

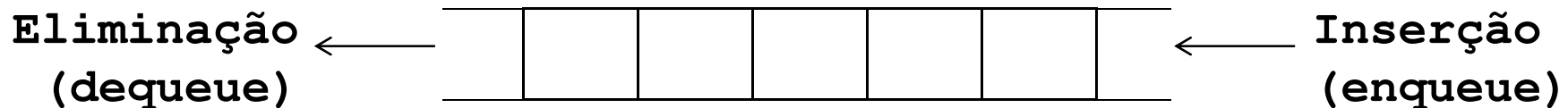
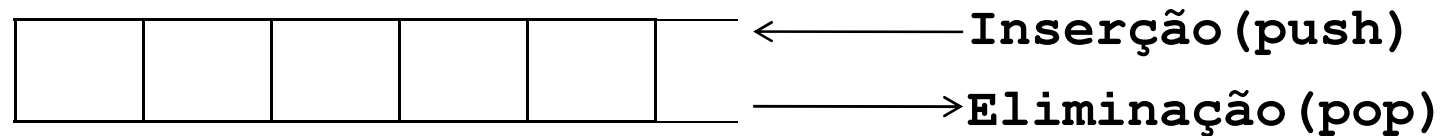
CES-11

- Pilhas
 - Definição
 - Operações
- Filas
 - Definição
 - Operações
- *Deque*s
 - Definição
 - Operações



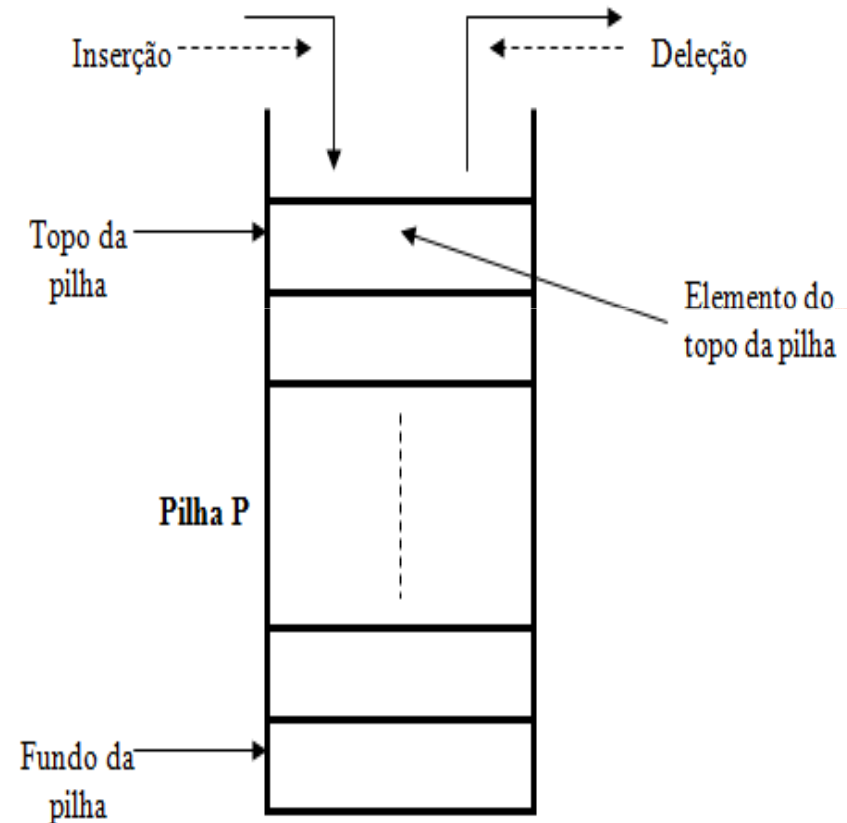
PILHAS, FILAS E *DEQUES*

- As listas lineares admitem inserção e eliminação em qualquer posição.
- Pilhas, filas e *deques* (filas com duplo-fim) só admitem acessos em suas extremidades:



PILHAS (STACKS)

- *Pilha* é uma estrutura linear que permite acesso em somente uma extremidade.
- Se n elementos são introduzidos na pilha e depois retirados, o primeiro a entrar será o último a ser retirados
- Por essa razão, a pilha é chamada de estrutura
- *FILO* (*first in / last out*).



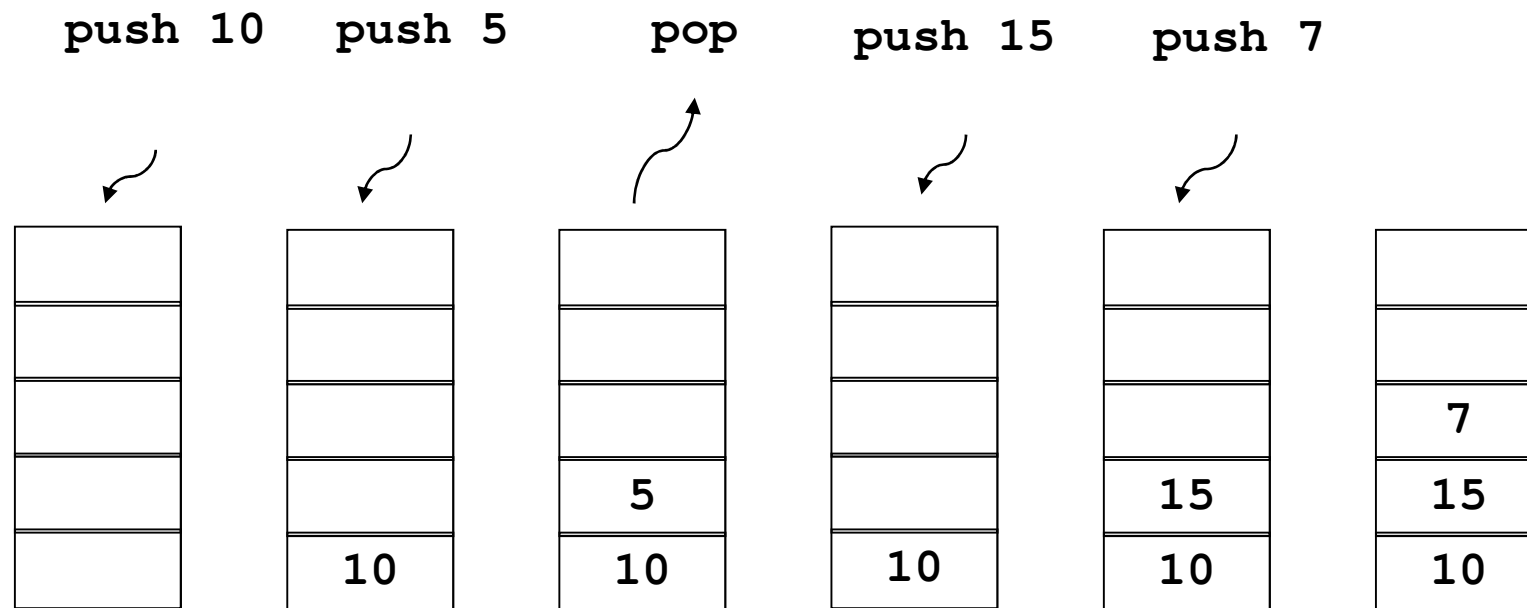
PILHAS (*STACKS*)

○ Operações:

- `push(x)`: insere `x` no topo
- `pop()`: retira o elemento topo
- `top()`: retorna o topo sem desempilhá-lo
- `size()`: retorna o tamanho atual da pilha
- `isEmpty()`: verifica se a pilha está vazia



EXEMPLOS DE OPERAÇÕES COM PILHAS



UMA APLICAÇÃO TÍPICA

- Pilhas podem ser utilizadas, por exemplo, para verificar o emparelhamento de delimitadores.

- Teste 1: (3+[2*4+{8-3}]) //15 caracteres

- Teste 2: (3+[2*4+{8-3}]) //15 caracteres

```
int verifica(char *v) {
    for(int i=0;v[i]!='\0';i++) {
        if(v[i]=='(' || v[i]=='[' || v[i]=='{')
            push(v[i]);
        else if(v[i]==')' || v[i]==']' || v[i]=='}') {
            char x = pop();
            //se delimitadores não "batem" retorna erro
            if((x=='(' && v[i]!=')') ||
               (x=='[' && v[i]!='']') || (x=='{' && v[i]!='}'))
                return ERRO;
        }
    }
    if(isEmpty()) return OK;
    else return ERRO;
}
```



IMPLEMENTAÇÃO COM VETOR



```
struct Pilha {  
    int S[N]; int t; };
```

```
int size(Pilha p) {  
    return p.t+1; }
```

```
bool isEmpty() {  
    return (p.t<0); }
```

```
int top(Pilha p) {  
    if (isEmpty())  
        return Erro;  
    return p.S[p.t]; }
```

```
int push(int x,Pilha* p) {  
    if (size() == N)  
        return Erro;  
    p->S[++(p->t)]=x; }
```

```
int pop(Pilha* p) {  
    if (isEmpty())  
        return Erro;  
    return p->S[(p->t)--];  
}
```



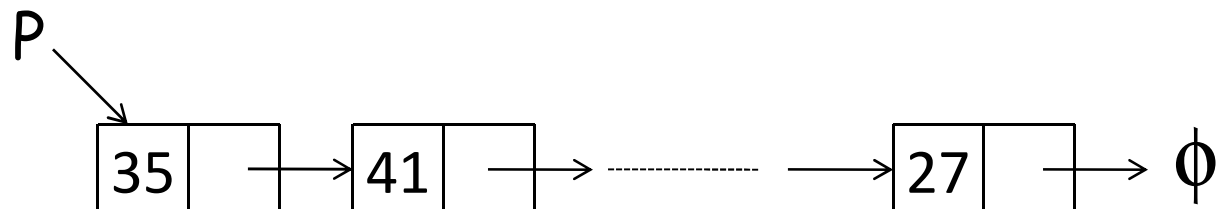
EFICIÊNCIA DESSA IMPLEMENTAÇÃO

- Na implementação com vetor, todas as operações anteriores podem ser executadas em tempo $O(1)$.



IMPLEMENTAÇÃO COM NÓS ENCADEADOS

- Os elementos da pilha podem estar conectados através de uma cadeia de nós:



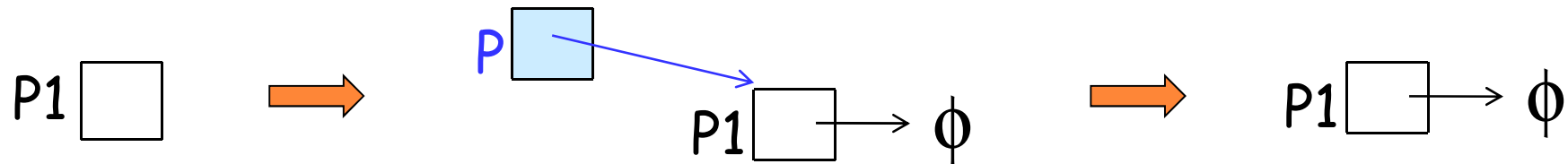
```
struct node {int elem; node *prox;};  
typedef node *pilha;  
pilha P;
```



IMPLEMENTAÇÃO COM NÓS ENCADEADOSOS

- `inicPilha (&P1)` : inicializa a pilha P1

```
void inicPilha (pilha *P) {  
    *P = NULL;  
}
```

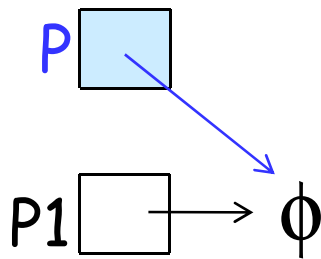


IMPLEMENTAÇÃO COM NÓS ENCADEADOSOS

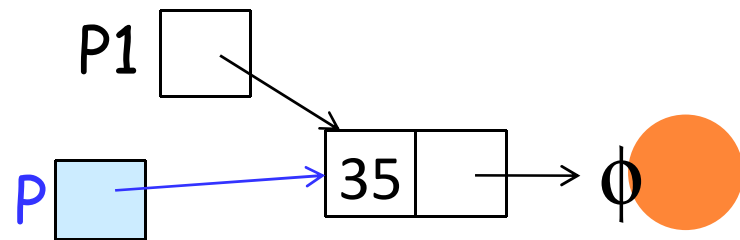
- **isEmpty (P1)** : verifica se a pilha P1 está vazia

```
bool isEmpty (pilha P) {  
    if (P == NULL) return TRUE;  
    else return FALSE;  
}
```

Pilha vazia



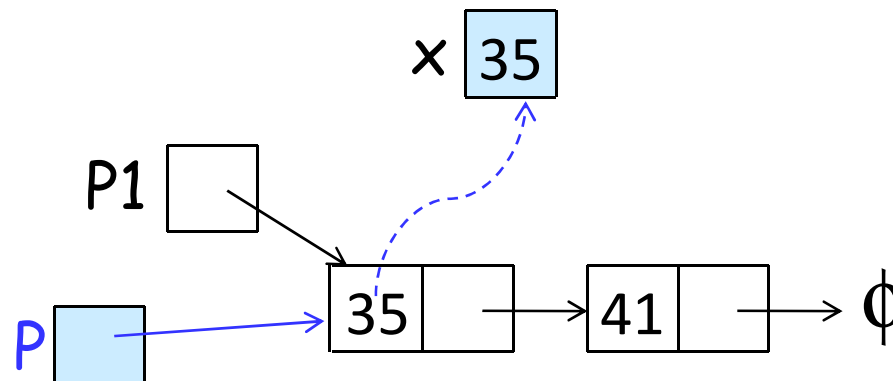
Pilha não vazia



IMPLEMENTAÇÃO COM NÓS ENCADEADOSOS

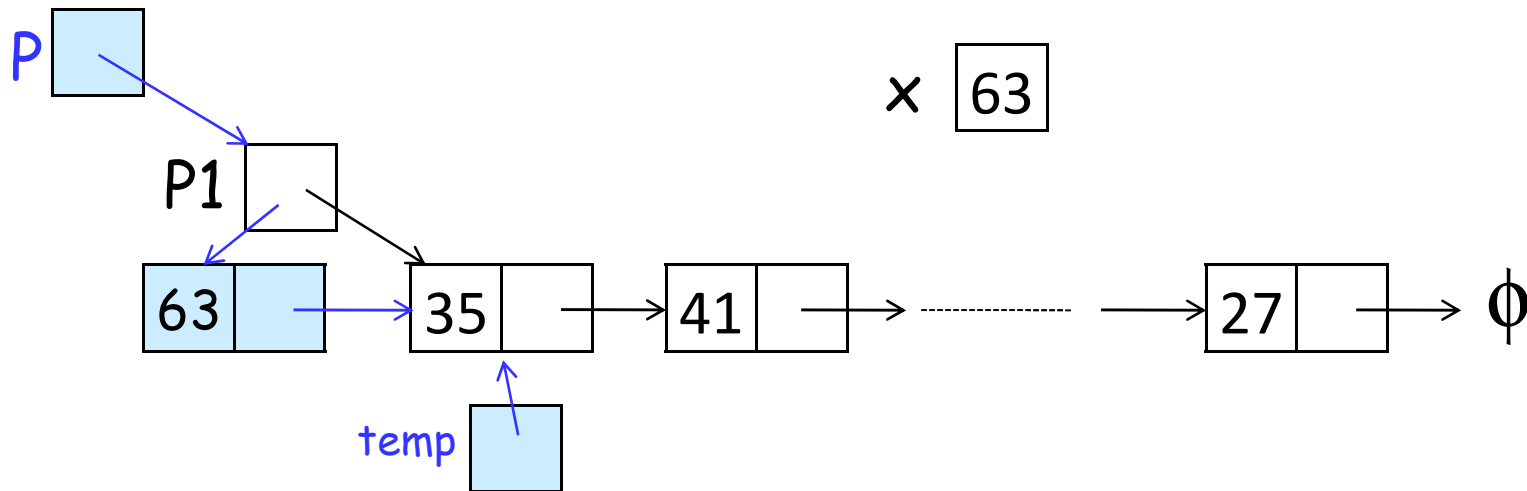
- **x = top(P1)**: x recebe o topo da pilha P1

```
int top (pilha P) {  
    if (!isEmpty (P))  
        return P->elem;  
    else  
        Erro ("Pilha vazia");  
}
```



IMPLEMENTAÇÃO COM NÓS ENCADEADOS

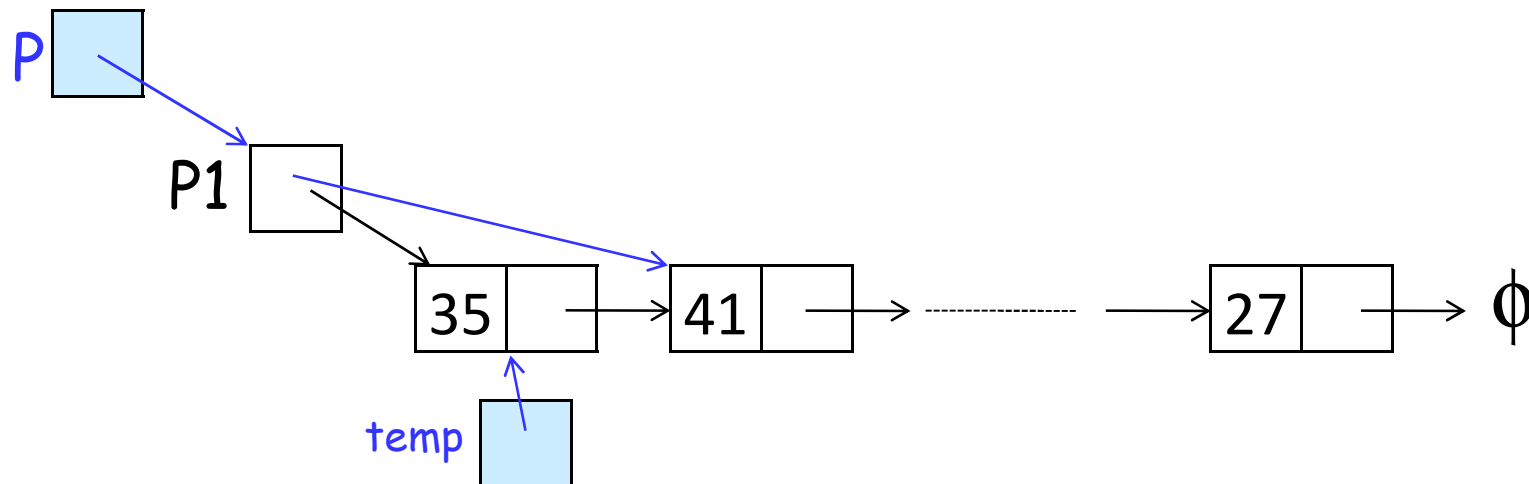
- **push (x, &P1)** : empilha o conteúdo de x na pilha P1



```
void push (char x, pilha *P) {  
    node *temp;  
    temp = *P;  
    *P = (node *) malloc (sizeof (node));  
    (*P)->elem = x;  
    (*P)->prox = temp;  
}
```

IMPLEMENTAÇÃO COM NÓS ENCADEADOSOS

- **pop (&P1):** desempilha o topo da pilha P1



```
char pop (pilha *P) {  
    node *temp; char c;  
    if (!isEmpty (*P)) {  
        temp = *P;  
        *P = (*P)->prox;  
        c=temp->elem;  
        free (temp); return c; }  
    else Erro ("Pilha vazia"); }
```



EFICIÊNCIA DESSA IMPLEMENTAÇÃO

- Na implementação com nós encadeados, todas as operações anteriores podem ser executadas em tempo $O(1)$.
- Qual a vantagem em relação à implementação com vetor?
 - A memória é alocada gradativamente, apenas quando necessário.



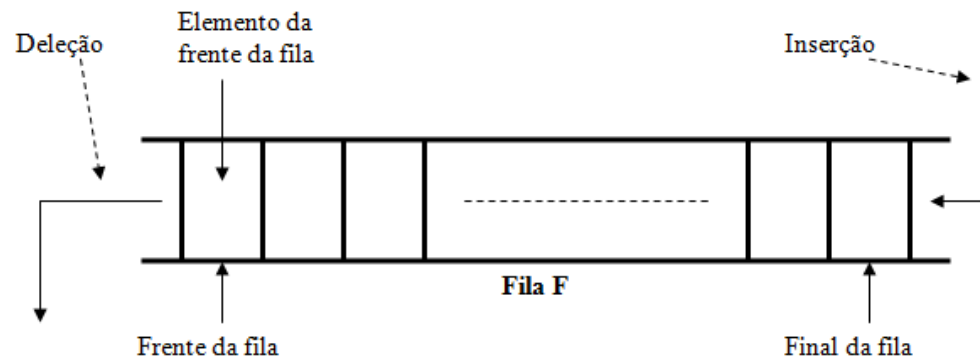
CES-11

- Pilhas
 - Definição
 - Operações
- Filas
 - Definição
 - Operações
- *Deque*s
 - Definição
 - Operações



FILAS (*QUEUES*)

- *Filas* são simples sequências de espera: crescem através do acréscimo de novos elementos no final e diminuem com a saída dos elementos da frente.
- Os elementos são acrescentados em uma extremidade e removidos da outra.
- Em relação à pilha, a principal diferença é que a fila é uma estrutura *FIFO* (*first in / first out*).



FILAS

○ Operações:

- `size()`: retorna o tamanho atual da fila
- `isEmpty()`: verifica se a fila está vazia
- `first()`: retorna o primeiro elemento da fila sem removê-lo
- `dequeue()`: retira o primeiro elemento da fila
- `enqueue(x)`: coloca x no final da fila



EXEMPLOS DE OPERAÇÕES COM FILAS

enqueue 10 enqueue 5



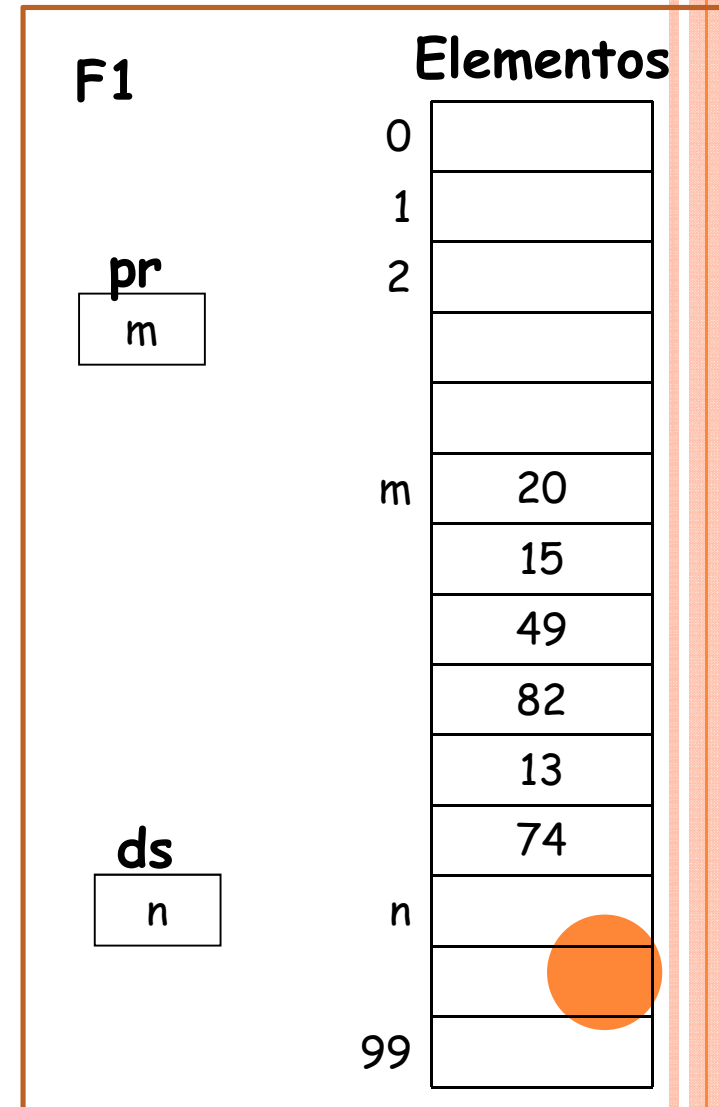
IMPLEMENTAÇÃO COM VETOR CIRCULAR

- O vetor é utilizado de maneira circular

```
const int max = 100;  
typedef int vetor[max];  
struct fila {  
    vetor Elementos;  
    int pr, ds;  
};  
fila F1, F2, F3;
```

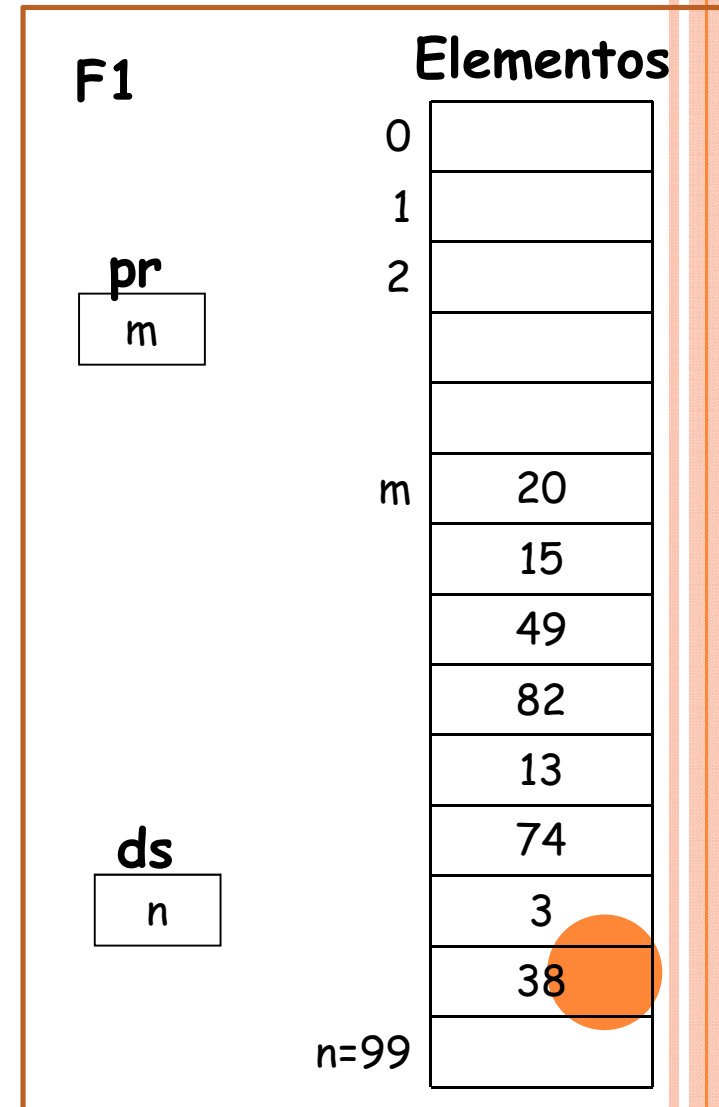
Próxima posição disponível

Posição do primeiro elemento



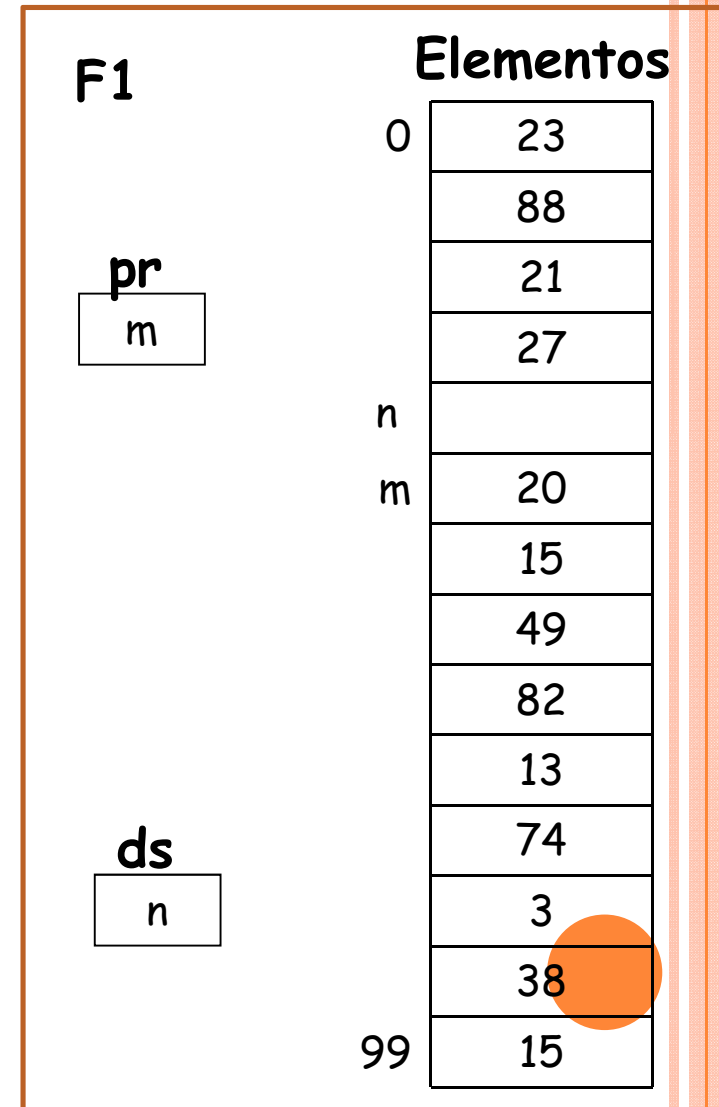
IMPLEMENTAÇÃO COM VETOR CIRCULAR

- É circular: quando $ds=99$, uma inserção levará ds para o índice 0 (zero).
- Desse modo, não haverá necessidade de deslocar todos os elementos para cima...
- Se $pr=ds$ a fila está vazia



IMPLEMENTAÇÃO COM VETOR CIRCULAR

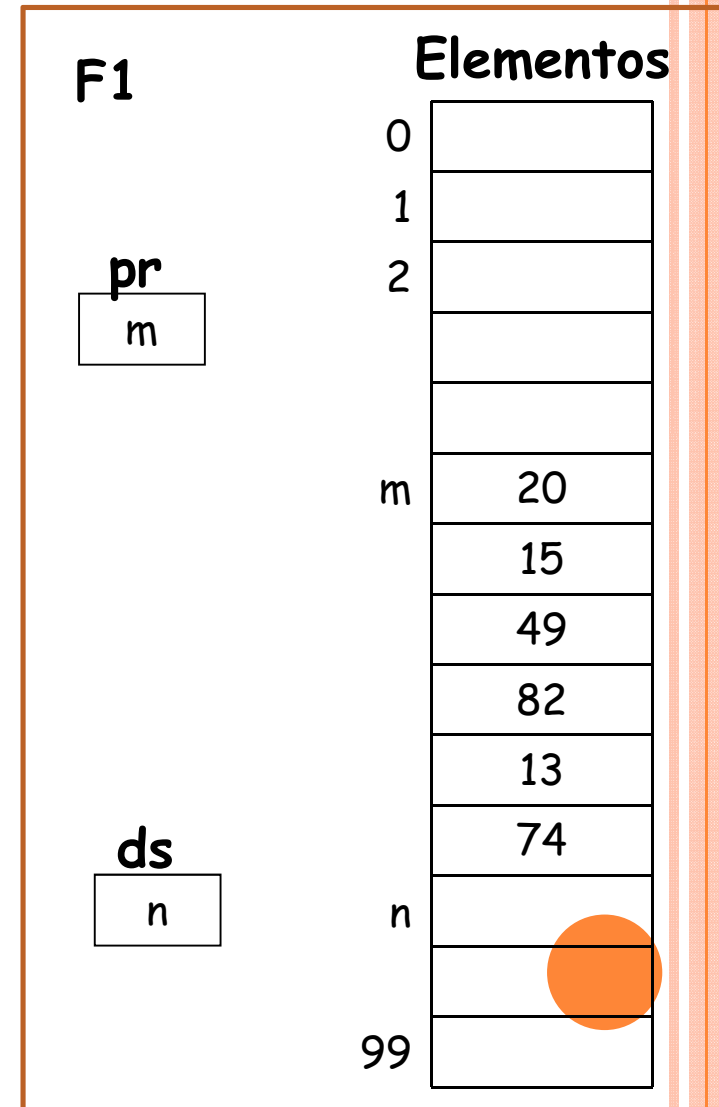
- Quando $\text{max}=100$, é possível admitir 100 elementos na fila?
 - Não: fila cheia seria confundida com fila vazia...
- Caso geral desta implementação: em uma fila com max posições, podemos ter no máximo $\text{max}-1$ elementos.



IMPLEMENTAÇÃO COM VETOR CIRCULAR

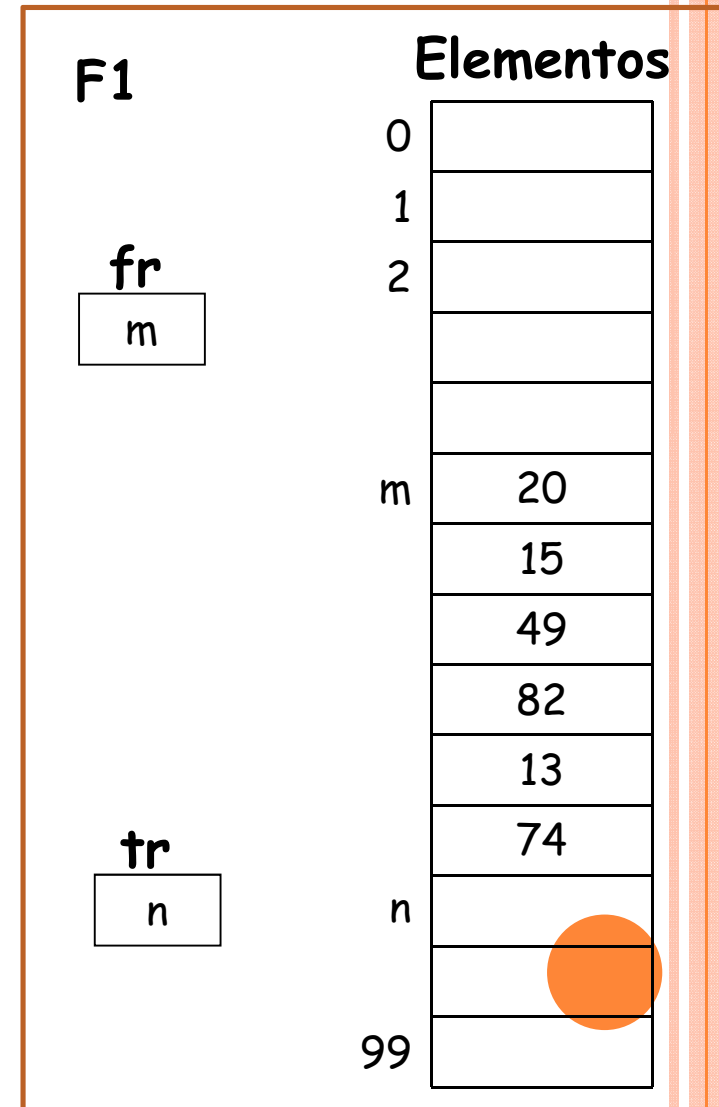
```
void inicFila (fila *F) {  
    F->pr= 0;  
    F->ds = 0;  
}
```

```
int prox (int i)  {  
    return (i+1) % max;  
}
```



IMPLEMENTAÇÃO COM VETOR CIRCULAR

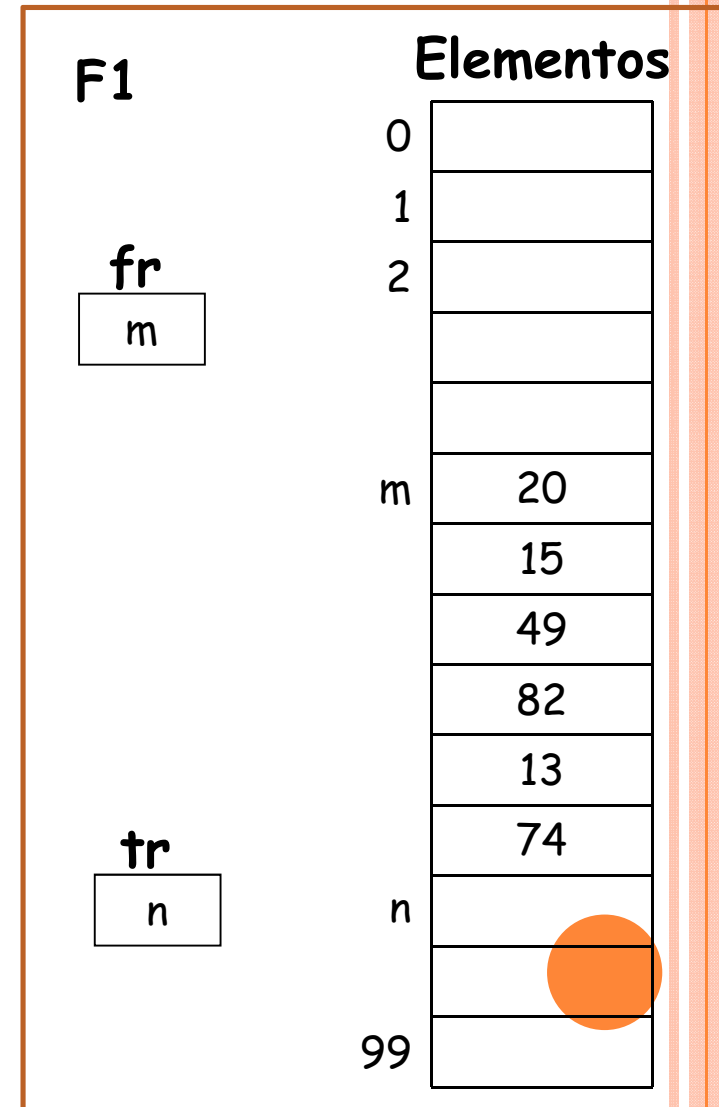
```
bool isEmpty (fila *F) {  
    if (F->ds == F->pr)  
        return true;  
    else  
        return false;  
}  
  
int first (fila *F) {  
    if (isEmpty (F))  
        Erro ("Fila vazia");  
    else  
        return F->Elementos[F->pr];  
}
```



IMPLEMENTAÇÃO COM VETOR CIRCULAR

```
void enqueue (int x, fila *F)
{
    if ( prox(F->ds) == F->pr )
        Erro ("Fila cheia");
    else {
        F->Elementos[F->ds] = x;
        F->ds = prox(F->ds);
    }
}
```

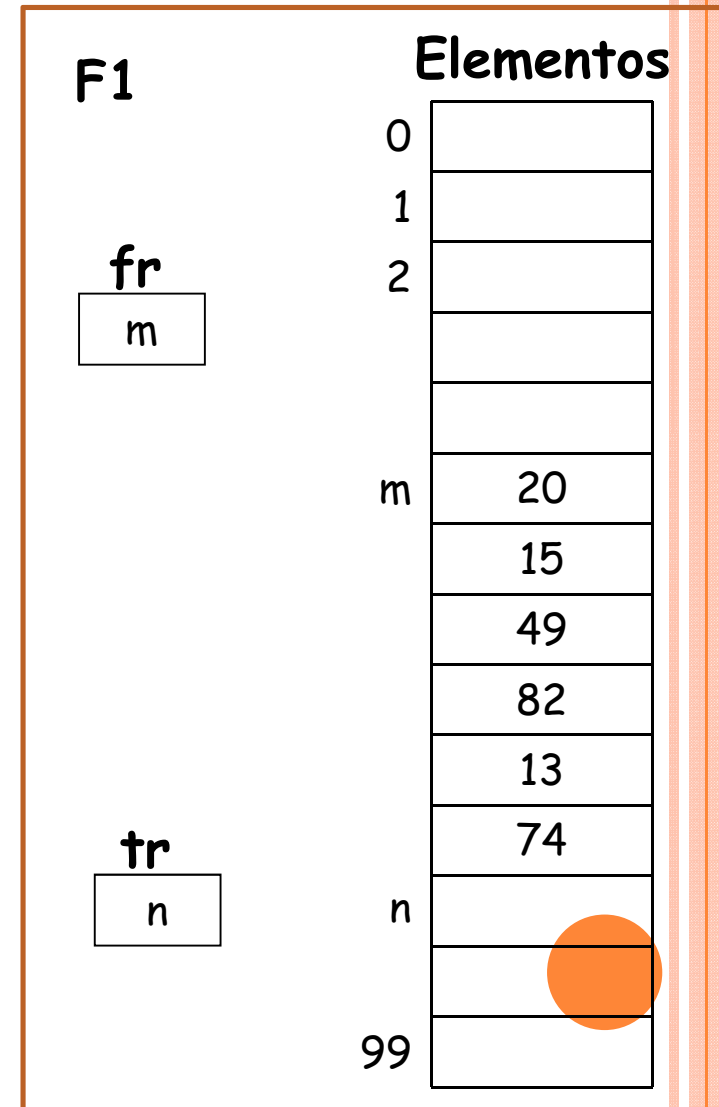
```
void dequeue (fila *F)
    if (isEmpty (F))
        Erro ("Fila vazia");
    else
        F->pr = prox(F->pr);
}
```



IMPLEMENTAÇÃO COM VETOR CIRCULAR

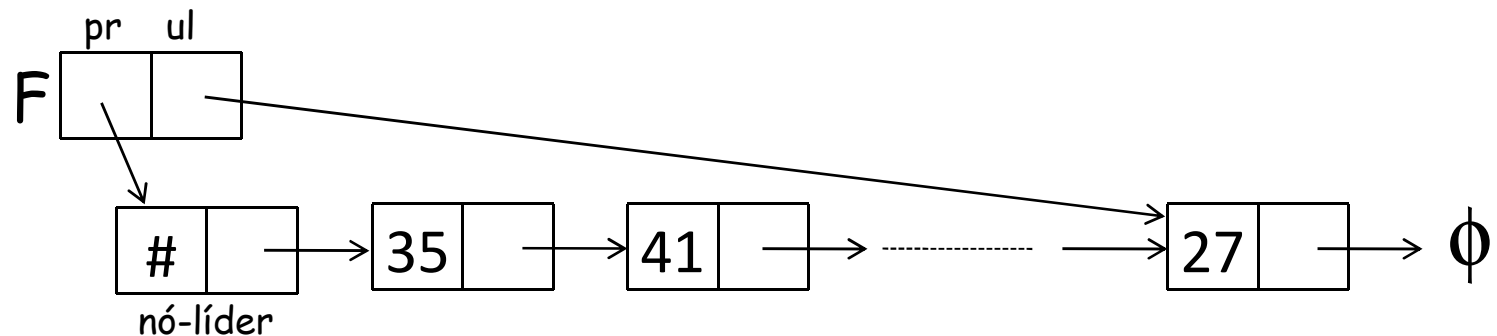
```
int num(fila *F)
{
    int a=F->ds - F->pr;
    if ( a <0)
        return (max+F->ds-F->pr);
    else {
        return a;
    }
}
```

```
void dequeue (fila *F)
    if (isEmpty (F))
        Erro ("Fila vazia");
    else
        F->pr = prox(F->pr);
}
```



IMPLEMENTAÇÃO COM NÓS ENCADEADOSOS

- Os elementos da fila podem estar conectados através de uma cadeia de nós:



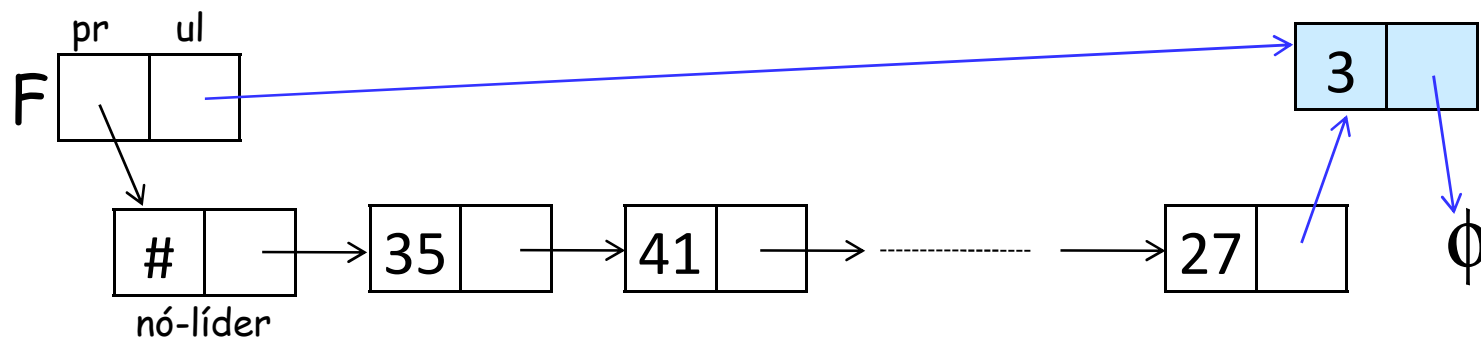
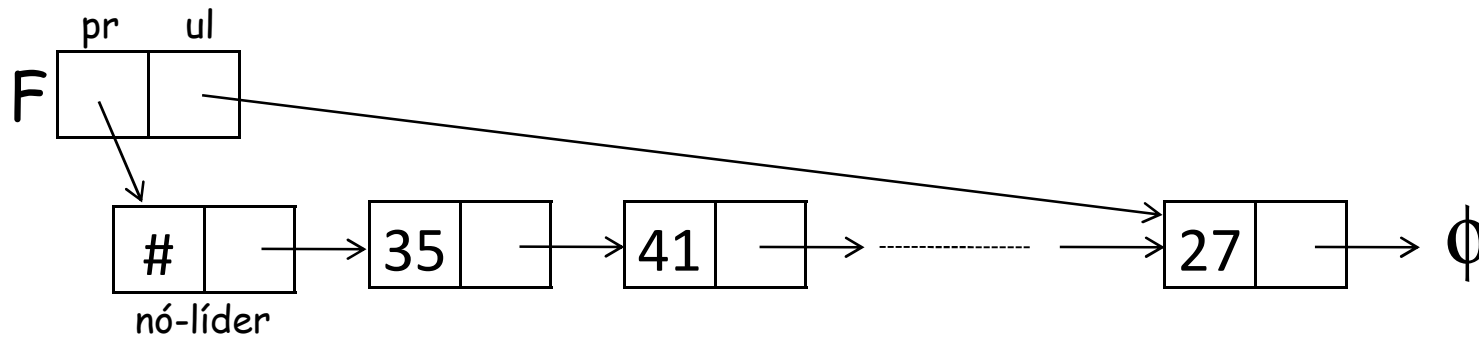
```
struct node {int elem; node *prox;};  
struct fila {node *pr, *ul;};  
fila F;
```



IMPLEMENTAÇÃO COM NÓS ENCADEADOSOS

○ enqueue (x, F) :

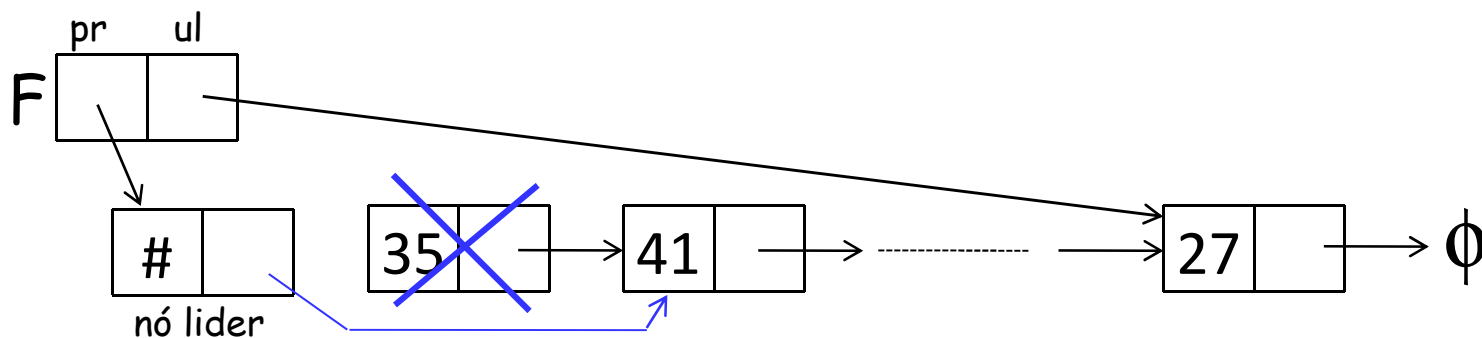
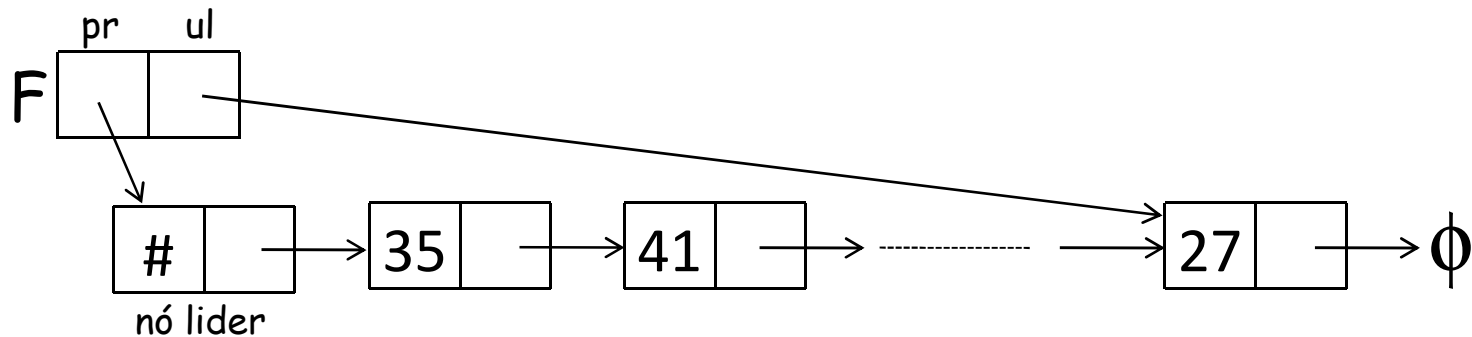
- Inserir um nó com valor x após o último elemento
- Atualizar o ponteiro $F.tr$



IMPLEMENTAÇÃO COM NÓS ENCADEADOSOS

○ dequeue (F) :

- Apagar o primeiro nó da cadeia
- Se esse nó fosse o único, F.tr deverá apontar para o nó-líder



IMPLEMENTAÇÃO COM NÓS ENCADEADOS

- Primeiro elemento da fila F
 - `F.pr->prox->elem`
- Verificação de fila F vazia
 - `F.pr == F.ul` ou
 - `F.pr->prox == NULL`
- Esvaziamento da fila F
 - Liberar os nós de F, exceto o líder
 - `F.pr->prox = NULL;`
 - `F.ul = F.pr`
- Exercício
 - Programar as operações enqueue e dequeue para fila implementada com nós encadeados



CES-11

- Pilhas
 - Definição
 - Operações
- Filas
 - Definição
 - Operações
- *Deque*
 - *Definição*
 - Operações



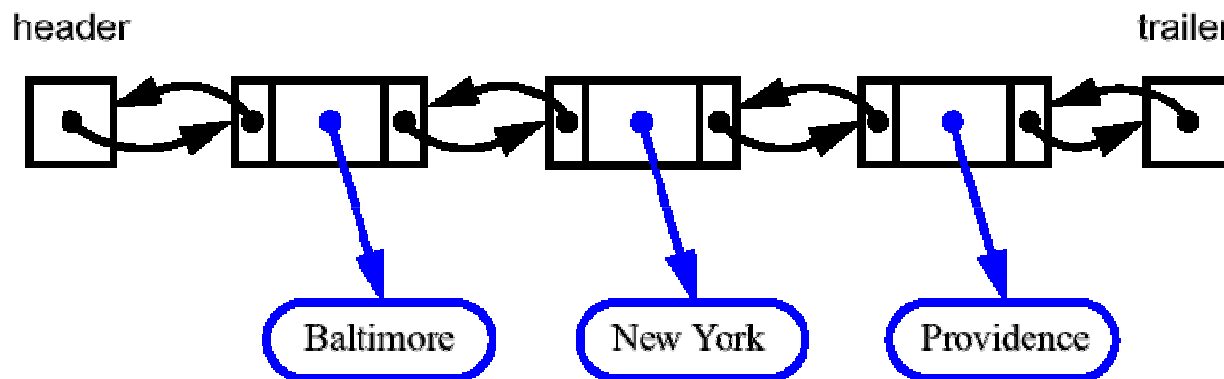
DEQUES (FILAS COM DUPLO-FIM)

- Filas com duplo-fim (*doubled-ended queue*) permitem inserção e remoção, em tempo constante, tanto no início como no fim.
- Uma *deque* pode simular tanto as operações de uma pilha como as de uma fila.
- Operações:
 - insertFirst(x): insere x no início
 - insertLast(x): insere x no final
 - removeFirst(): remove o primeiro elemento
 - removeLast(): remove o último elemento
 - first(): retorna o primeiro elemento
 - last(): retorna o último elemento
 - size(): retorna o tamanho atual
 - isEmpty(): verifica se está vazia



IMPLEMENTAÇÃO COM ENCADEAMENTO DUPLO

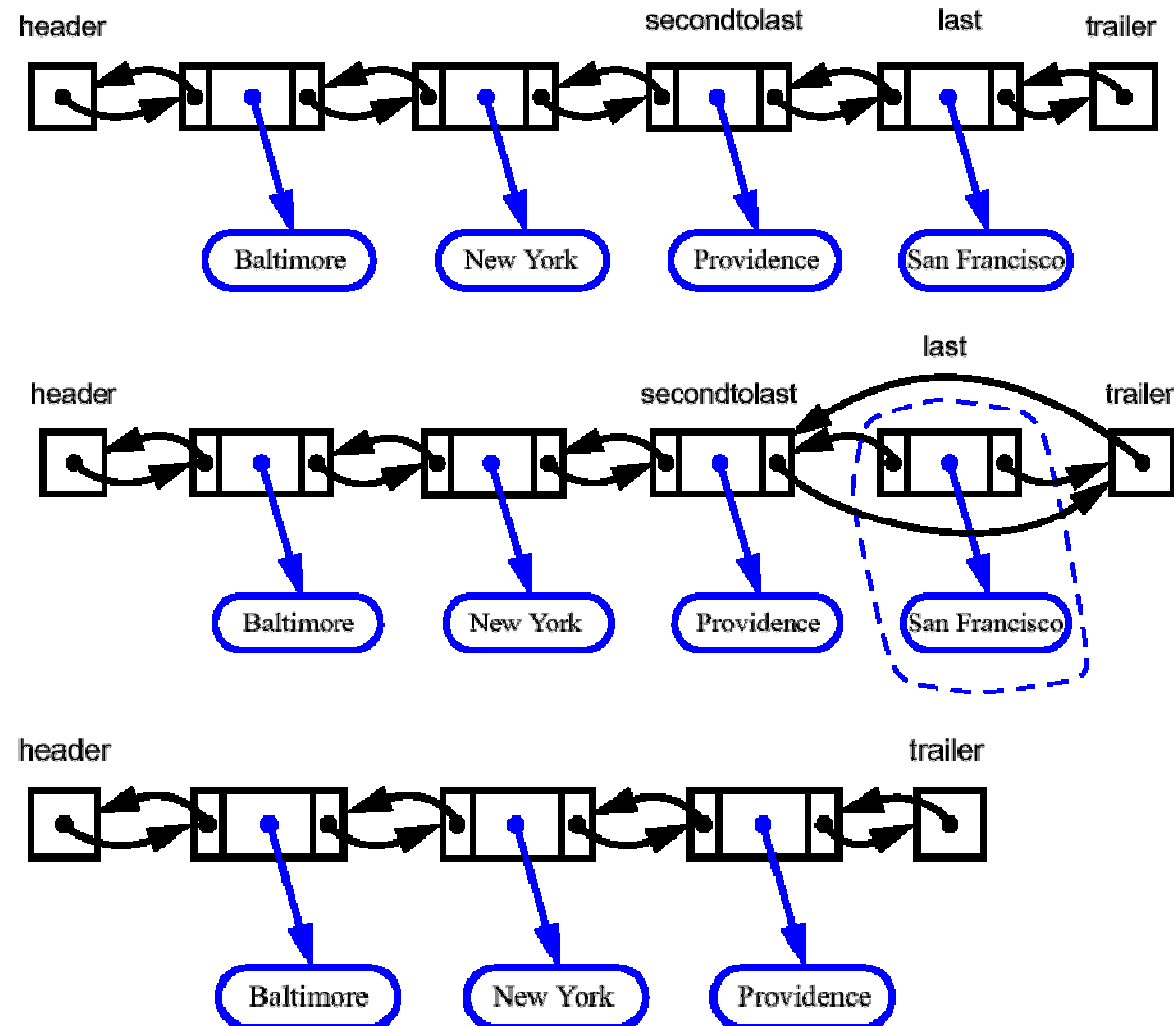
- Podemos implementar uma *deque* através de uma lista duplamente ligada:



- Cada nó tem dois ponteiros: para o próximo e para o anterior.
- *header* e *trailer* são nós especiais, chamados de sentinelas. Não são necessários, mas facilitam a codificação.

IMPLEMENTAÇÃO COM ENCADEAMENTO DUPLO

- Remoção do último elemento:



PERGUNTAS

- Seria possível implementar a deque através de vetor?
- Qual o grau de complexidade $O(?)$ do enqueue e dequeue em uma estrutura de dados em fila implementada com vetor? E se for implementada com nós encadeados?

