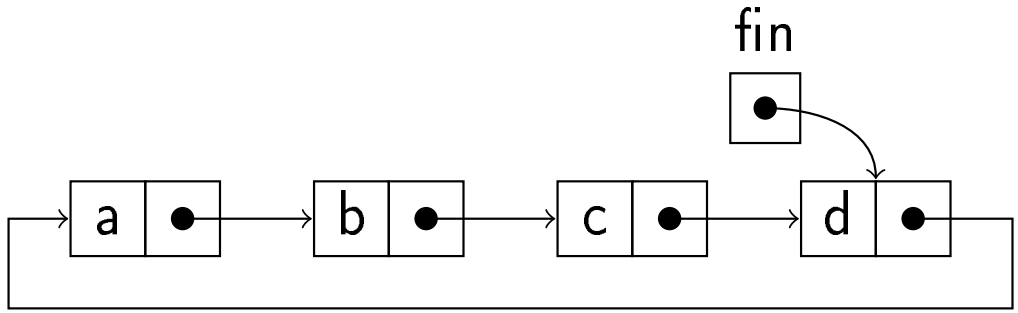
Implementación mediante una estructura enlazada

Estructura dinámica

El tamaño de la estructura de datos varía con el de la cola durante la ejecución del programa, por lo que la capacidad de la cola no está limitada. A cambio se ocupa espacio adicional con los enlaces.



(a) Dos punteros a los extremos



(b) Estructura enlazada circular

Implementación mediante una estructura enlazada (colaenla.h)

```
1  #ifndef COLA_ENLA_H
2  #define COLA_ENLA_H
3  #include <cstddef>    // size_t
4  #include <cassert>

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

Implementación mediante una estructura enlazada (colaenla.h)

```
17  private:
18      struct nodo {
19        T elto;
20        nodo* sig;
21        nodo(const T& e, nodo* p = nullptr) : elto(e), sig(p) {}
22      };

24      nodo *inicio, *fin;     // Extremos de la cola
25      size_t n_eltos;         // Tamaño de la cola

27      void copiar(const Cola& C);
28  };
```

Implementación mediante una estructura enlazada (colaenla.h)

```
30 template <typename T>
31 inline Cola<T>::Cola() : // Solo hacemos que inicio apunte
32    inicio(nullptr),          a nulo —> no hay elementos
33    // fin(nullptr), // Innecesario en esta implementación
34    n_eltos(0)
35 {}

37 template <typename T>
38 inline bool Cola<T>::vacia() const
39 { return inicio == nullptr; } // Alternativa: return n_eltos == 0;

41 template <typename T>
42 inline size_t Cola<T>::tama() const
43 { return n_eltos; }

45 template <typename T>
46 inline const T& Cola<T>::frente() const
47 {
48    assert(!vacia());
49    return inicio->elto;
50 }
```

Implementación mediante una estructura enlazada (colaenla.h)

```
52 template <typename T>
53 inline void Cola<T>::pop()
54 {
55    assert(!vacia());
56    nodo* p = inicio;
57    inicio = p->sig;
58    // if (!inicio) fin = nullptr; // Innecesario en esta implementación
59    delete p;
60    --n_eltos;
61 }

63 template <typename T>
64 inline void Cola<T>::push(const T& x)
65 {
66    if (vacia()) //Si la cola está vacía hay que cambiar inicio e insertar elemento
67       inicio = fin = new nodo(x);
68    else
69       fin = fin->sig = new nodo(x);
70    ++n_eltos;
71 }
```

Implementación mediante una estructura enlazada (colaenla.h)

```
73 // Constructor de copia
74 template <typename T>
75 inline Cola<T>::Cola(const Cola& C) : Cola()
76 {
77    copiar(C);
78 }

80 // Asignación entre colas
81 template <typename T>
82 inline Cola<T>& Cola<T>::operator =(const Cola& C)
83 {
84    if (this != &C) {    // Evitar autoasignación
85       this->~Cola();    // Vaciar la cola actual
86       copiar(C);
87    }
88    return *this;
89 }
```

//Llamamos al Ctor para crear una Cola nueva VACIA

//Llama a un método que copia colas

Implementación mediante una estructura enlazada (colaenla.h)

```
91  // Destructor: vacía la cola
92  template <typename T>
93  Cola<T>::~Cola()
94  {
95     nodo* p;
96     while (inicio) {
97        p = inicio->sig;
98        delete inicio;
99        inicio = p;
100    }
101    // fin = nullptr; // Innecesario en esta implementación
102    n_eltos = 0;
103 }
```

//Por defecto: Destruye 1x1 inicio, fin y contador

//Creado por nosotros: Liberar memoria que ocupa la cadena de Nodos

Implementación mediante una estructura enlazada (colaenla.h)

```
105 // Método privado
106 template <typename T>
107 void Cola<T>::copiar(const Cola& C)
108 // Pre: *this está vacía.
109 // Post: *this es copia de C.
110 {
111    if (!C.vacia()) {
112        // Copiar el primer elto.
113       inicio = fin = new nodo(C.inicio->elto);
114       n_eltos = 1;
115       // Copiar el resto de elementos hasta el final de C.
116       for (nodo* p = C.inicio->sig; p; p = p->sig, ++n_eltos)
117          fin = fin->sig = new nodo(p->elto);
118    }    // p método privado de la Cola, donde p = puntero nodo, por tanto el
                                        bucle para cuando es nulo.
119 }

121 #endif // COLA_ENLA_H
```