Pilas y Colas

Cursos Propedéuticos 2015

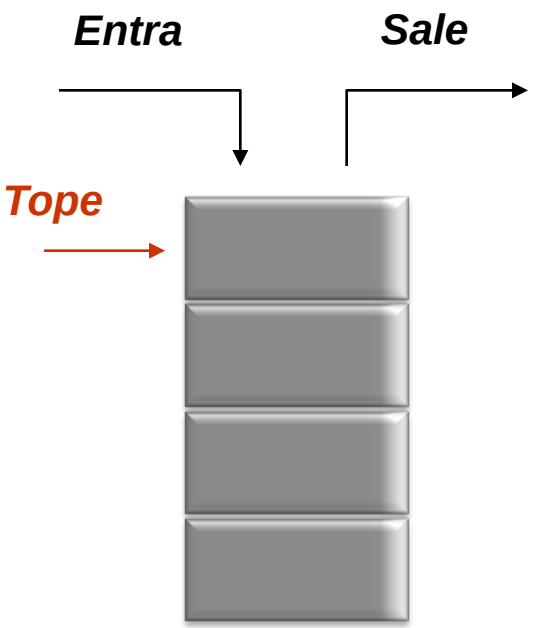
Dr. René Cumplido
M. en C. Luis Rodríguez Flores

Contenido de la sección

- Pilas
  - Estructura
  - Operaciones
  - Ejemplos de aplicación
  - Implementación
- Colas
  - Definición
  - Operaciones
  - Ejemplos de aplicación
  - Implementación
- Colas con prioridad
- Colas circulares

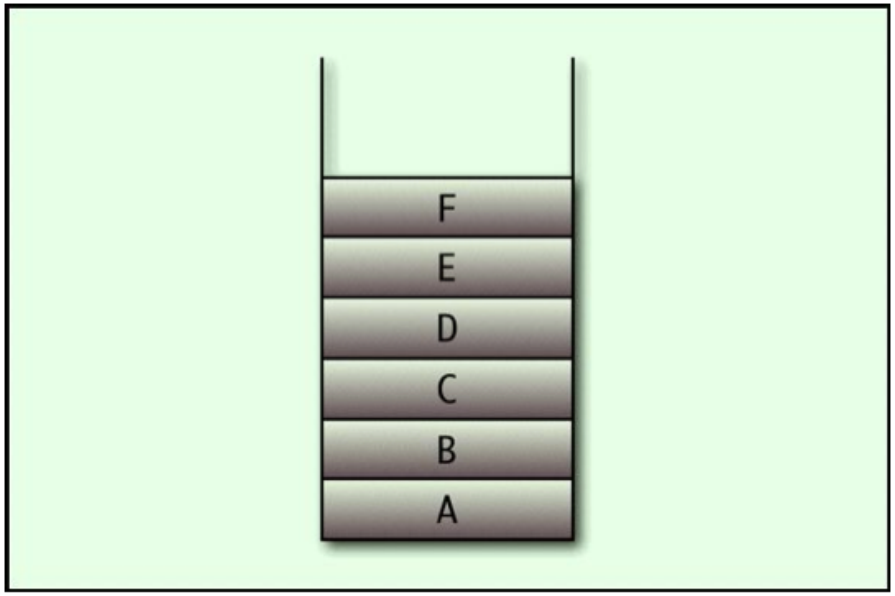
La estructura de datos PILA

- Su nombre se deriva de la metáfora de una pila de platos en una cocina.
- La inserción y extracción de elementos de la pila siguen el principio LIFO (*last-in-first-out*).
- El último elemento en entrar es el único accesible en cada momento.

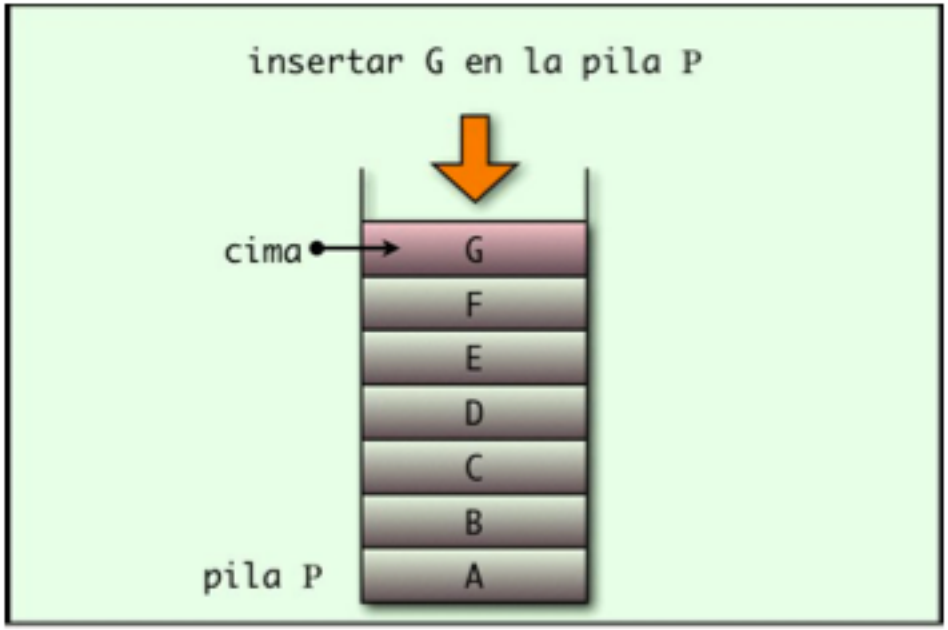


Definición de Pila

- *Una pila (stack) es una colecciona ordenada de elementos en la cual los datos se insertan o se retiran por el mismo extremo llamado “parte superior” de la pila.*

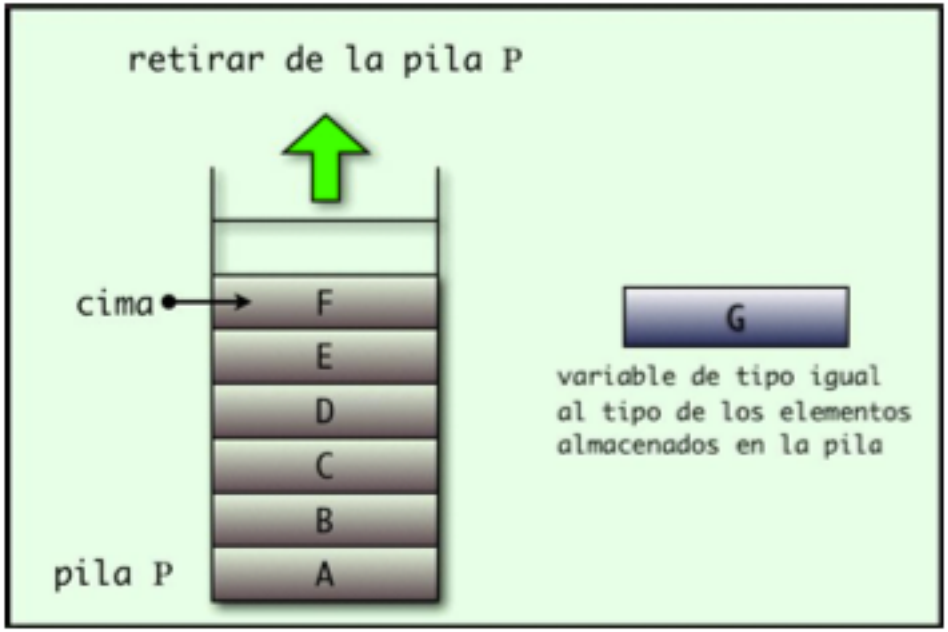


Operaciones básicas en Pilas (Push)



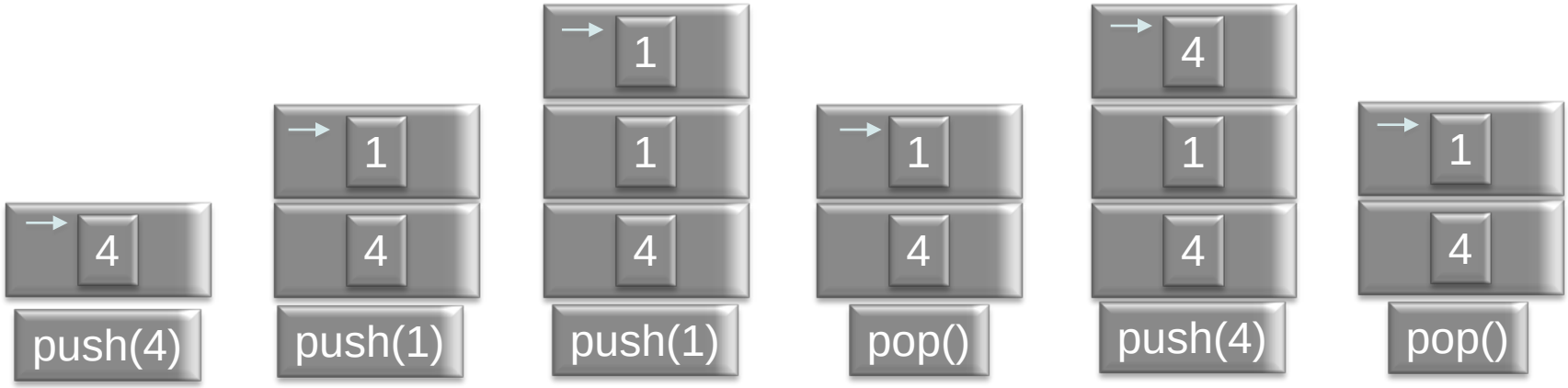
Existe solamente un lugar en donde cualquier elemento puede ser agregado a la pila. Después de haber insertado el nuevo elemento, G ahora es el elemento en la cima.

Operaciones básicas en Pilas (Pop)



Basta indicar que sea retirado un elemento. No podemos decir “retirar C”, porque C no está en la cima de la pila.

Ejemplo



La dinámica de la pila, es decir, la manera en cómo entran y salen los datos a la estructura de datos se denomina fifo (*first input, first output*)

Utilidad de las Pilas

El concepto de pila es muy importante en computación y en especial en teoría de lenguajes de programación. En lenguajes procedurales como Pascal o C, la pila es una estructura indispensable, debido a las llamadas a función. Por que?

Cuando ocurre una llamada a alguna función, el estado global del sistema se almacena en un registro y éste en una pila. Cuando se termina de ejecutar algún procedimiento, se recupera el registro que está en la cima de la pila.

Otras aplicaciones de las Pilas?

Aplicaciones de las pilas

- Navegador Web
  - Se almacenan los sitios previamente visitados
  - Cuando el usuario quiere regresar (presiona el botón de retroceso), simplemente se extrae la última dirección (*pop)* de la pila de sitios visitados.
- Editores de texto
  - Los cambios efectuados se almacenan en una pila
  - Usualmente implementada como arreglo
  - Usuario puede deshacer los cambios mediante la operación “*undo*”, la cual extraer el estado del texto antes del último cambio realizado.

Balance de Paréntesis

Es útil poder detectar si es que los paréntesis en un archivo fuente están o no balanceados.

Se puede usar un stack: a+(b+c)*[(d+e])/f
Pseudo-código:

Crear el stack.

Mientras no se ha llegado al final del archivo de entrada:

- *Descartar símbolos que no necesiten ser balanceados.*
- *Si es un paréntesis de apertura: poner en el stack.*
- *Si es un paréntesis de cierre, efectuar un pop y comparar.*
- *Si son de igual tipo continuar*
- *Si son de diferente tipo: avisar el error.*
- *Si se llega al fin de archivo, y el stack no esta vacío: avisar error.*

Implementación de las Pilas

Una pila está conformada por dos elementos:

• Un espacio suficiente para almacenar los elementos insertados en la pila.
• Una elemento que indique cuál es el elemento en la cima de la pila.

La estructura Pila:

definir nuevo tipo estructura llamado "stack" con item :

un arreglo de 0 a “máximos” elementos enteros
top : un numero de -1 a (máximos – 1)

fin de la nueva estructura

Implementación de las Pilas

Es posible escribir un código en C/C++ que represente lo anteriormente propuesto.

```
// En la parte de definiciones
#define maxElem 100

// En la parte de tipos
struct stack {
  int item[maxElem];
  int top;
};

// En la parte de variables
struct stack A;
```

Implementación con arreglos

- Una pila es una colección ordenada de objetos.
- En C, los arreglos permiten almacenar colecciones ordenadas.
- La desventaja de implementar una pila mediante un arreglo es que esta última es de tamaño fijo, mientras que usando punteros la pila puede ser de tamaño dinámico.

```
#define STACKSIZE 100
typedef struct stack{
    int top;
    int nodes[STACKSIZE];
}Stack;

Stack *create_stack
                (Stack *S) {
    S= (Stack *)
        malloc(sizeof(Stack));
    S->top=-1;
    return S;
}

main(){
   Stack *S;
   S=create_stack(S);
.
}
```

Operaciones en Pilas

Las operaciones básicas de una pila son:

1. En la pila S, insertar un elemento e: **push(S,e)**

2. Retirar un elemento de la pila S: **pop(S)**

3. Verificar si la pila S está vacía: **stackempty(S)**

4. Saber cuál es el elemento en la cima de la pila S: **stacktop(S).**

La Operación Push

Esta operación sirve para insertar un elemento e en la pila S, lo vamos a escribir como: **push(S,e).**
Después de hacer esta operación sucede que: El elemento en la cima de la pila S ahora es e

La Operación Push

(1) La operación push recibe : la dirección de una estructura pila y un elemento entero.
(2) Incrementa el tope (cima) de la pila para agregar el elemento en una posición libre de la pila.
(3) Asignando el valor e en la casilla S->top.

```
(1) void push(struct stack *S,int e){
(2)  S->top++;
(3)  S->item[S->top]=e;
(4) }
```

La Operación Push

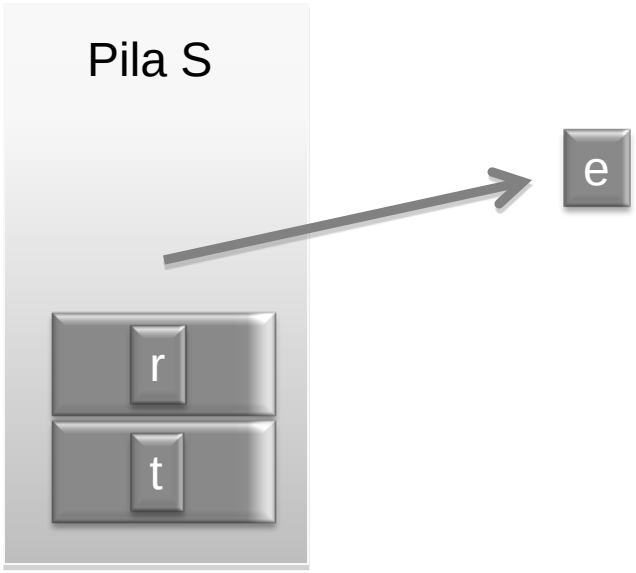
(1) La operación push recibe : la dirección de una estructura pila y un elemento entero.
(2) Incrementa el tope (cima) de la pila para agregar el elemento en una posición libre de la pila.
(3) Asignando el valor e en la casilla S->top.

```
void push(Stack *S, int e) {

 if(S->top == (STACKSIZE-1))
   printf(“Full stack\n");
 else {
   S->top++;
   S->nodes[S->top] = e;
 }
}
```

La Operación Pop

Esta operación sirve para retirar el elemento en la cima de la pila S, lo vamos a escribir como: **pop(S,e).**



La Operación Pop

(1) La función devuelve un tipo entero al recibir la dirección de una variable de tipo estructura pila (struct stack *). Las líneas (4) y (5) son las mas importantes ya que se almacena el valor que ser devuelto y se decrementa el tope de la pila.

```
(1) int pop(struct stack *S){
(2)  int valReturn;
(3)
(4)  valReturn=S->item[S->top];
(5)  S->top--;
(6)  return valReturn;
(7) }
```

La Operación Pop

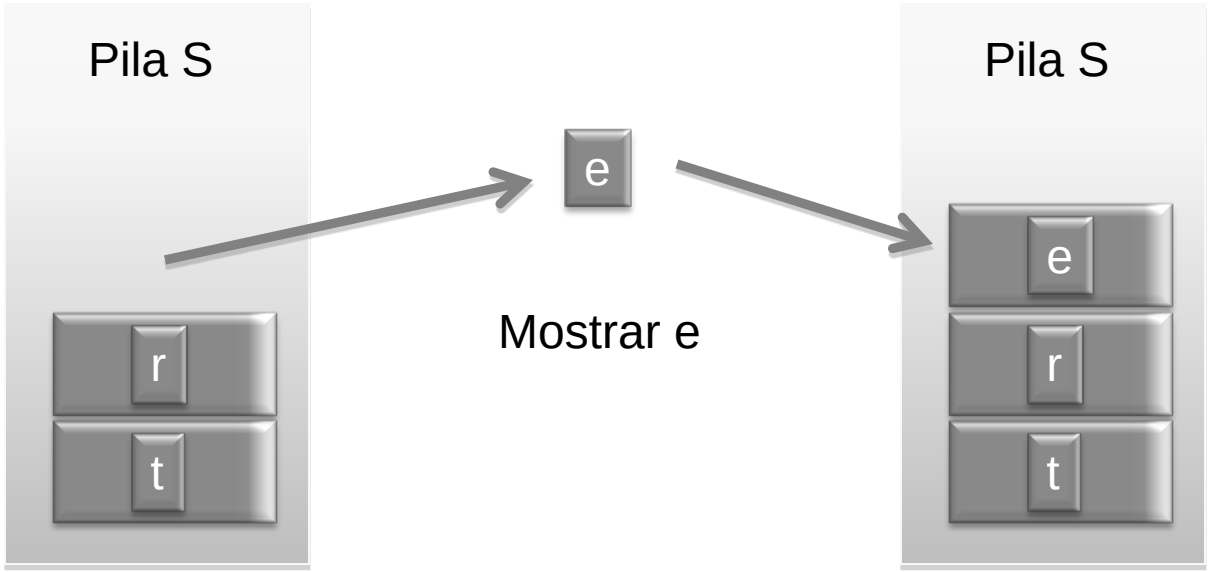
(1) La función devuelve un tipo entero al recibir la dirección de una variable de tipo estructura pila (struct stack *). Las líneas (4) y (5) son las mas importantes ya que se almacena el valor que ser devuelto y se decrementa el tope de la pila.

```
int pop(Stack *ps) {
  if(ps->top == -1)
    printf(“Empty stack\n");
  else {
    int t;
    t = ps->nodes[ps-> top];
    ps->top--;
    return t;
  }
}
```

La Operación Stacktop

Esta función debe devolver un número entero y dejar la pila sin cambio. Para esto:

- pop(&A)
- Mostrar el elemento A
- push(&A,elemento).



La Operación Stacktop

El siguiente segmento de código ilustra cómo se han usado las funciones antes creadas para implementar Stacktop, por supuesto que se pueden separar y crear una nueva función que haga lo mismo:

```
    ...
(1) case 4:{
(2)  if(not stackempty(&A)){
(3)  valor=pop(&A);
(4)  std::cout<<"La cima de la pila es: "<<valor<<"\n";
(5)  push(&A,valor);
(6)  } else
(7)  std::cout<<"La pila esta vacia";
(8)  break;
(9)  }
   ...
```

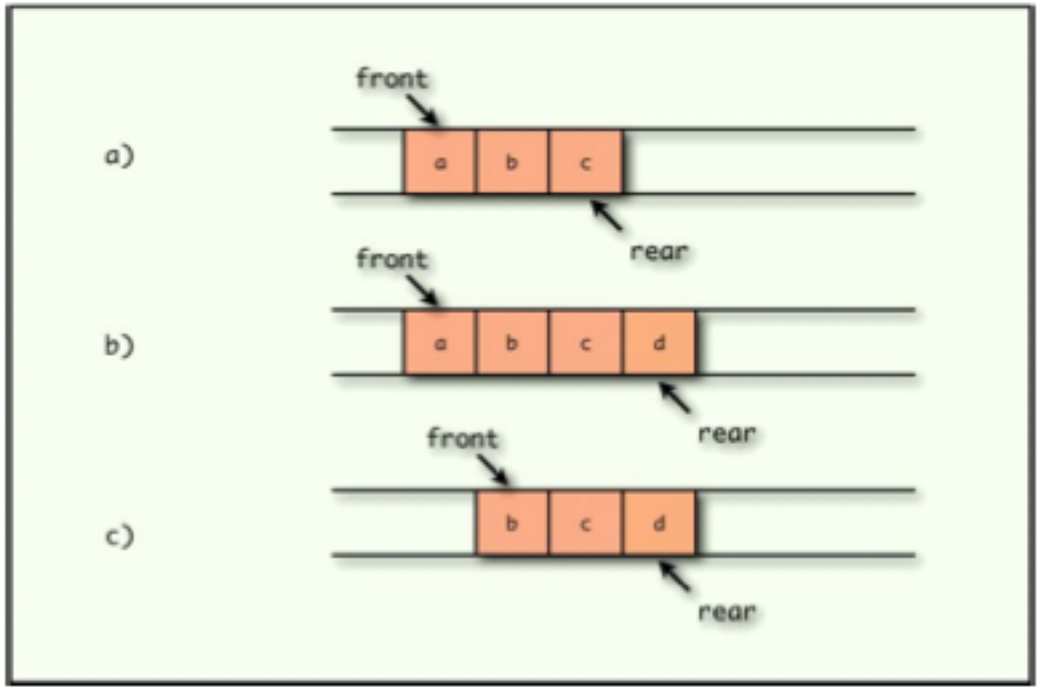
La Operación Stackempty

La operación stackempty se describe en el siguiente segmento de código:

```
(1) bool stackempty(struct stack *S){
(2)    bool valorDevuelto;
(3)    if(S->top== -1)
(4)          valorDevuelto=true;
(5)          else
(6)       valorDevuelto=false;
(7)    return valorDevuelto;
(8) }
```

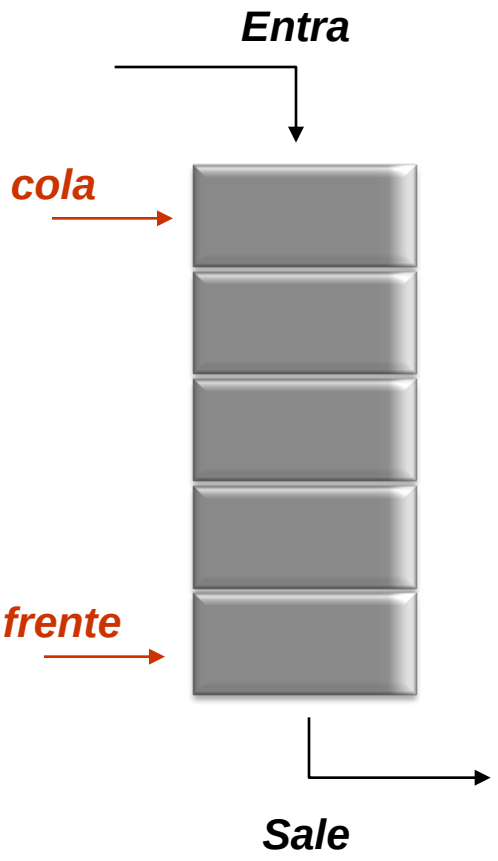
La estructura de datos cola

En una cola hay dos extremos, uno es llamado la parte delantera y el otro extremo se llama la parte trasera de la cola. En una cola, los elementos se retiran por la parte delantera y se agregan por la parte trasera.



La estructura de datos cola

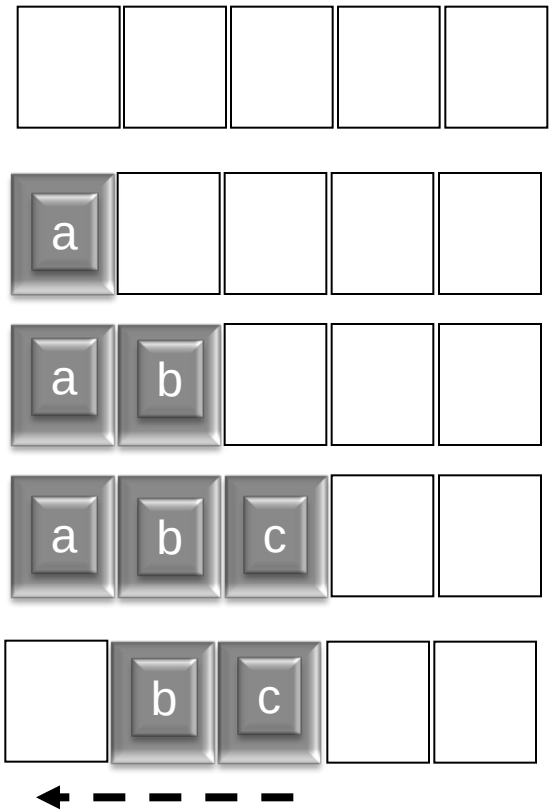
- Su nombre se deriva de la metáfora de una cola de personas en una taquilla.
- La inserción y extracción de elementos de la cola siguen el principio FIFO (*first-in-first-out*).
- El elemento con más tiempo en la cola es el que puede ser extraído.



Operaciones en una cola

- Las operaciones básicas de una cola son "*enqueue*" (meter) y "*dequeue*" (sacar)
  - enqueue: añade un nuevo elemento al final de la cola
  - dequeue*:* elimina (saca) el primer elemento de la cola
- Otras operaciones usualmente incluidas en el tipo abstracto COLA son:
  - **`isEmpty`** (estáVacia): verifica si la cola está vacía
  - **`isFull`** (estáLlena): verifica si la cola está llena

Ejemplo de operaciones en una cola



Cola vacía

enqueue(a)

enqueue(b)

enqueue(c)

dequeue()

Aplicaciones de las colas

- En general, operaciones en redes de computadoras.
  - Trabajos enviados a una impresora
  - Solicitudes a un servidor.
- Clientes solicitando ser atendidos por una telefonista.
- Mas aplicaciones de las estructuras tipo cola?

Implementación de la estructura cola

De manera similar a las pilas, las colas definen una estructura no estándar, de manera que se debe crear un nuevo tipo de dado, el tipo cola, que debe tener los siguientes elementos:

- Un arreglo de n elementos de algún tipo específico, puede incluso ser un tipo estándar o no.
- Un número que indica el elemento que está en la posición del frente de la cola.
- Un número que indica el elemento que está en la posición trasera de la cola.

Implementación de la estructura cola

Suponiendo que los elementos son números enteros, una idea para representar una cola en C/C++ es usar un arreglo para contener los elementos y emplear otras dos variables para representar la parte frontal y trasera de la cola.

```
#define maxQueue 100

struct cola{
int items[maxQueue];
int front;
int rear;
};
```

Implementación de la estructura cola

- Una cola es una colección ordenada de objetos.
- En C, los arreglos permiten almacenar colecciones ordenadas.
- Los arreglos tienen tamaño fijo.

```
#define QUEUESIZE 100
typedef struct queue{
     int front;
     int rear;
     int nodes[QUEUESIZE];
}Queue;

Queue *create_queuek
                    (Queue *ps) {
     ps = (Queue *)
            malloc(sizeof(Queue));
     ps->front = 0;
     ps->rear = -1;
     return ps;
}

main(){
   Queue *ps;
     ps=create_queue(ps);
}
```

Operaciones en colas

Suponiendo que no existiera la posibilidad de caer en un desbordamiento del arreglo, las operaciones enqueue (insert) y dequeue (remove) quedan**:

```
void insert(struct cola *C, int e){
C->items[++C->rear]=e;
}
```

```
int remove(struct cola *C){
return C->items[C->front++];
```

** Asumiendo un comportamiento tipo Buffer.

Operaciones en colas

ENQUEUE

```
void enqueue(Queue* ps,int i) {
    if (ps->rear == (QUEUESIZE -1))
        printf("Full queue\n");
    else {
        ps->rear++;
        ps->nodes[ps->rear] = i;
    }
}
```

DEQUEUE

```
int dequeue (Queue *ps) {
  if(ps->front == ps->rear + 1)
     printf("Empty queue\n");
  else{
   int t= ps->nodes[ps-> front];
   ps->front++;**
   return t;
  }
}
```

** Asumiendo un comportamiento tipo Buffer.

Operaciones en colas

La operación empty:

```
bool empty(struct cola *C){
if(C->front>C->rear)
return true;
else
return false;
}
```

Colas con prioridad

Una cola con prioridad es una estructura de datos en la que se ordenan los datos almacenados de acuerdo a un criterio de prioridad. Hay dos tipos de colas de prioridad:

- Las colas de prioridad con ordenamiento descendente.
- Las colas de prioridad con ordenamiento ascendente.

Colas con prioridad

Si CPA es una cola de prioridad ascendente, la operación **insert(CPA,x)** inserta el elemento x en la cola CPA; y la operación **x=minRemove(CPA)** asigna a x el valor del elemento menor (de su prioridad) y lo remueve de la cola.

En las colas de prioridad descendente las operaciones aplicables son **insert(CPD,x)** y **x=maxRemove(CPD)**, cuando CPD es una cola de prioridad descendente y x es un elemento.

La operación **empty(C)** se aplica a cualquier tipo de cola y determina si una cola de prioridad está vacía. La operación de borrar se aplica solamente si la pila no está vacía.

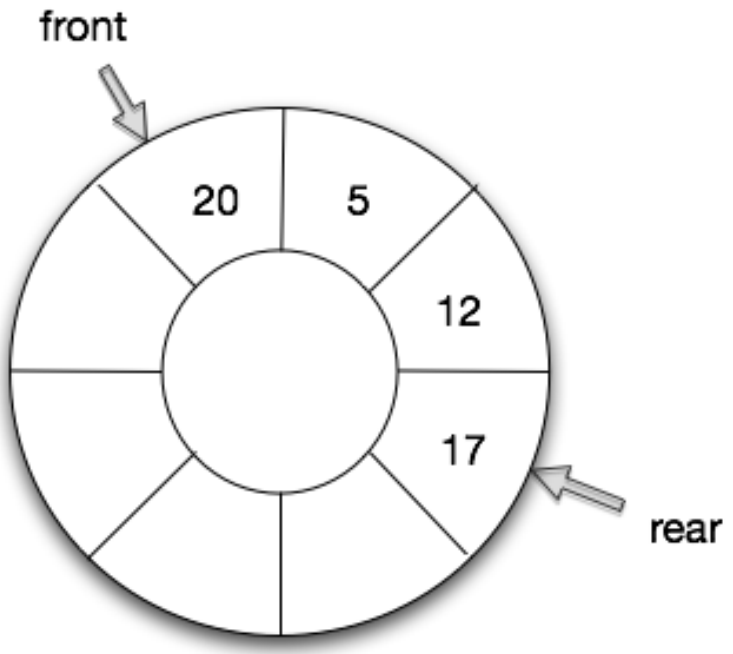
Colas con prioridad

Cuando se requiere eliminar un dato de una cola de prioridad se necesitan verificar todos los elementos almacenados para saber cuál es el menor (o el mayor).

Esto conlleva algunos problemas, el principal es que el tiempo necesario para eliminar un elemento puede crecer tanto como elementos tenga la cola.

Colas Circulares

- El objetivo de una cola circular es aprovechar al máximo el espacio del arreglo.
- La idea es insertar elementos en las localidades previamente desocupadas.
- La implementación tradicional considera dejar un espacio entre el frente y la cola.



Colas Circulares

Si se eliminan componentes quedan espacios disponibles en las primeras posiciones del arreglo.

Cuando se encolan elementos disminuye el espacio para agregar nuevos elementos en la zona alta del arreglo.

Este buffer se puede implementar aplicando aritmética modular, si el anillo tiene *N* posiciones, la operación: *rear* = (*rear*+1) Mod *N*, mantiene el valor de la variable cola entre 0 y *N*-1.

Colas Circulares

Operación similar puede efectuarse para la variable *head* cuando deba ser incrementada en uno.

La variable *rear* puede variar entre 0 y N-1.

Si *rear* tiene valor N-1, al ser incrementada en uno (módulo N), tomará valor cero.

También se agrega una variable N con el numero de elementos encolados para poder distinguir entre la cola vacía y llena.