if (Is_Empty()){

BLOQUE 1: [size++ → ~~Null~~ ! Is_empty]

·Estructura de datos = Forma de [organizar] datos en la computadora. Se busca → Almacenamiento } Eficiente (tiempo y espacio)
→ Acceso

TAD: Conjunto con operaciones (clase).
→ Constructor !
→ Observador } Especificador → Formalmente
→ Modificador → Informalmente (Diseño por Contrato)

O: Complejidad de una operación

Array:

Make (size) → pre-c: ~~(...)~~ size>0
→ post-c: -

Size → pre-c: -
→ post-c: -

Get(i) → pre-c: i≥0 y i< size
→ post-c: -

Set(i,T) → pre-c: i≥0 y i<size
→ post-c: Get(i) == item(T)

Array dinámico: * En un array dinámico la última posición es [size-1]

Make → pre-c: [redacted] —
→ post-c: capacity = 1 // size = 0

Size → pre-c: —
→ post-c: —

Get(i) → pre-c: —
→ post-c: i ∈ [0, size)

Capacity → pre-c: —
→ post-c: —

Es_full → pre-c: —
→ post-c: —

Set(i,T) → pre-c: i ∈ [0, size)
→ post-c: Get(i) == item(T)

Push_back (T) → pre-c: — (¡is_empty <NO, ya que llama a grow!
→ post-c: size++ // get(size-1) == [redacted] item(T)

Pop_back → pre-c: size > 0
→ post-c: size-- // get(size-1) == get(size_old-2)

Insert (i,T) → pre-c: [redacted] i ∈ [0, size)
→ post-c: size++ // set(i) == item(T)

Remove(i) → pre-c: i ∈ [0, size)
→ post-c: size-- // set(i) == set(i+1)

Grow → pre-c: —
→ post-c: —

Invariante
size ≤ capacity

Array dinámico circular (Anterior) t;

CInc (i,T) → pre-c:
→ post-c:

CDec (i,T) → pre-c:
→ post-c:

Lista simple:

Lista con CDArray (Anterior)

Lista simple:

Nodo:

Create(T, Node) → pre-c: –
→ post-c: item() == item // next() == Node

Next → pre-c: –
→ post-c: –

Item → pre-c: –
→ post-c: –

Set_next (Node) → pre-c: –
→ post-c: next == Node

Set_item (T) → pre-c: –
→ post-c: item() == T

Lista simple

Create → pre-c: -
└→ post-c: is_Empty

Is_empty → pre-c: -
└→ post-c: -

Size → pre-c: -
└→ post-c: -

Front → pre-c: ! Is_empty
└→ post-c: -

Push_front (T) → pre-c: -
└→ post-c: front == T / size++

Pop_front → pre-c: ! Is_empty
└→ post-c: size--

Invariante
Is_empty or size>0

Pila = Lista simple ([illegible] pero front = top.
└→ Paradigma LIFO

Cola (Uso de 2 pilas)

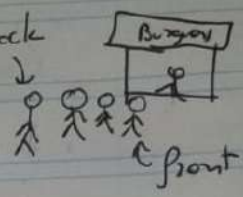
Create → pre-c: -
└→ post-c: Is_empty

Is_empty → pre-c: -
└→ post-c: -

Size → pre-c: -
└→ post-c: -

Front → pre-c: ! Is_empty
↳ post-c: front == "más antiguo"

Back → pre-c: ! Is_empty
↳ post-c: back == "más nuevo"



Enqueue → pre-c: —
(T) ↳ post-c: back == T // size++

Dequeue → pre-c: ! Is_empty
↳ post-c: size-- // front == "segundo más antiguo"

Cola (Usando (DArray) = Anteriores.

Iteradores:

~~Crear~~ (No existe)

Is_valid → pre-c: —
↳ post-c: —

Operator:: → pre-c: —
↳ post-c: —

Distance → pre-c: —
↳ post-c: —

Get → pre-c: "valid operator"
↳ post-c: —

Next(i) → pre-c: —
↳ post-c: distance (return de next) = i

Prev(i) → pre-c: —
↳ post-c: distance (return de prev) = i

set (T) → pre-c: "valid iterator"
→ post-c: get() == T

gotoNext (i) → pre-c: -
→ post-c: distance(this) = i

gotoPrev (i) → pre-c: -
→ post-c: distance(this) = i

Lista doblemente enlazada:

Create (T, Node*, Node* post) → pre-c: -
→ post-c: -

Back → pre-c: !Is_empty
→ post-c: -

~~Pred~~ End → pre-c: ~~Is_empty~~ -
→ post-c: -

Begin → pre-c: -
post-c: -

find (T, it) → pre-c: -
→ post-c: -

PushBack (T) → pre-c: -
→ post-c: size++ // back == T

PopBack → pre-c: ~~Is_empty~~ -
→ post-c: size--

Insert (T, it) → pre-c: -
→ post-c: size++ // item == T // next == it

Remove: → pre-c: "iterator is valid"
(it)
→ post-c: size-- // old next == ret

~~Invariante~~

~~!Is_empty or begin != end~~

Lista ordenada(Lista t:

Insert → pre-c: ~~[illegible]~~ -
(T)
→ post-c: current == T // not next // size++

Invariante

!Is_empty or begin() == end()