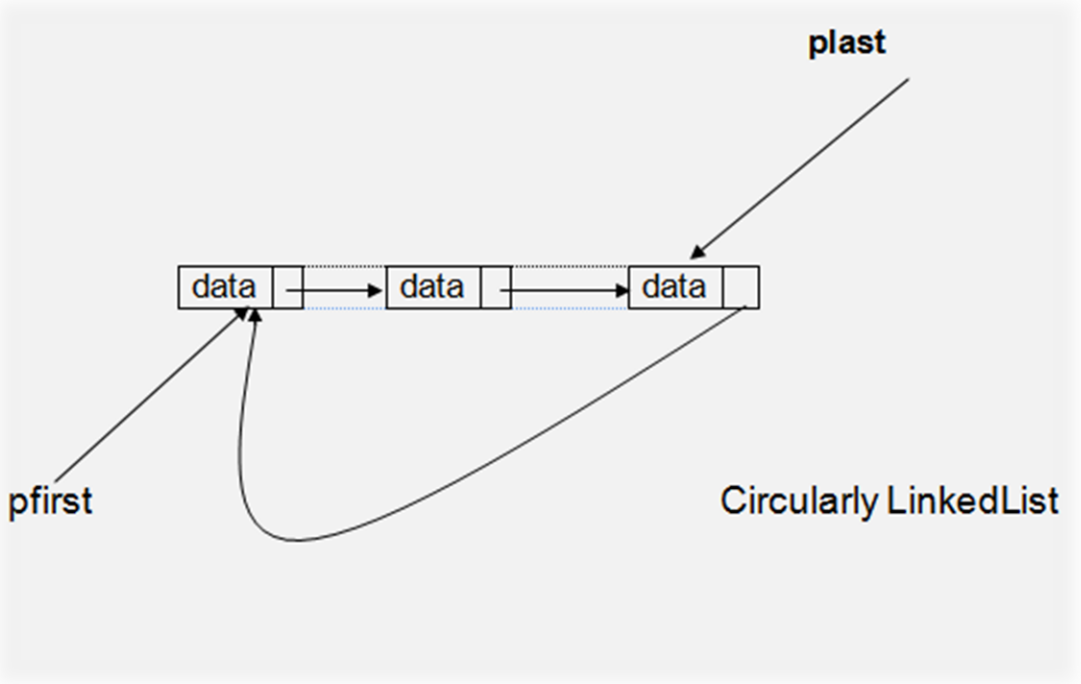
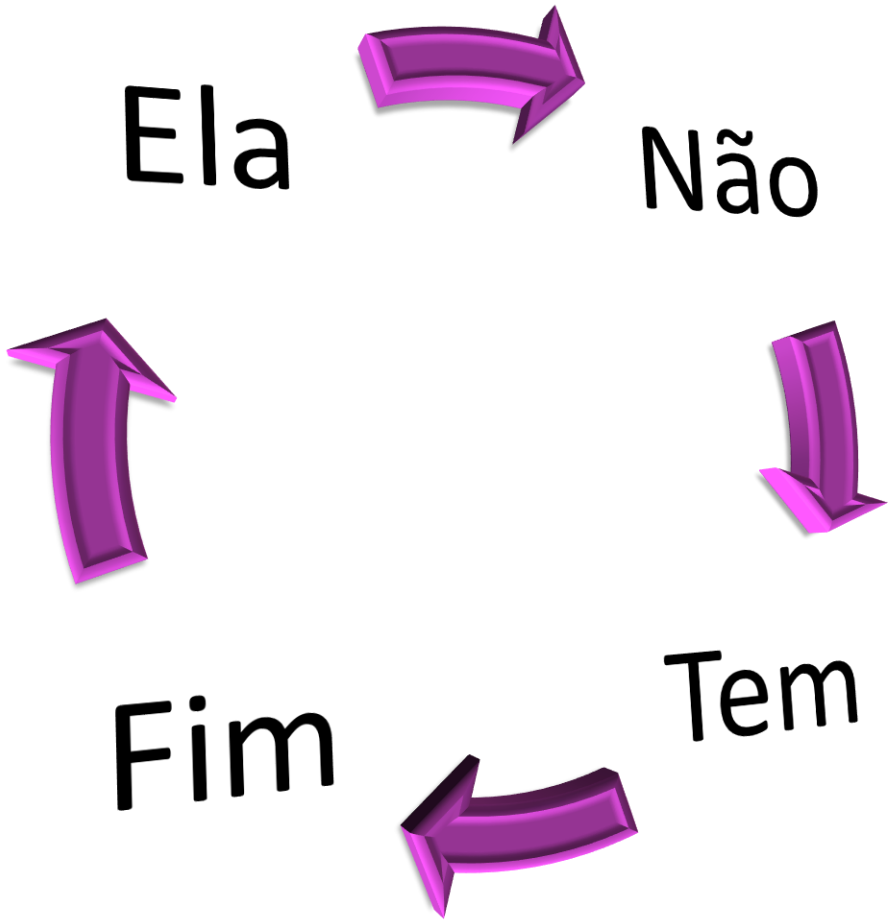
Listas Circulares



Característica adicional para o deslocamento na lista:



LISTAS CIRCULARES

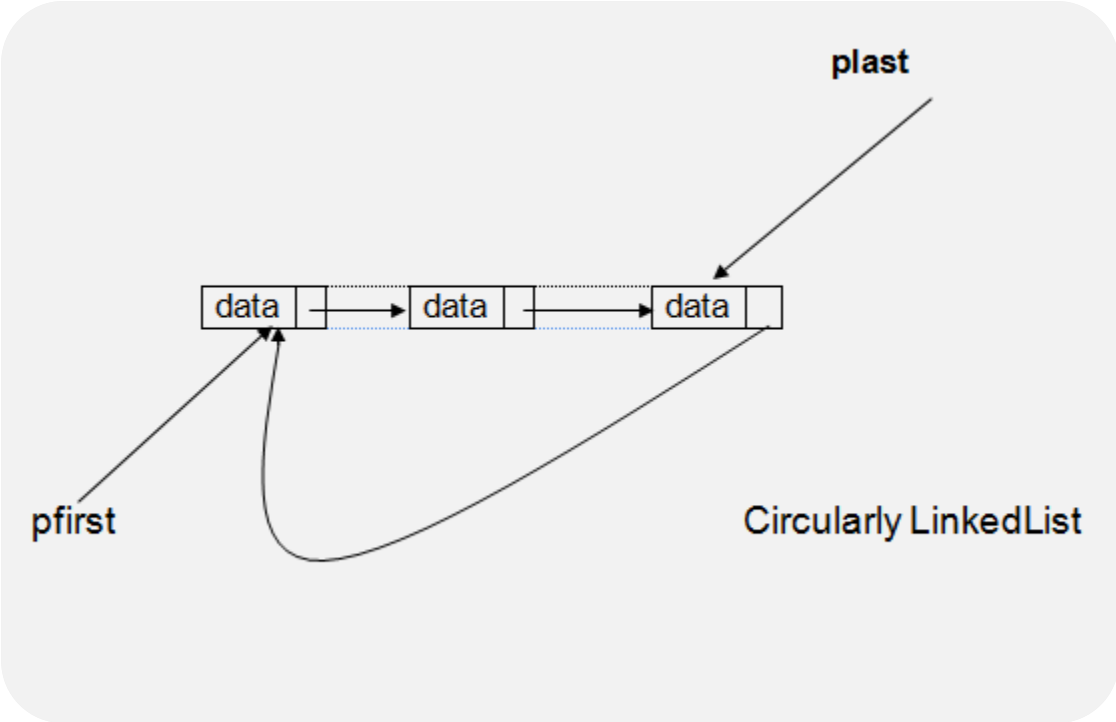

Para tornar a lista interminável, o ponteiro **seguinte** do último elemento apontará para o **primeiro elemento** da lista, em vez do valor NULL
Nas listas circulares, nunca chegaremos a uma posição a partir da qual não poderemos mais nos mover.
Em suma, trata-se de uma rotação.
Chegando ao último elemento, o deslocamento vai recomeçar no primeiro elemento

Na lista encadeada clássica, o ponteiro de **próximo** do último elemento aponta para NULL.

Na lista circular, o **último** elemento aponta para o **primeiro.**

LISTAS CIRCULARES

O último item da Lista aponta para o primeiro item da Lista. Ao permitir que o último item aponte para o primeiro item, todos os itens da lista são vinculados Circularmente.



Vamos construir uma lista circular que tenha dois ponteiros :

- um primeiro (pfirst) aponta para o primeiro item da lista
- outro (plast) aponta para o último item da lista

Escopo para a criação de uma lista circular que guarda inteiros:

```
struct NO
{
    int valor;
    NO *proximo;
};

NO *start;
```

Duas partes: dados e ponteiro.

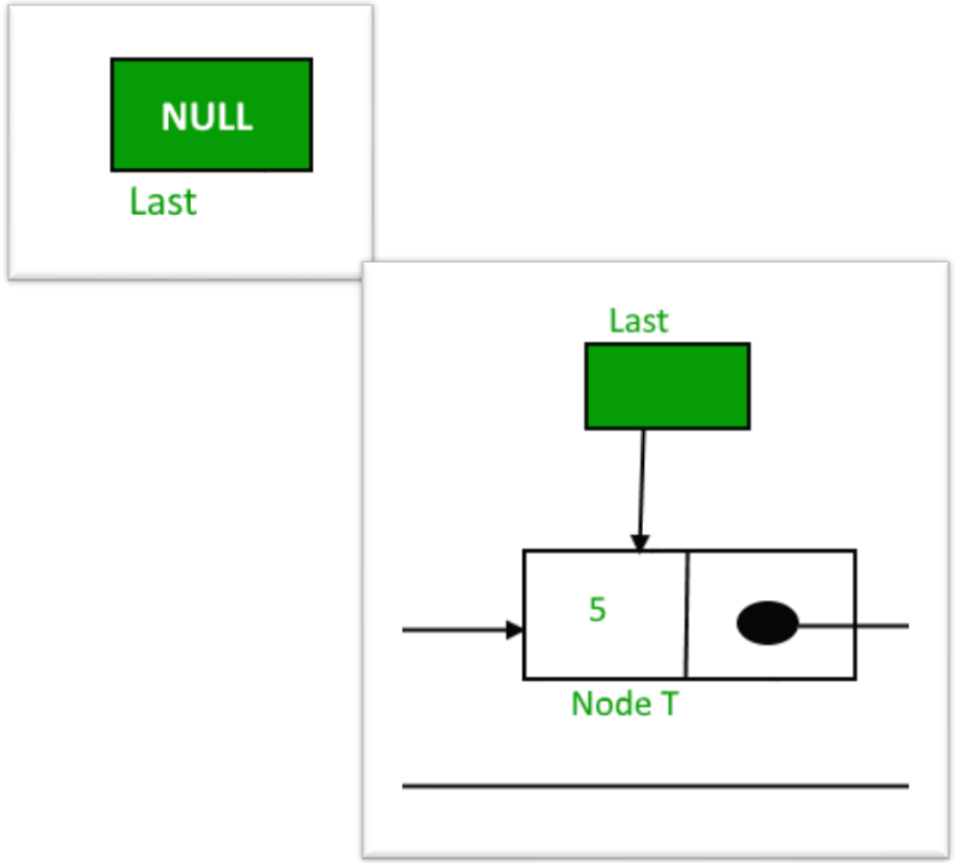
Portanto, definimos o elemento da lista ligada circularmente usando uma estrutura que possui dois membros - valor e próximo ponteiro.

Vamos então desenvolver as principais funções inerentes a uma lista circular. Quais sejam:

Insere()
Remove()
Busca()
Mostra()

INSERE ()

Para inserir um elemento ao final de uma lista circular, precisa-se primeiramente encontrar o último elemento.
Sabendo disso, alocamos memória para prox ser um novo elemento, definimos o dado do prox e o prox do prox para lista.

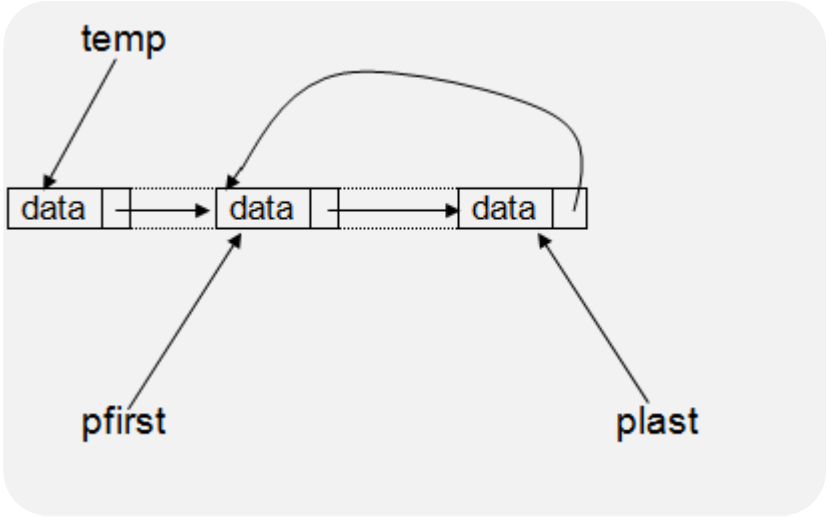


REMOVE ()

Para deletar um elemento da lista circular deve-se considerar:

Se o elemento a ser deletado é o primeiro elemento da lista e a lista contém apenas um elemento atribui-se NULL aos ponteiros pfirst plast.

Se o elemento a ser excluído for o primeiro elemento da lista e a lista contiver mais de um elemento, é necessário um ponteiro temporário para apontar para pfirst e, em seguida, mover pfirst para apontar para o próximo elemento e definir o ponteiro temporário como NULL.



REMOVE ()

Para deletar um elemento da lista circular deve-se considerar:

Se o elemento a ser excluído estiver no meio da lista, necessita-se de um ponteiro de deslocamento (temp) para apontar para o elemento **antes** do elemento a ser excluído e de um ponteiro temporário (del) para apontar para o elemento a ser excluído.
Em seguida, deixe o link do ponteiro de deslocamento apontar para o link do ponteiro temporário.

Para lidar com situações em que o elemento a ser excluído é o último elemento da lista, é necessário testar se o item alvo é igual ao *plast*. Se for realmente igual, deve-se atualizar o ponteiro de *plast* para apontar para o ponteiro de deslocamento. Por fim, definir o ponteiro temporário para NULL

BUSCA ()

Outra operação importante da lista circular é a busca de um item. Procurar por um item específico na lista é um processo seqüencial. A comparação começa do início da lista até que o item de destino seja encontrado ou até o final da lista ser atingido. Portanto, o primeiro item correspondido é retornado.

```
void busca(int x)
{
    system("cls");
    NO *t = start;

    int encontra = 0;

    while(t -> proximo != start)
    {
        if( t -> valor == x)
            {
                cout << "\nEncontrado!";

                encontra = 1;

                break;
            }

        t = t -> proximo;
    }

                if(encontra == 0)
                    {
                        cout<<"\nNao encontrado!";
                    }
}
```

MOSTRA()

Mostrar todos os elementos da lista é simples, deve-se percorrer a lista e gerar os dados de cada elemento

```
void mostra()
{
    system("cls");
    NO *t = start;

        do
            {
                cout << t -> valor << "\t";

                t = t -> proximo;
            }

        while (t != start);

}
```

Aplicações

Alternativa ao problema de timesharing resolvido pelo sistema operacional. Em um ambiente de tempo compartilhado, o sistema operacional deve manter uma lista de usuários presentes e deve alternadamente permitir que cada usuário use uma pequena parte do tempo da CPU, um usuário por vez.

O sistema operacional selecionará um usuário, permitirá que ele use uma pequena quantidade de tempo de CPU e passará para o próximo usuário. Para este aplicativo, não deve haver nenhum ponteiro NULL, a menos que não haja absolutamente ninguém solicitando tempo de CPU, ou seja, a lista está vazia.