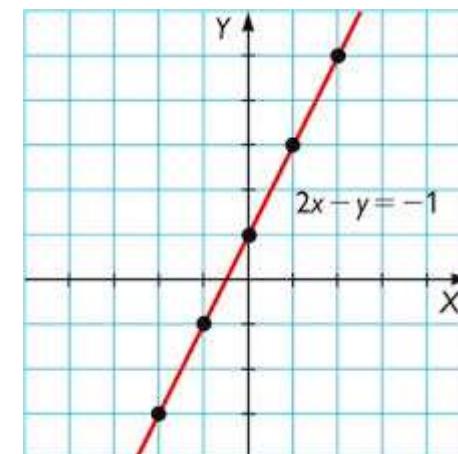


## 1.7 Estructuras de datos Lineales

---

## 1.7.1 Generalidades

- Las estructuras de datos lineales están compuestas por una secuencia de 0 o más elementos de algún tipo determinado en ocasiones ordenados de alguna forma.



## 1.7.1 Generalidades

- Se puede aumentar o disminuir la cantidad de datos y podrán insertarse o eliminarse elementos en determinadas posiciones sin alterar su estructura lógica.



## 1.7.1.1 El concepto de Lista

- Una lista es una colección de elementos del mismo tipo
- Para el acceso y manejo de esos elementos se utilizan índices o posiciones



## 1.7.1.1 El concepto de Lista

- Una lista tiene operaciones de inserción, eliminación y búsqueda
- Una lista es una estructura de datos flexible en las operaciones que se realizan

LISTA DE INSCRITOS	
NUMERO	NOMBRE
1	JOSE
2	ANNE
3	ROBERTO
4	CARLOS
5	MARIA
6	ANTONIO
7	PAULINA
8	EDUARDO
9	FRANCISCO
10	ISABEL
11	JUAN
12	ROBERTA
13	LUIS
14	ANGELA
15	JOSE
16	ANNE
17	ROBERTO
18	CARLOS
19	MARIA
20	ANTONIO
21	PAULINA
22	EDUARDO
23	FRANCISCO
24	ISABEL
25	JUAN
26	ROBERTA
27	LUIS
28	ANGELA
29	JOSE
30	ANNE
31	ROBERTO
32	CARLOS
33	MARIA
34	ANTONIO
35	PAULINA
36	EDUARDO
37	FRANCISCO
38	ISABEL
39	JUAN
40	ROBERTA
41	LUIS
42	ANGELA
43	JOSE
44	ANNE
45	ROBERTO
46	CARLOS
47	MARIA
48	ANTONIO
49	PAULINA
50	EDUARDO
51	FRANCISCO
52	ISABEL
53	JUAN
54	ROBERTA
55	LUIS
56	ANGELA
57	JOSE
58	ANNE
59	ROBERTO
60	CARLOS
61	MARIA
62	ANTONIO
63	PAULINA
64	EDUARDO
65	FRANCISCO
66	ISABEL
67	JUAN
68	ROBERTA
69	LUIS
70	ANGELA
71	JOSE
72	ANNE
73	ROBERTO
74	CARLOS
75	MARIA
76	ANTONIO
77	PAULINA
78	EDUARDO
79	FRANCISCO
80	ISABEL
81	JUAN
82	ROBERTA
83	LUIS
84	ANGELA
85	JOSE
86	ANNE
87	ROBERTO
88	CARLOS
89	MARIA
90	ANTONIO
91	PAULINA
92	EDUARDO
93	FRANCISCO
94	ISABEL
95	JUAN
96	ROBERTA
97	LUIS
98	ANGELA
99	JOSE
100	ANNE
101	ROBERTO
102	CARLOS
103	MARIA
104	ANTONIO
105	PAULINA
106	EDUARDO
107	FRANCISCO
108	ISABEL
109	JUAN
110	ROBERTA
111	LUIS
112	ANGELA
113	JOSE
114	ANNE
115	ROBERTO
116	CARLOS
117	MARIA
118	ANTONIO
119	PAULINA
120	EDUARDO
121	FRANCISCO
122	ISABEL
123	JUAN
124	ROBERTA
125	LUIS
126	ANGELA
127	JOSE
128	ANNE
129	ROBERTO
130	CARLOS
131	MARIA
132	ANTONIO
133	PAULINA
134	EDUARDO
135	FRANCISCO
136	ISABEL
137	JUAN
138	ROBERTA
139	LUIS
140	ANGELA
141	JOSE
142	ANNE
143	ROBERTO
144	CARLOS
145	MARIA
146	ANTONIO
147	PAULINA
148	EDUARDO
149	FRANCISCO
150	ISABEL
151	JUAN
152	ROBERTA
153	LUIS
154	ANGELA
155	JOSE
156	ANNE
157	ROBERTO
158	CARLOS
159	MARIA
160	ANTONIO
161	PAULINA
162	EDUARDO
163	FRANCISCO
164	ISABEL
165	JUAN
166	ROBERTA
167	LUIS
168	ANGELA
169	JOSE
170	ANNE
171	ROBERTO
172	CARLOS
173	MARIA
174	ANTONIO
175	PAULINA
176	EDUARDO
177	FRANCISCO
178	ISABEL
179	JUAN
180	ROBERTA
181	LUIS
182	ANGELA
183	JOSE
184	ANNE
185	ROBERTO
186	CARLOS
187	MARIA
188	ANTONIO
189	PAULINA
190	EDUARDO
191	FRANCISCO
192	ISABEL
193	JUAN
194	ROBERTA
195	LUIS
196	ANGELA
197	JOSE
198	ANNE
199	ROBERTO
200	CARLOS
201	MARIA
202	ANTONIO
203	PAULINA
204	EDUARDO
205	FRANCISCO
206	ISABEL
207	JUAN
208	ROBERTA
209	LUIS
210	ANGELA
211	JOSE
212	ANNE
213	ROBERTO
214	CARLOS
215	MARIA
216	ANTONIO
217	PAULINA
218	EDUARDO
219	FRANCISCO
220	ISABEL
221	JUAN
222	ROBERTA
223	LUIS
224	ANGELA
225	JOSE
226	ANNE
227	ROBERTO
228	CARLOS
229	MARIA
230	ANTONIO
231	PAULINA
232	EDUARDO
233	FRANCISCO
234	ISABEL
235	JUAN
236	ROBERTA
237	LUIS
238	ANGELA
239	JOSE
240	ANNE
241	ROBERTO
242	CARLOS
243	MARIA
244	ANTONIO
245	PAULINA
246	EDUARDO
247	FRANCISCO
248	ISABEL
249	JUAN
250	ROBERTA
251	LUIS
252	ANGELA
253	JOSE
254	ANNE
255	ROBERTO
256	CARLOS
257	MARIA
258	ANTONIO
259	PAULINA
260	EDUARDO
261	FRANCISCO
262	ISABEL
263	JUAN
264	ROBERTA
265	LUIS
266	ANGELA
267	JOSE
268	ANNE
269	ROBERTO
270	CARLOS
271	MARIA
272	ANTONIO
273	PAULINA
274	EDUARDO
275	FRANCISCO
276	ISABEL
277	JUAN
278	ROBERTA
279	LUIS
280	ANGELA
281	JOSE
282	ANNE
283	ROBERTO
284	CARLOS
285	MARIA
286	ANTONIO
287	PAULINA
288	EDUARDO
289	FRANCISCO
290	ISABEL
291	JUAN
292	ROBERTA
293	LUIS
294	ANGELA
295	JOSE
296	ANNE
297	ROBERTO
298	CARLOS
299	MARIA
300	ANTONIO
301	PAULINA
302	EDUARDO
303	FRANCISCO
304	ISABEL
305	JUAN
306	ROBERTA
307	LUIS
308	ANGELA
309	JOSE
310	ANNE
311	ROBERTO
312	CARLOS
313	MARIA
314	ANTONIO
315	PAULINA
316	EDUARDO
317	FRANCISCO
318	ISABEL
319	JUAN
320	ROBERTA
321	LUIS
322	ANGELA
323	JOSE
324	ANNE
325	ROBERTO
326	CARLOS
327	MARIA
328	ANTONIO
329	PAULINA
330	EDUARDO
331	FRANCISCO
332	ISABEL
333	JUAN
334	ROBERTA
335	LUIS
336	ANGELA
337	JOSE
338	ANNE
339	ROBERTO
340	CARLOS
341	MARIA
342	ANTONIO
343	PAULINA
344	EDUARDO
345	FRANCISCO
346	ISABEL
347	JUAN
348	ROBERTA
349	LUIS
350	ANGELA
351	JOSE
352	ANNE
353	ROBERTO
354	CARLOS
355	MARIA
356	ANTONIO
357	PAULINA
358	EDUARDO
359	FRANCISCO
360	ISABEL
361	JUAN
362	ROBERTA
363	LUIS
364	ANGELA
365	JOSE
366	ANNE
367	ROBERTO
368	CARLOS
369	MARIA
370	ANTONIO
371	PAULINA
372	EDUARDO
373	FRANCISCO
374	ISABEL
375	JUAN
376	ROBERTA
377	LUIS
378	ANGELA
379	JOSE
380	ANNE
381	ROBERTO
382	CARLOS
383	MARIA
384	ANTONIO
385	PAULINA
386	EDUARDO
387	FRANCISCO
388	ISABEL
389	JUAN
390	ROBERTA
391	LUIS
392	ANGELA
393	JOSE
394	ANNE
395	ROBERTO
396	CARLOS
397	MARIA
398	ANTONIO
399	PAULINA
400	EDUARDO
401	FRANCISCO
402	ISABEL
403	JUAN
404	ROBERTA
405	LUIS
406	ANGELA
407	JOSE
408	ANNE
409	ROBERTO
410	CARLOS
411	MARIA
412	ANTONIO
413	PAULINA
414	EDUARDO
415	FRANCISCO
416	ISABEL
417	JUAN
418	ROBERTA
419	LUIS
420	ANGELA
421	JOSE
422	ANNE
423	ROBERTO
424	CARLOS
425	MARIA
426	ANTONIO
427	PAULINA
428	EDUARDO
429	FRANCISCO
430	ISABEL
431	JUAN
432	ROBERTA
433	LUIS
434	ANGELA
435	JOSE
436	ANNE
437	ROBERTO
438	CARLOS
439	MARIA
440	ANTONIO
441	PAULINA
442	EDUARDO
443	FRANCISCO
444	ISABEL
445	JUAN
446	ROBERTA
447	LUIS
448	ANGELA
449	JOSE
450	ANNE
451	ROBERTO
452	CARLOS
453	MARIA
454	ANTONIO
455	PAULINA
456	EDUARDO
457	FRANCISCO
458	ISABEL
459	JUAN
460	ROBERTA
461	LUIS
462	ANGELA
463	JOSE
464	ANNE
465	ROBERTO
466	CARLOS
467	MARIA
468	ANTONIO
469	PAULINA
470	EDUARDO
471	FRANCISCO
472	ISABEL
473	JUAN
474	ROBERTA
475	LUIS
476	ANGELA
477	JOSE
478	ANNE
479	ROBERTO
480	CARLOS
481	MARIA
482	ANTONIO
483	PAULINA
484	EDUARDO
485	FRANCISCO
486	ISABEL
487	JUAN
488	ROBERTA
489	LUIS
490	ANGELA
491	JOSE
492	ANNE
493	ROBERTO
494	CARLOS
495	MARIA
496	ANTONIO
497	PAULINA
498	EDUARDO
499	FRANCISCO
500	ISABEL



## 1.7.2 Pila (Stack)

## 1.7.2 Pila

- Una pila (stack) es un tipo abstracto de datos tipo LIFO (*last in, first out*).
- Es un tipo de dato que soporta operaciones de inserción y eliminación de elementos de un conjunto.
- Normalmente se usa en aplicaciones en las que se necesita recuperar información en orden inverso a como ha entrado.

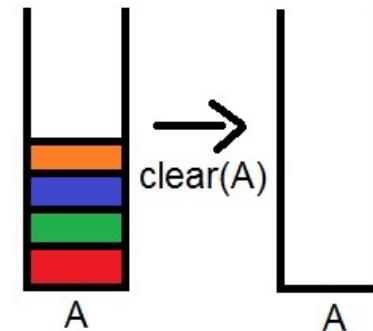
## 1.7.2 Pila

- Esta estructura funciona de forma similar a una torre de elementos del mismo tipo (documentos pendientes, platos, libros, etc).
- Cada vez que llega un elemento, se coloca arriba de los que ya están acumulados, y cuando se van atendiendo se toma el que se encuentra en la parte superior.

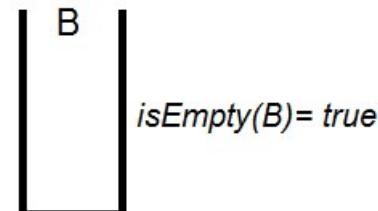
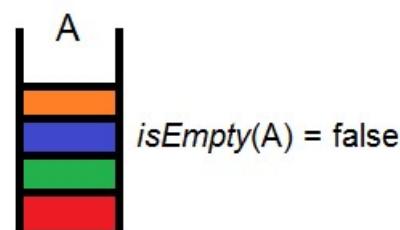
# Operaciones (Pila)

- Las operaciones que se realizan sobre una pila son:

- *clear()* Borra todos los elementos de una pila

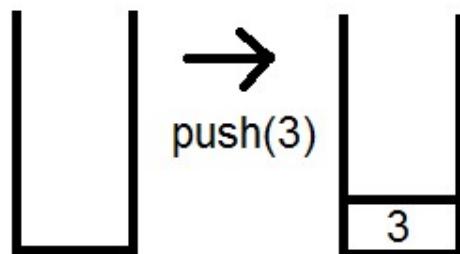


- *isEmpty()* Verifica si una pila se encuentra vacía

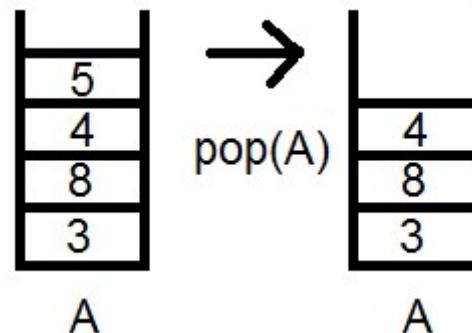


# Operaciones (Pila)

- *push()* Ingresa un elemento *en* el tope de la pila

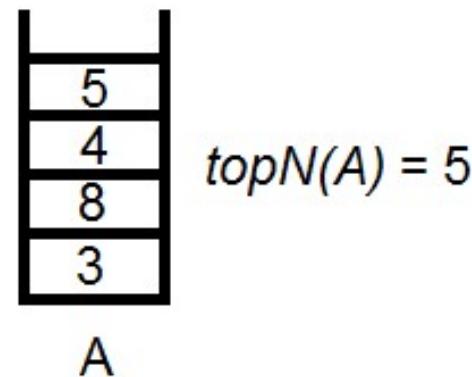


- *pop()* *Retira el elemento superior*



# Operaciones (Pila)

- *top()* Verifica el elemento que se encuentra en la parte superior de la pila sin eliminarlo.



# Operaciones (Pila)

- La complejidad temporal asintótica de las operaciones es:

Operación	Orden
Create	$O(1)$
clear()	$O(n)$
isEmpty()	$O(1)$
Push()	$O(1)$
Pop()	$O(1)$
top	$O(1)$

# Implementación (Pila)

**Pila:**

```
int tope;  
Lista elementos;
```

**Pila crearPila()**

```
Pila p;  
Lista listal;  
p.tope = -1;  
p.elementos = listal;  
return p;
```

# Implementación (Pila)

```
bool esVacia(Pila p)          //isEmpty(..)
    if(p.tope== -1)
        return true;
    return false;

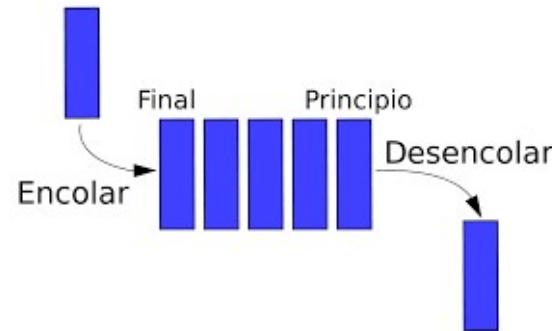
void meter(Pila p,int x)      //void push(..)
    p.tope = p.tope+1
    p.elementos[p.tope] = x
```

# Implementación (Pila)

```
int sacar(Pila p) // int pop(..)
    if(esVacia(p))
        print("Error")
        return -1
    else
        int aux= p.elementos[p.tope]
        p.elementos[p.tope]=NULL;
        p.tope=p.tope-1
        return aux
```

# Implementación (Pila)

```
int top(Pila p)
    if(esVacia(p))
        print("Error");
        return -1;
    else
        return p.elementos[p.tope]
```



## 1.7.3 Cola (Queue)

---

## 1.7.3 Cola

- Una cola (queue) es un tipo abstracto de datos de tipo FIFO (first in first out).
- Es un tipo de dato que soporta operaciones de inserción y eliminación de elementos de un conjunto de datos



## 1.7.3 Cola

- Modela situaciones en las que una tarea se realiza en el mismo orden en el que se reciben.
- Ejemplos de ello son tareas que esperan ser atendidas por alguna aplicación en la computadora o en un servidor.
- Los cambios que se realizan, se deben monitorear en ambos extremos de la estructura

# Operaciones (Queue)

- Las operaciones que se realizan sobre una cola son:
  - *create()* Crea una cola vacía.
  - *isEmpty()* Devuelve verdadero si una cola se encuentra vacía.
  - *enqueue()* Ingresa un elemento a la cola

# Operaciones (Queue)

- *dequeue()* Extraer el elemento al inicio de la cola.
- *clear()* Borra todos los elementos de la cola.
- *first()* Regresa el primer elemento de la cola sin eliminarlo.

# Operaciones (Queue)

- La complejidad temporal asintótica de las operaciones es:

Operación	Orden
Create	$O(1)$
clear()	$O(n)$
isEmpty()	$O(1)$
enqueue()	$O(1)$
dequeue()	$O(1)$
first(n)	$O(1)$

# Implementación (Queue)

**Cola:**

```
int primero;  
int ultimo;  
Lista elementos;
```

**Queue CrearCola ()**

```
Queue c;  
Lista miLista;  
c.primer = 0  
c.ultimo = -1  
c.elementos = miLista;  
return c;
```

# Implementación (Queue)

```
bool esVacia(Queue c)
    if(c.primero==c.ultimo+1)
        return true;
    return false;
```

```
void encolar(Queue c,int x)
    c.ultimo = c.ultimo+1
    c.elementos[c.ultimo] = x
```

# Implementación (Queue)

```
int desencolar(Queue c)    // (dequeue)
    if(esVacia(c))
        "error"
        return -1
    else
        int aux = c.elementos[c.primero]
        c.elementos[c.primero]=null;
        c.primero = c.primero + 1
        if(primer==ultimo+1)
            c=crearCola()
        return aux
```

## 1.7.3.4 Cola Circular

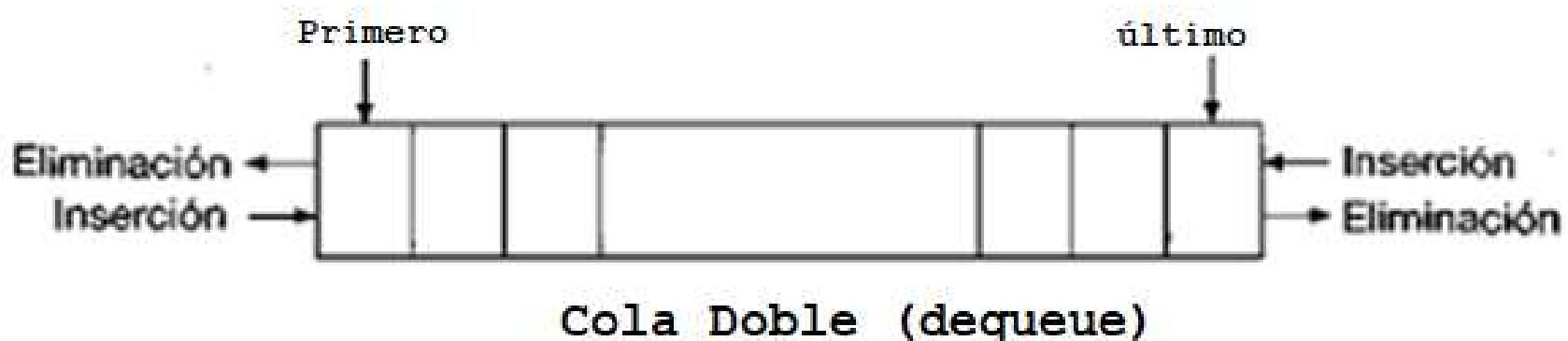
- Los elementos están de forma “circular”, lo que implica que todos los elementos tienen sucesor y predecesor.
- El sucesor del último elemento se trata del primer elemento mientras que el antecesor del primero es el último elemento.
- En una cola circular es indispensable definir el tamaño de la estructura
- La ventaja de una Cola circular es que el manejo de memoria es más eficiente

## 1.7.3.4 Cola Circular (operaciones)

- Las operaciones con la cola circular, son las mismas que en una “queue” simple (*enqueue*, *dequeue*, *first*, etc).
- El usuario no ve diferencia cuando utiliza una u otra
- **En la implementación** se requiere añadir elementos a las funciones encolar y desencolar, para el correcto manejo de índices (si se implementa con arreglos) o para el correcto manejo de memoria (si se implementa con listas ligadas)

### 1.7.3.5 Cola Doble

- Una cola doble (dqueue) es un tipo abstracto de datos en donde cada elemento puede ser ingresado y recuperado por ambos lados de la estructura.
- Con esto se da una mayor flexibilidad al tipo de dato ya que el elemento que ingresa puede ser el primero o el último en salir.



# Operaciones (Cola doble)

- *create()* Crea una cola vacía.
- *isEmpty()* Devuelve verdadero si una cola se encuentra vacía.
- *enqueueEnd()*: Ingresa un elemento al final de la cola.
- *dequeueBegin()*: Elimina y devuelve el elemento que se encuentra al principio de la cola.
- *enqueueBegin()*: Ingresa un elemento al principio de la cola.
- *dequeueEnd()*: Elimina y devuelve el elemento que este a final de la cola.

# Cola de Prioridad

Se caracteriza por admitir inserciones, consulta y eliminación de elementos con valores de prioridad ponderados.

Cuando ingresa un elemento se coloca en la última posición entre aquellos con su misma prioridad

# Aplicaciones (Pila)

- Evaluación de expresiones aritméticas (1)

## **Shunting-yard Algorithm**

Propuesto por Edsger W. Dijkstra en la década de los 60's.  
Utiliza dos pilas, una para operandos y una para operadores.

Para este tipo de expresiones se requiere hacer una lectura  
de la expresión revisando cada uno de los caracteres que la  
conforman

# Aplicaciones (Pila)

Dada una expresión aritmética (leyendo de izquierda a derecha)

- Ignorar los paréntesis de apertura
- Operando: **Push(opn)** en stack de operandos
- Operador: **Push(opr)** en stack de operadores
- Al encontrar un paréntesis de cierre
  - **Pop** operando
  - **Pop** operador
  - **Pop** operando
  - Realizar operación
  - **Push** resultado en stack de operandos
- Al finalizar la expresión
  - Pop resultado final.

# Aplicaciones (Pila)

- Evaluación de expresiones aritméticas (2).

## **Reverse Polish Notation**

La ventaja de esta forma de escribir expresiones aritméticas, es que no es necesario utilizar paréntesis ni reglas de precedencia.

En este algoritmo se utiliza una sola pila (stack).

# Aplicaciones (Pila)

Sea una expresión aritmética (leyendo de izquierda a derecha)

- Si se encuentra un operando:
  - **Push** operando.
- Si se encuentra un operador:
  - **Pop** operando.
  - **Pop** operando.
  - **Push** resultado.

# Aplicaciones (Queue)

- Algunos ejemplos de uso de “queues”:
  - Procesos atendidos por un procesador
  - Documentos pendientes de imprimir
  - En redes de computadoras, los paquetes suelen transportarse a través de colas.

