Estructuras de Datos

EEDD - GRADO EN ING. INFORMATICA - UCO

Estructuras lineales:
Lista simple, pilas y colas

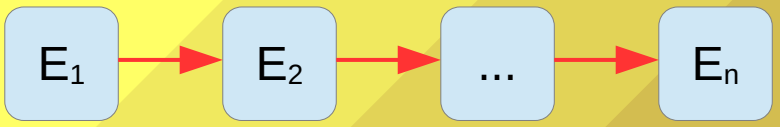
fjmadrid@uco.es

Contenidos

- Características de la estructuras lineales.
- Lista Simple.
- Pilas.
- Colas.

Introducción

- Estructuras lineales.
  - Contenedores de datos genéricos.
  - Relación 1-1: cada elemento tiene un predecesor y un sucesor (salvo el inicial y el final).
  - Indicadas cuando se realiza un proceso secuencial de los datos.



Lista Simple

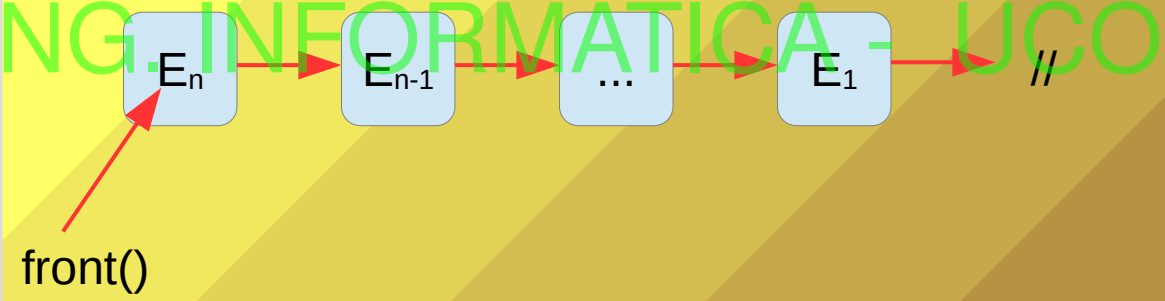
- El TAD SList[T].

SList[T]
Makers:
- **create():SList[T]** //makes an empty list.
  - post-c: isEmpty() is True.

Observers:
- **isEmpty():Boolean** //is the list empty?
- **size():Integer** //Number of items in the list.
- **front():T** //return the first Item of the list.
  - pre-c: not isEmpty()

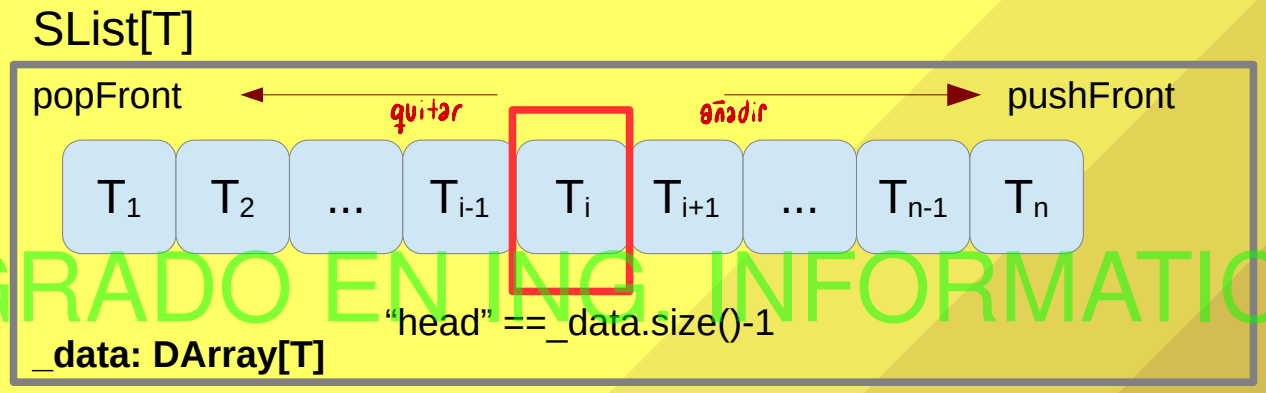
Modifiers:
- **pushFront(item:T)** //insert item before the head.
  - post-c: front() == item
  - post-c: size()==old.size()+1
- **popFront()** //delete the first item of the list.
  - pre-c: not isEmpty()
  - post-c: size()==old.size()-1

Invariants:
- isEmpty() or size()>0



Lista Simple

- SList[T]: diseño usando DArray[T].

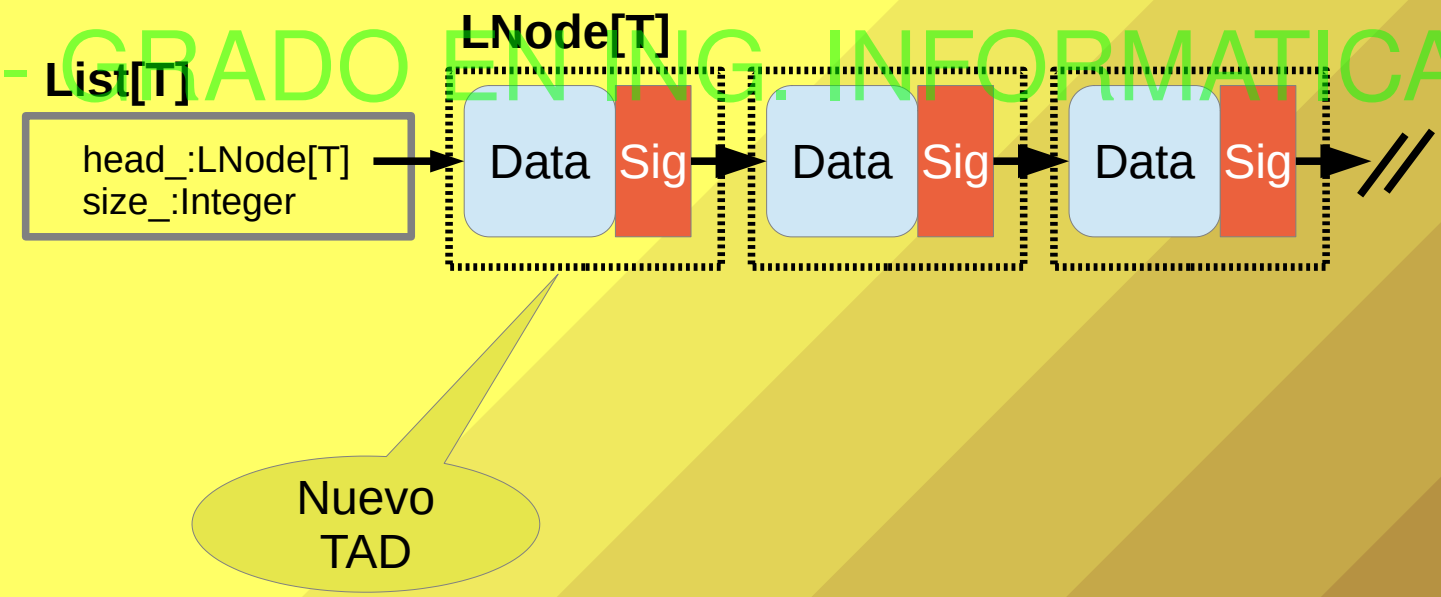



```
SList::create()
    _data ← DArray()

SList::isEmpty():Boolean
    return _data.size()==0

SList::size():Integer
    return _data.size()

SList::front():T //O(1)
    return _data.get(_data.size()-1)
```

```
SList::pushFront(newItem:T) //O(n)CA(1)
   _data.pushBack(newItem)


SList::popFront() //O(1)
    _data.popBack()
```

Lista Simple

- SList[T]: diseño con lista de nodos enlazados.



Lista Simple

- LNode[T]: Nodo simple.

TAD LNode[T]
Makers:
- create(item:T, n:LNode[T]):LNode[T]
  - Post-c: item()=item
  - Post-c: next()==n

Observers:
- next():LNode[T] //Gets next node.
- item():T //Gets the stored data.

Modifiers:
- setNext(n:LNode[T]) //Sets the link to next node.
  - Post-c: next()==n
- setItem(item:T) //Sets the stored data.
  - Post-c: item()==item.

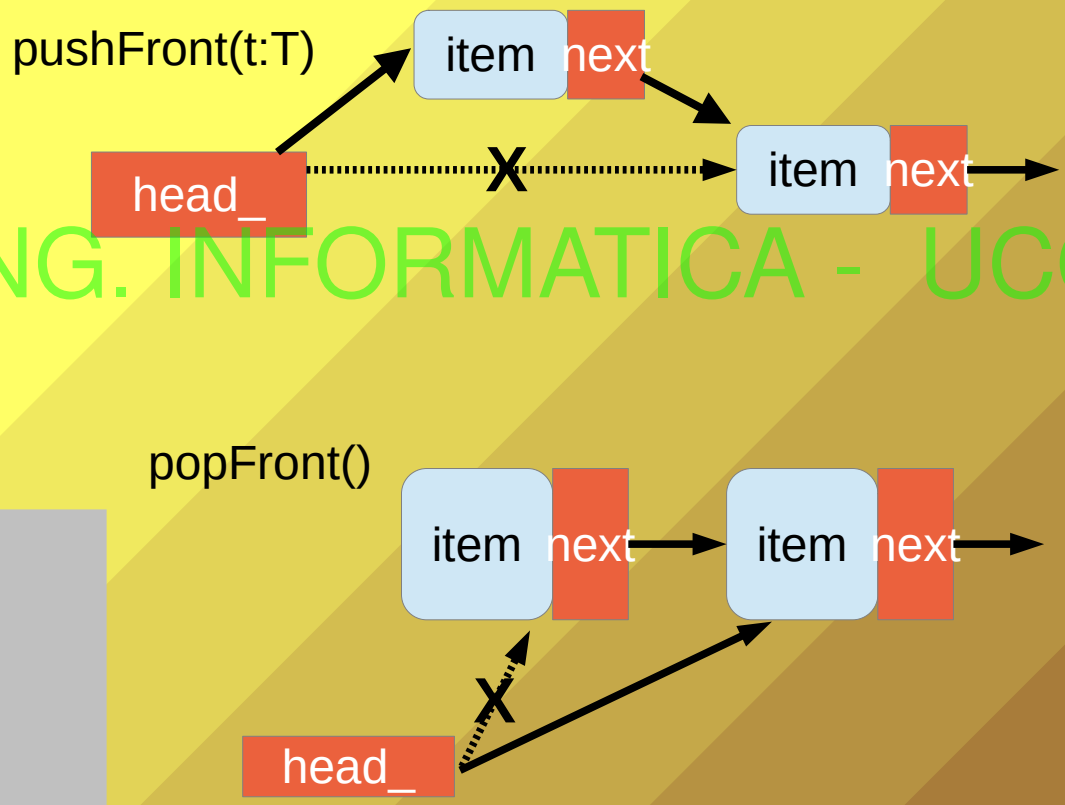
LNode[T]

```
item_:T
next_:LNode[T]
```

Lista Simple

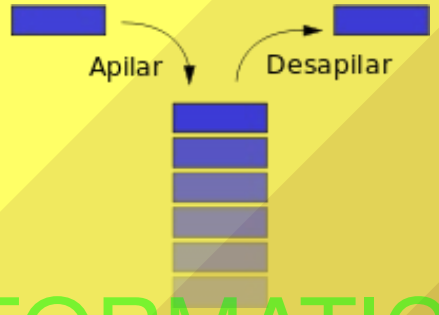
- SList: diseño con nodos enlazados.

```
SList::create()
  head_ ← Void (nullptr)

SList::isEmpty():Boolean
  Return head_ = Void

SList::size():Integer
  Return size_

SList::front():T
  Return head_.item()

SList::pushFront(item:T) // O(1)
   head_ ← LNode::make(item, head_)
   size_ ← size_ + 1

SList::popFront() // O(1)
  head_ ← head_.next()
  size_ ← size_ - 1
```



Pilas = lista simple

- Adapta el acceso a una lista al paradigma LIFO (Last-In-First-Out).
- ADT: Stack[T]
  - **Makers**:
    - create():Stack[T] //create empty stack
      - post-c: isEmpty().
  - **Observers**:
    - isEmpty():Boolean //Is the stack empty?
    - size():Integer //How many items?
    - T top() // gets the last inserted item.
      - pre-c: not isEmpty()
  - **Modifiers**:
    - push(T it) //insert item in the stack.
      - post-c: not isEmpty()
      - post-c: top() = it
    - pop() //Delete the last inserted item.
      - pre-c: not isEmpty().
  - **Invariants**:
    - isEmpty() or size()>0



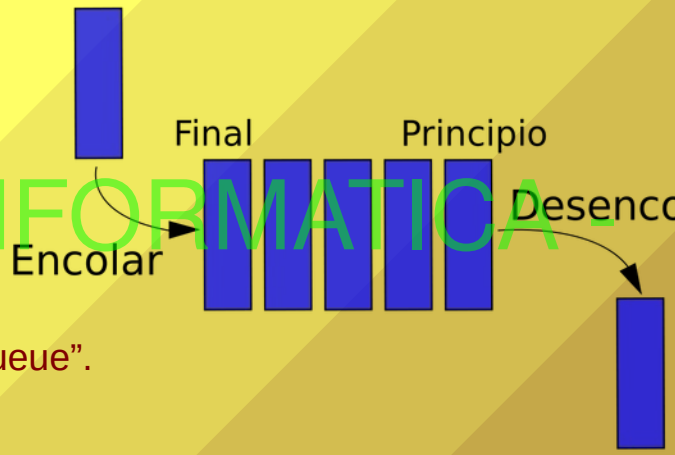
Stack[T]

```
l_:SList<T>
```

```
isEmpty(): Bool //O( )
   Return l_.isEmpty()
size(): Integer //O( )
   return l_.size()
top():T //O( )
   return l_.front()
push(it:T) //O(n) / CA (1)
   l_.pushFront(it)
pop() //O( )
   l_.popFront()
```

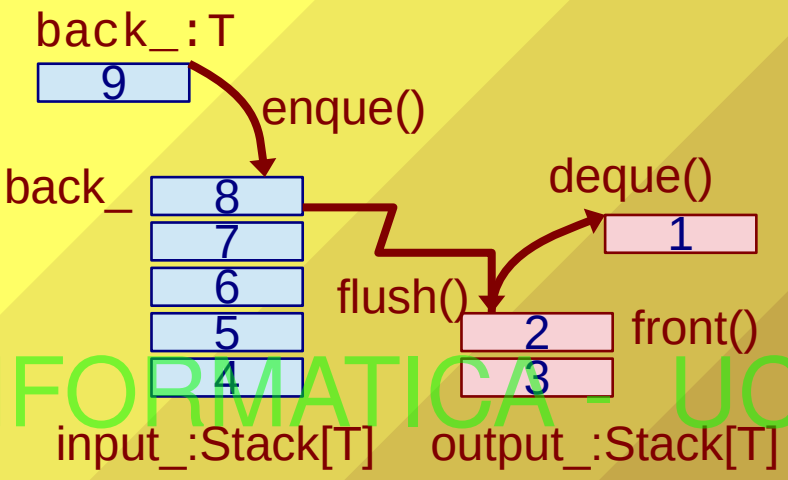
Colas

- Adapta el acceso a una lista al paradigma FIFO (First-In-First-Out).
- ADT Queue[T]
  - **Makers**:
    - make():Queue[T]
      - post-c: isEmpty()
  - **Observers**:
    - isEmpty():Bool
    - size():Integer
    - front():T
      - pre-c: not isEmpty().
      - post-c: front == “oldest inserted item in the queue”.
    - back():T
      - pre-c: not isEmpty().
      - post-c: back == “newest inserted item in the queue”.
  - **Modifiers**:
    - enque(it:T) encolar
      - post-c: not isEmpty()
      - post-c: back()==it
    - deque() desencolar
      - pre-c: not isEmpty()
      - post-c: isEmpty() or front()==”previous of old.front()”
  - **Invariants:**
    - isEmpty() or size()>0



Colas

- Diseño usando dos pilas.

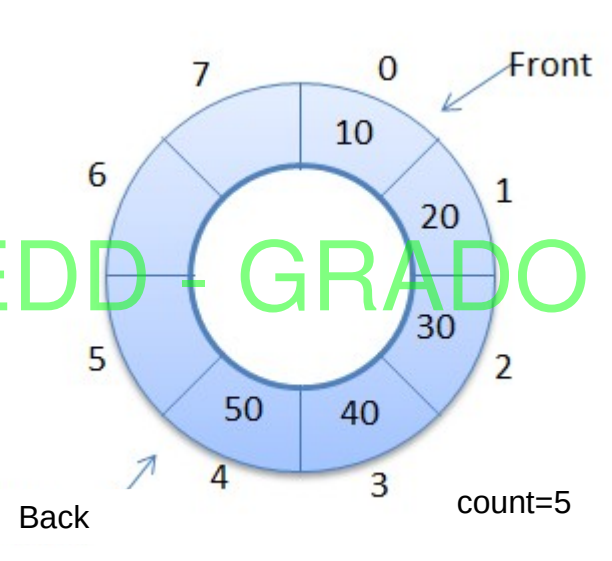



```
Queue::isEmpty():Boolean //O(1)
   return input_.isEmpty() and
             output_.isEmpty()
Queue::size():Integer //O(n)
   return input_.size()+output_.size()

Queue::front():T // O(n) CA(1)
   if ouput_.isEmpty() then
      flush()
   return output_.top()

Queue::back():T // O(1)
   return back_

Queue::enque(v:T) //O(1)
   back_<- v
   input_.push(v)
```

```
Queue::deque() //O(n) CA( 1 )
   if ouput_.isEmpty()
      flush()
   output_.pop()

Queue::flush() // O(n)
prec-c: not input_.isEmpty()
  back_ ← back()
  while !input_.isEMpty() do
   output_.push(input_.top())
    input_.pop()
```

Colas

- Diseño con un Array Dinámico Circular.



Queue[T]

```
data_:CDArray[T]
```

```
Queue::isEmpty(): Boolean //O(1)
    Return data_.isEmpty()

Queue::size(): Integer //O(1)
    Return datat_.size()

Queue::front():T //O(1)
    Return data_.front()

Queue::back():T //O(1)
    Return data_.back()

Queue::enque (T v):  //O(n) CA(1)
    data_.pushBack(v)

Queue::deque (): //O(1)
    Pre-c: not isEmpty()
    data_.popFront()
```

Resumiendo

- La Lista simple está pensada para el acceso/procesamiento secuencial sólo desde la cabeza.
- La pila adapta la lista al paradigma LIFO.
- La cola adapta la lista al paradigma FIFO.

Referencias

- Lecturas recomendadas:
  - Caps. 8 y 9 de “Estructuras de Datos”, A. Carmona y otros. U. de Córdoba. 1999.
  - Caps 6 y 7 de *“Data structures and software develpment in an object oriented domain”*, Tremblay J.P. y Cheston, G.A. Prentice-Hall, 2001.
  - Wikipedia.