TEMA 6: TIPO DE DATOS ABSTRACTO *COLA*

ALGORITMOS Y ESTRUCTURAS DE DATOS

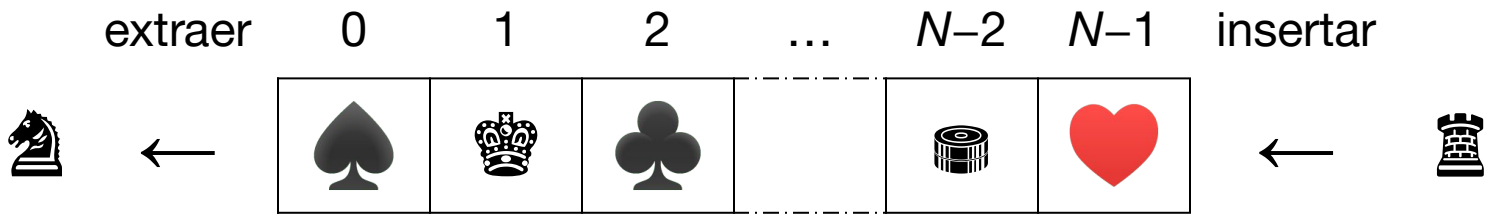
M. Colebrook Santamaría
J. Riera Ledesma
J. Hernández Aceituno

Objetivos

- Especificación formal del TDA cola.
- Implementación del TDA cola mediante estructuras estáticas y objetos dinámicos.
- Operaciones sobre colas:
  - Inserción (*push*)
  - Extracción (*pop*)
  - Lectura del frente de la cola (*front*)
  - Lectura del final de la cola (*back*)

Especificación formal del TDA cola

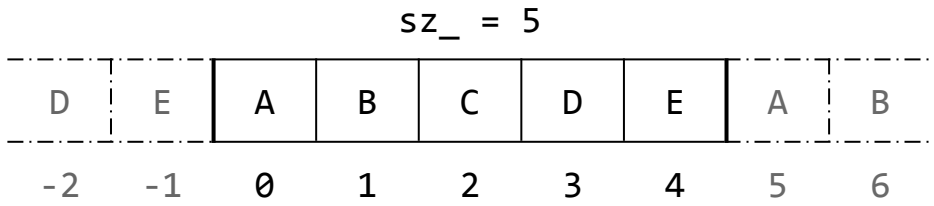
- Una **cola** (***queue*** en inglés) es una estructura de datos con un modo de acceso a sus elementos de tipo **FIFO** (*First In First Out*: primero en entrar, primero en salir).
- El comportamiento de este tipo de estructura de datos es análogo al de una cola en el supermercado, en la taquilla del cine, etc.



Implementación del TDA cola mediante estructuras estáticas (1)

- La clase `round_vector_t<T>` implementa un vector circular, en el que la posición de un elemento se calcula con respecto al **módulo (%) de su tamaño**. Esto permite un tamaño virtualmente "infinito".

```
template <class T>
class round_vector_t {

private:
  T*    v_;
  int   sz_;

public:
  ...

  const T& at(const int pos) const { return v_[pos % sz_]; }
  T& at(const int pos)             { return v_[pos % sz_]; }
};
```




```
at(4)  -> v_[4  % 5] = v_[4] = E
at(5)  -> v_[5  % 5] = v_[0] = A
at(97) -> v_[97 % 5] = v_[2] = C
```

Implementación del TDA cola mediante estructuras estáticas (2)

- Para la versión con estructuras **estáticas** usaremos un objeto de tipo `round_vector_t<T>`:

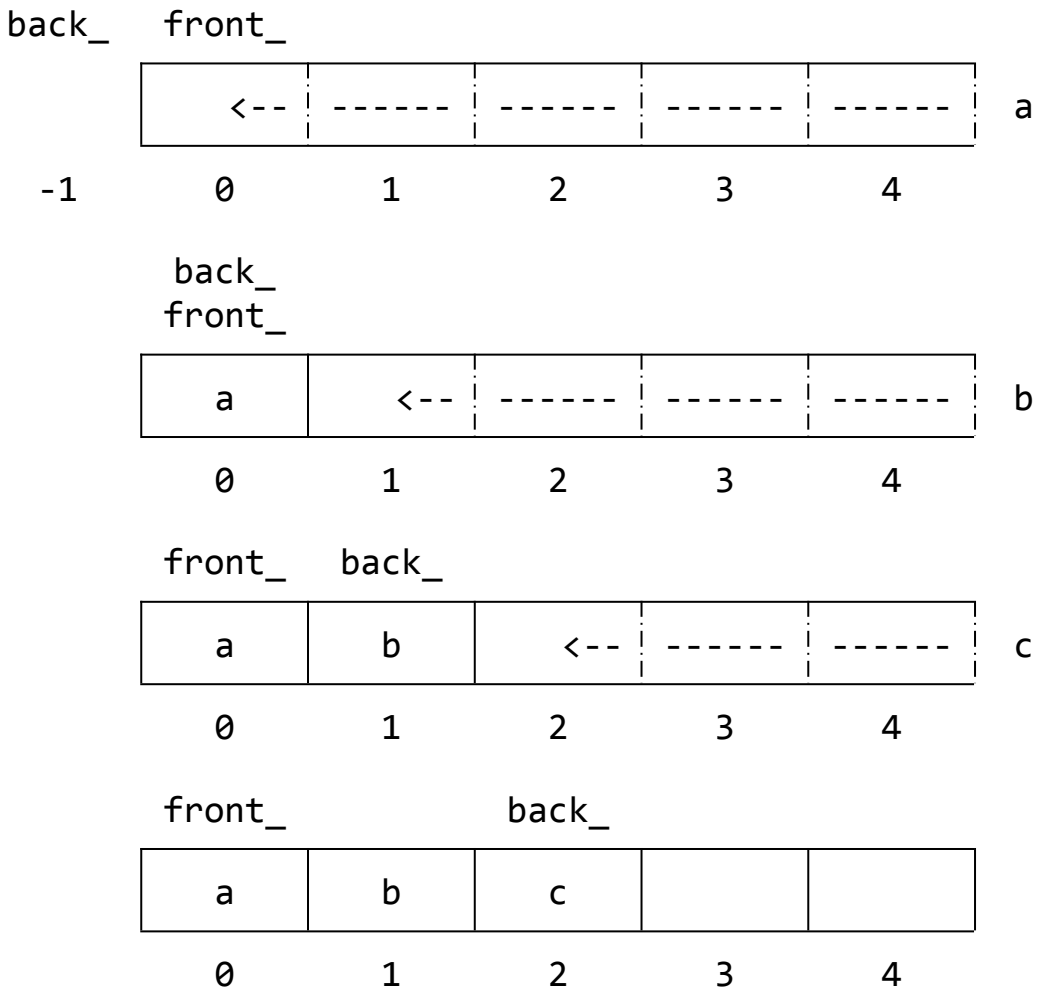
```
template <class T>
class queue_v_t
{
private:
  round_vector_t<T> v_;
  int front_;  // Índice del primer elemento
  int back_;   // Índice del último elemento

public:
  ...
};
```

Implementación del TAD cola mediante estructuras estáticas (3)

- Los métodos iniciales son:

```
// constructor y destructor
queue_v_t(const int max_sz = MAX_VECTOR_SIZE):
  v_(max_sz), front_(0), back_(-1) {}

~queue_v_t(void) {}

// cola vacía
bool empty(void) const { return back_ < front_; }

// cola llena
bool full(void) const { return size() == v_.get_size(); }

// tamaño de la cola
int size(void) const { return back_ - front_ + 1; }
```

Inserción (*push*)

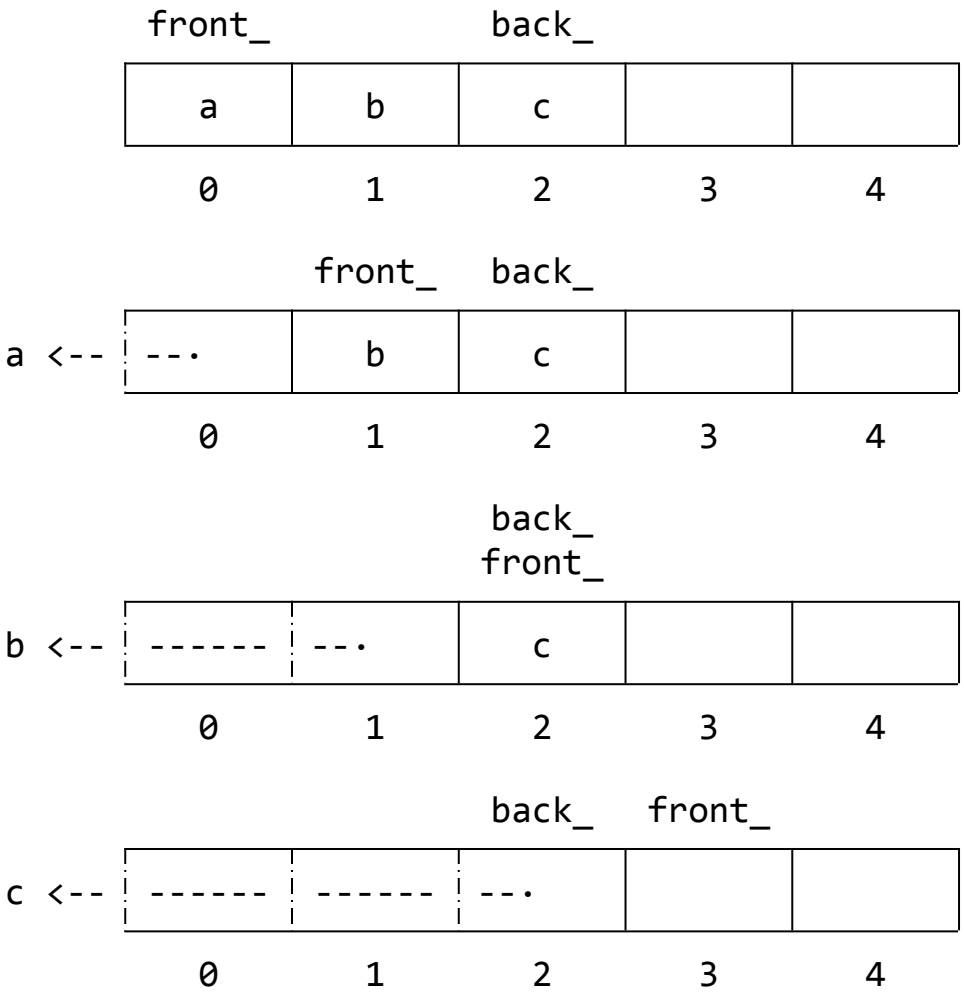



```
void push(const T& dato) {
  assert(!full());
  back_++;
  v_.at(back_) = dato;
}
```

que equivale a

```
void push(const T& dato) {
  assert(!full());
  v_.at(++back_) = dato;
}
```

Extracción (*pop*)



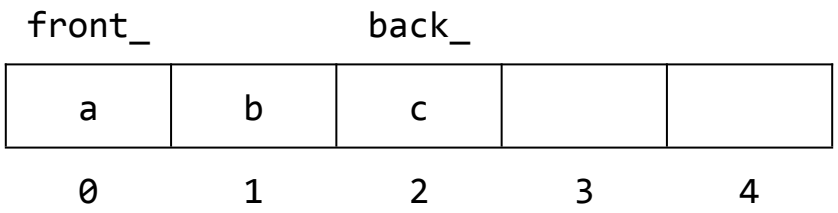

```
void pop(void) {
  assert(!empty());
  front_++;
}
```

o bien

```
T& pop(void) {
  assert(!empty());
  return v_.at(front_++);
}
```

Lectura del *front*

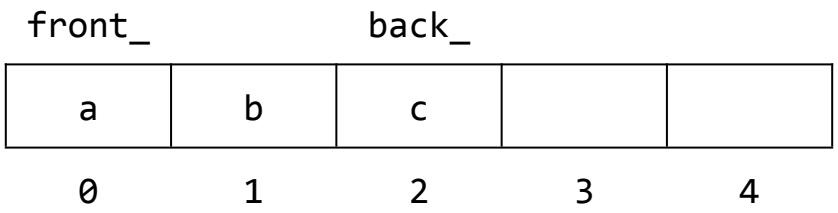
- Se puede realizar una lectura del dato que está justo en el `front` de la cola sin tener que extraerlo.




```
const T& front(void) const {
  assert(!empty());
  return v_.at(front_);
}
```

Lectura del *back*

- Asimismo, se puede realizar una lectura del dato que está justo en el `back` de la cola sin tener que extraerlo.




```
const T& back(void) const {
  assert(!empty());
  return v_.at(back_);
}
```

Implementación mediante objetos dinámicos

- Para el caso de la implementación de un TAD cola con objetos dinámicos, usaremos una lista doblemente enlazada `dll_t<T>`:

```
template <class T>
class queue_l_t {
private:
  dll_t<T>    l_;
public:
  queue_l_t(void): l_() {}  // constructor
  ~queue_l_t(void) {}       // destructor
  bool empty(void) const { return l_.empty(); }
  int size(void) const { return l_.get_size(); }

  ...
};
```

Inserción y extracción

```
void push(const T& dato) {
  // creamos nodo con el dato
  dll_node_t<T>* node = new dll_node_t<T>(dato);
  assert(node != NULL);
  l_.push_front(node); // lo insertamos
}

void pop(void) {
  assert(!empty());
  // extraemos nodo del final
  dll_node_t<T>* node = l_.pop_back();
  delete node; // lo eliminamos
  // delete l_.pop_back();
}
```

Lectura del *front* y del *back*

```
const T& front(void) const {
  assert(!empty());
  // accedemos al último nodo
  dll_node_t<T>* node = l_.get_tail();
  return node->get_data();
  // return l_.get_tail()->get_data();
}
const T& back(void) const {
  assert(!empty());
  // accedemos al primer nodo
  dll_node_t<T>* node = l_.get_head();
  return node->get_data();
  // return l_.get_head()->get_data();
}
```

Referencias

- Olsson, M. (2018), “C++ 17 Quick Syntax Reference”, Apress. Disponible en PDF en la BBTK-ULL: https://doi.org/10.1007/978-1-4842-3600-0
- Stroustrup, B. (2002), “El Lenguaje de Programación C++”, Addison Wesley.
- C++ Syntax Highlighting (código en colores): tohtml.com/cpp