

Estructuras de Datos

EEDD - GRADO EN INGENIERIA INFORMÁTICA - UCO

Estructuras lineales:
Lista simple, pilas y colas

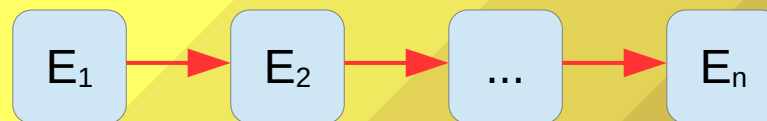
Contenidos

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

EEDD - GRADO EN ING. INFORMÁTICA - UCO

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:

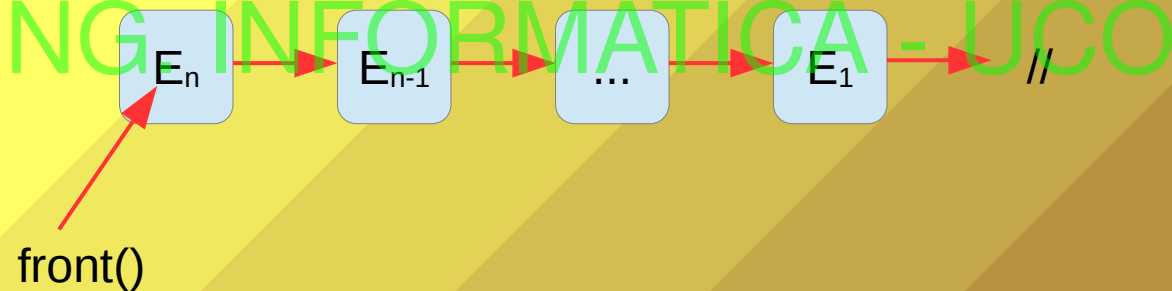
- `isEmpty():Boolean` //is the list empty?
- `size():Integer` //Number of items in the list.
- `front():T` //return the first Item of the list.
 - `pre-c: not isEmpty()`

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`

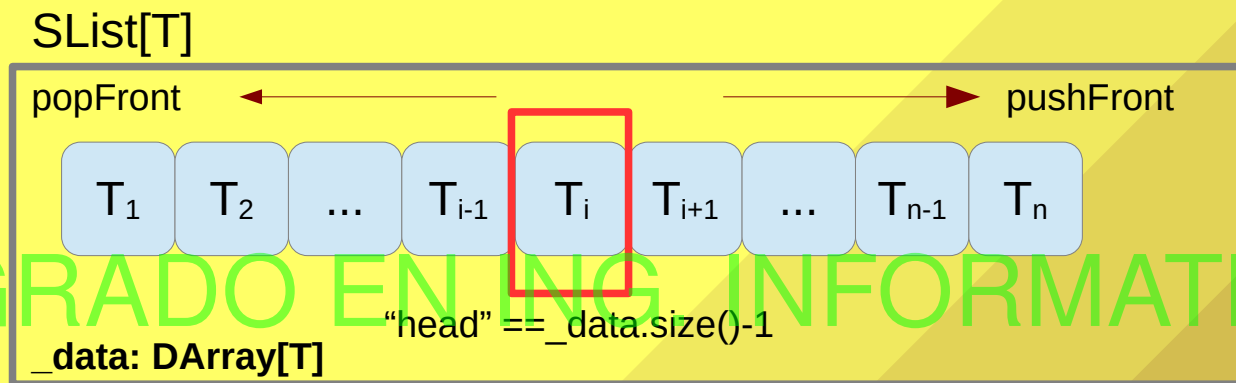
Invariants:

- `isEmpty()` or `size()>0`



Lista Simple

- **SList[T]:** diseño usando **DArray[T]**.

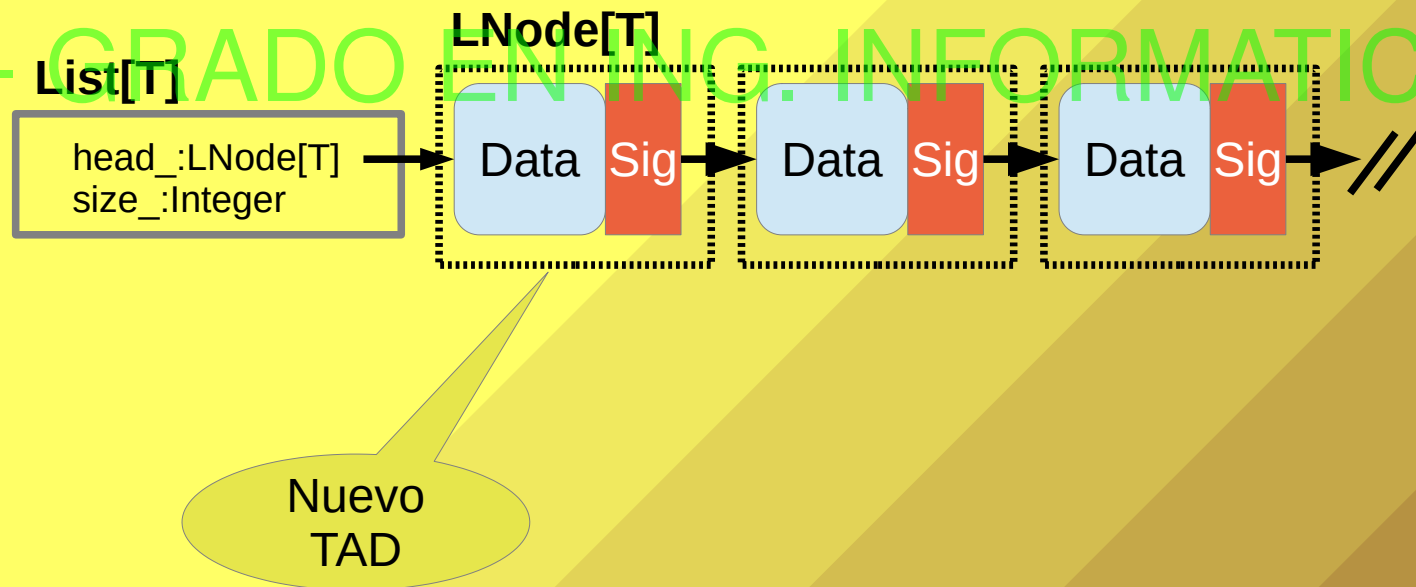


```
SList::create()  
    _data ← DArray()  
  
SList::isEmpty():Boolean  
    return _data.size()==0  
  
SList::size():Integer  
    return _data.size()  
  
SList::front():T //O( )  
    return _data.get(_data.size()-1)
```

```
SList::pushFront(newItem:T) //O( )CA()  
    _data.pushBack(newItem)  
  
SList::popFront() //O( )  
    _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.

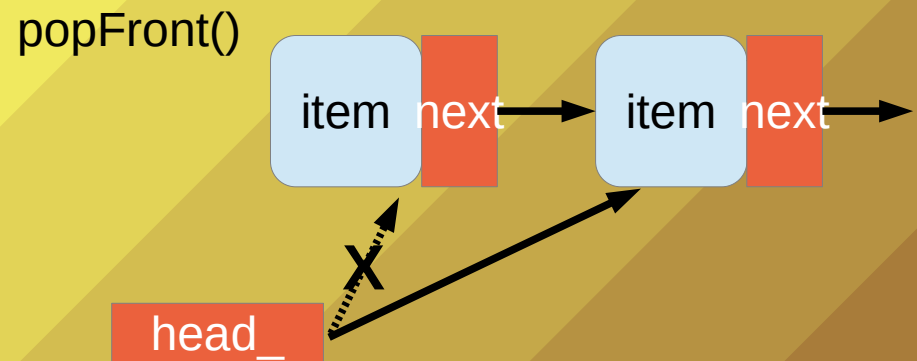
LNode[T]

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

Lista Simple

- SList: diseño con nodos enlazados.

```
SList::create()  
  head_ ← Void  
  
SList::isEmpty():Boolean  
  Return head_ = Void  
  
SList::size():Integer  
  Return size_  
  
SList::front():T  
  Return head_.item()  
  
SList::pushFront(item:T) // 0( )  
  head_ ← LNode::make(item, head_)  
  size_ ← size_ + 1  
  
SList::popFront() // 0( )  
  head_ ← head_.next()  
  size_ ← size_ - 1
```



Pilas

- 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:**

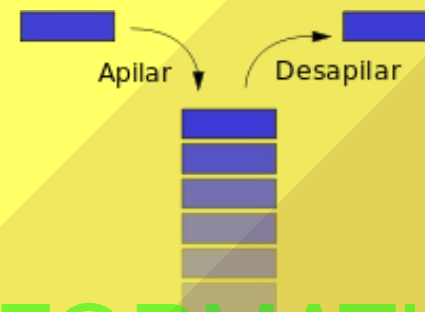
- isEmpty():Boolean //Is the stack empty?
- size():Integer //How many items?
- T top() // gets the last inserted item.
 - pre-c: not isEmpty()

- **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 //0( )
    Return l_.isEmpty()
size(): Integer //0( )
    return l_.size()
top():T //0( )
    return l_.front()
push(it:T) //0( ) || 0(),CA( )
    l_.pushFront(it) //0(?)
pop() //0( )
    l_.popFront()
```

Depende de
cómo se diseñe
la lista

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:**

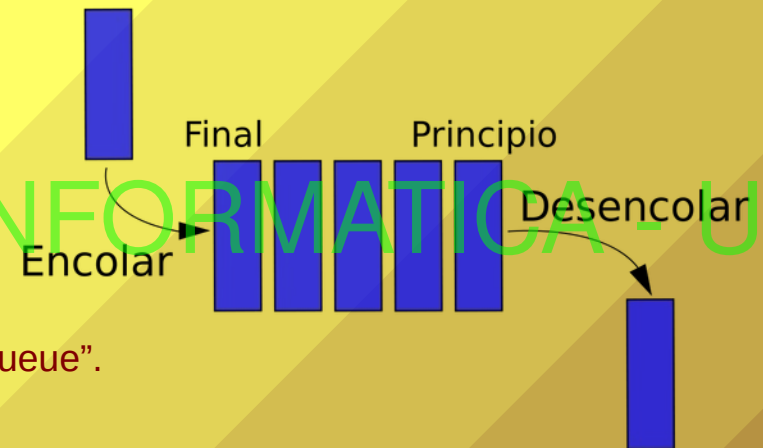
- isEmpty():Bool
 - size():Integer
 - front():T
 - pre-c: not isEmpty().
 - post-c: front == “oldest inserted item in the queue”.
 - back():T
 - pre-c: not isEmpty().
 - post-c: back == “newest inserted item in the queue”.

- **Modifiers:**

- enqueue(it:T)
 - post-c: not isEmpty()
 - post-c: back()==it
 - dequeue()
 - pre-c: not isEmpty()
 - post-c: isEmpty() or front()==“previous of old.front()”

- **Invariants:**

- isEmpty() or size()>0



Colas

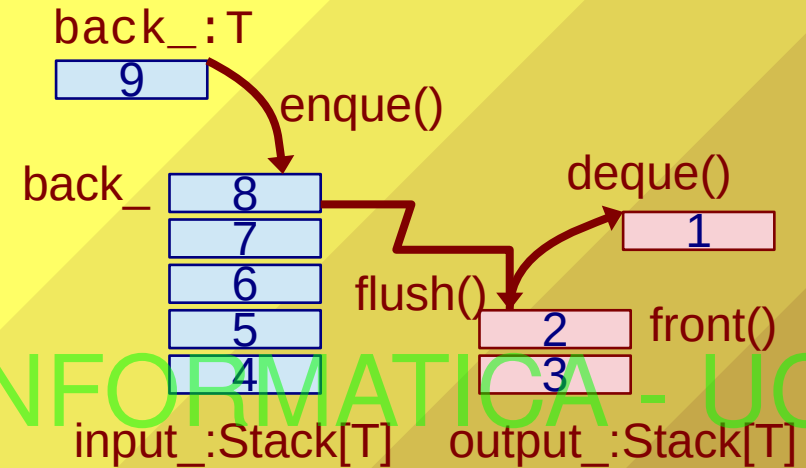
- Diseño usando dos pilas.

```
Queue::isEmpty():Boolean //0( )
    return input_.isEmpty() and
           output_.isEmpty()
Queue::size():Integer //0( )
    return input_.size()+output_.size()
Queue::front():T // 0( ) CA( )
    if output_.isEmpty() then
        flush()
    return output_.top()

Queue::back():T // 0( )
    return back_

Queue::enqueue(v:T) //0( )|| 0( ),CA( )
    back_<- v
    input_.push(v)
```

Depende de
cómo se diseñe
la pila

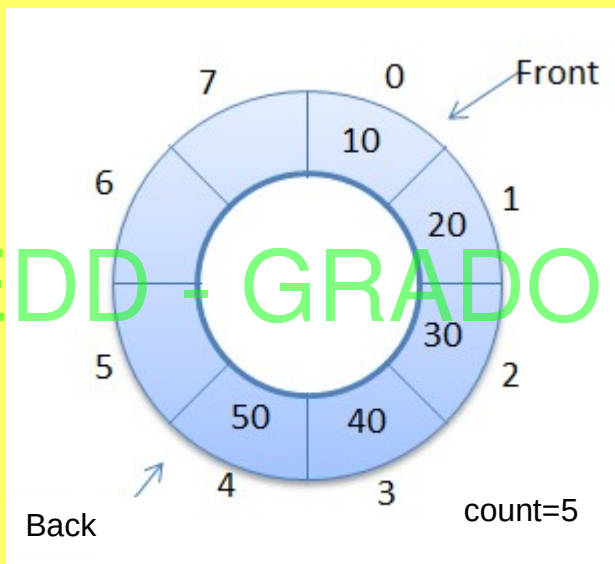


```
Queue::dequeue() //0( ) CA( )
    if output_.isEmpty()
        flush()
    output_.pop()

Queue::flush() // 0( )
    prec-c: not input_.isEmpty()
    While !input_.isEmpty() do
        output_.push(input_.top())
        input_.pop()
    End-While
```

Colas

- Diseño con un Array Dinámico Circular.



Queue[T]

data_: CArray[T]

```
Queue::isEmpty(): Boolean //0( )  
    Return data_.isEmpty()
```

```
Queue::size(): Integer //0( )  
    Return data_.size()
```

```
Queue::front(): T //0( )  
    Return data_.front()
```

```
Queue::back(): T //0( )  
    Return data_.back()
```

```
Queue::enqueue (T v): //0( ), CA ( )  
    data_.pushBack(v)
```

```
Queue::dequeue (): //0( )  
    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 development in an object oriented domain*”, Tremblay J.P. y Cheston, G.A. Prentice-Hall, 2001.
 - Wikipedia.