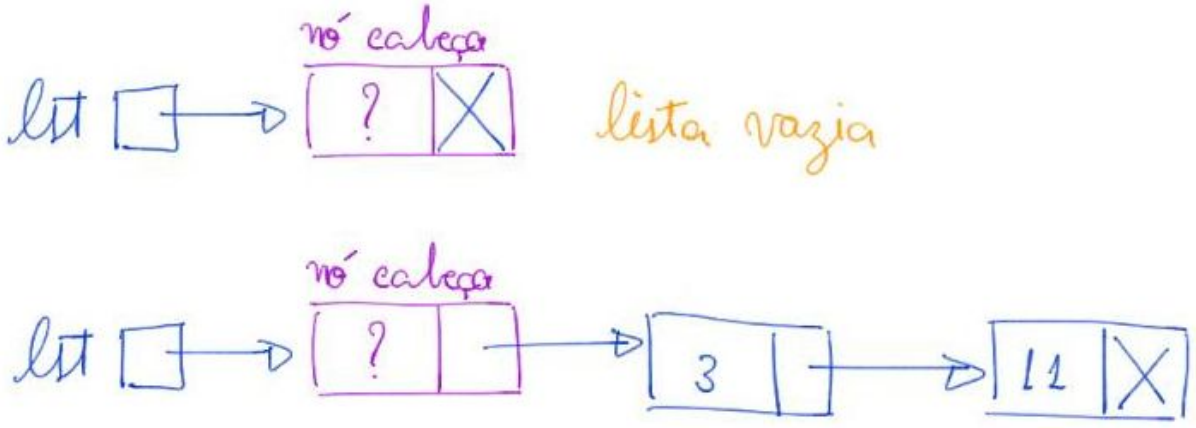
AED1 - Aula 10
Listas ligadas com nó cabeça, circulares e duplamente ligadas

Vamos ver algumas variantes da lista encadeada padrão e entender suas vantagens e desvantagens.

Listas ligadas com nó cabeça

- Estas listas têm uma célula inicial chamada nó cabeça.
- O nó cabeça está presente mesmo quando a lista está vazia.
- Sua principal vantagem é simplificar operações como
    - inserção e remoção.
- Isto porque, com a presença do nó cabeça,
    - toda célula tem uma antecessora.
- Uma desvantagem é desperdiçar a memória de uma célula.



Declaração da estrutura da célula:

```
typedef struct celula Celula;
struct celula
{
    int conteudo;
    Celula *prox;
};
```

- Note que é a mesma da lista ligada simples.

Exemplos de inicialização de uma lista com nó cabeça.

- Alocação estática:

```
    Celula *ini;
```

```
    Celula cabeca;
    ini = &cabeca;
    cabeca.prox = NULL;
```

- Alocação dinâmica:

```
    Celula *ini;
    ini = malloc(sizeof(Celula));
    ini->prox = NULL;
```

Imprime conteúdo de uma lista com nó cabeça

```
void imprime(Celula *lst)
{
    Celula *p = lst->prox; // pula o nó cabeça
    while (p != NULL)
    {
        printf("%d ", p->conteudo);
        p = p->prox;
    }
    printf("\n");
}
```

- Exemplo de uso

```
    imprime(ini);
```

Busca, encontrar um elemento leva tempo O(n) no pior caso.

```
Celula *busca(Celula *lst, int x)
{
    Celula *p = lst->prox; // pula o nó cabeça
    while (p != NULL && p->conteudo != x)
        p = p->prox;
    return p;
}
```

- Exemplo de uso

```
    Celula *p = busca(ini, 10);
```

Seleção, pegar o conteúdo do k-ésimo item leva tempo O(k).

```
Celula *selecao(Celula *lst, int k)
{
    Celula *p = lst->prox; // pula o nó cabeça
```

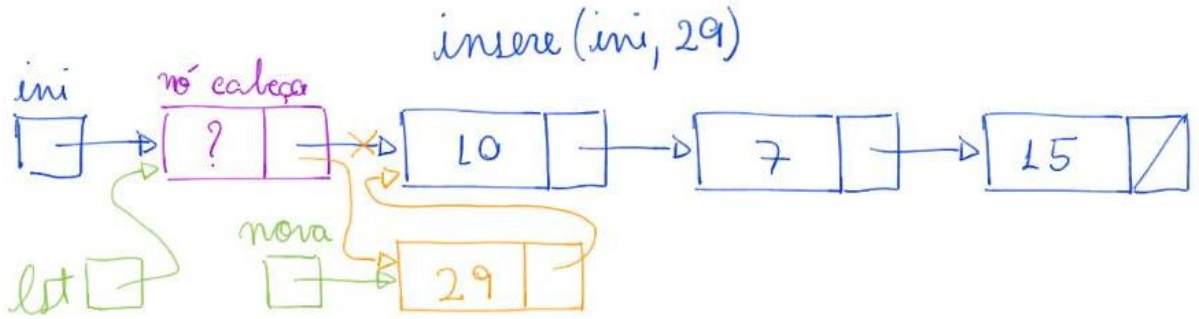
```
    int pos = 0;
    while (p != NULL && pos < k)
    {
        p = p->prox;
        pos++;
    }
    return p;
}
```

- Exemplo de uso

```
    Celula *q = selecao(ini, 10);
```

Inserção, inserir um elemento no início da lista, ou na frente de uma célula

- para a qual já temos um apontador, leva tempo constante, i.e., O(1).




```
void insere(Celula *lst, int x)
{
    Celula *nova;
    nova = malloc(sizeof(Celula));
    nova->conteudo = x;
    nova->prox = lst->prox;
    lst->prox = nova;
}
```

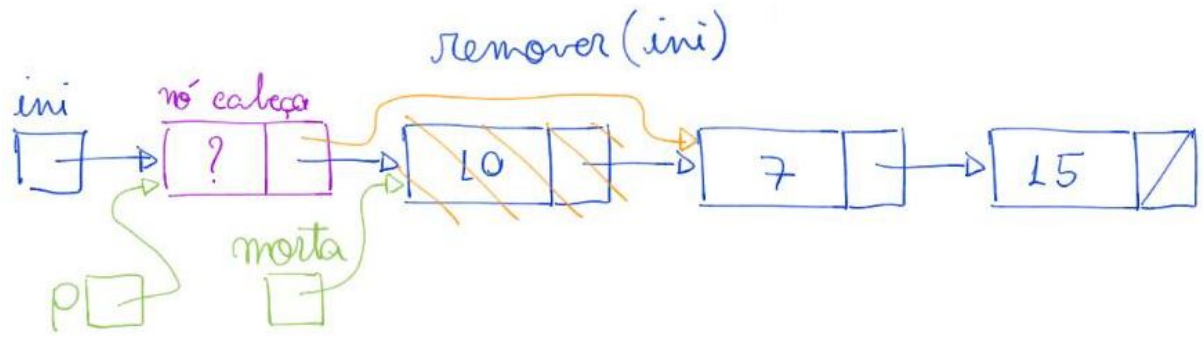
    - Note que, a função não usa apontador de apontador,
        - nem devolve um endereço.

- Exemplo de uso

```
    printf("Insere n elementos na lista\n");
    for (i = 0; i < n; i++)
        insere(ini, i);
```

Remoção, remover um elemento no início da lista, ou a célula seguinte

- leva tempo constante, i.e., O(1).

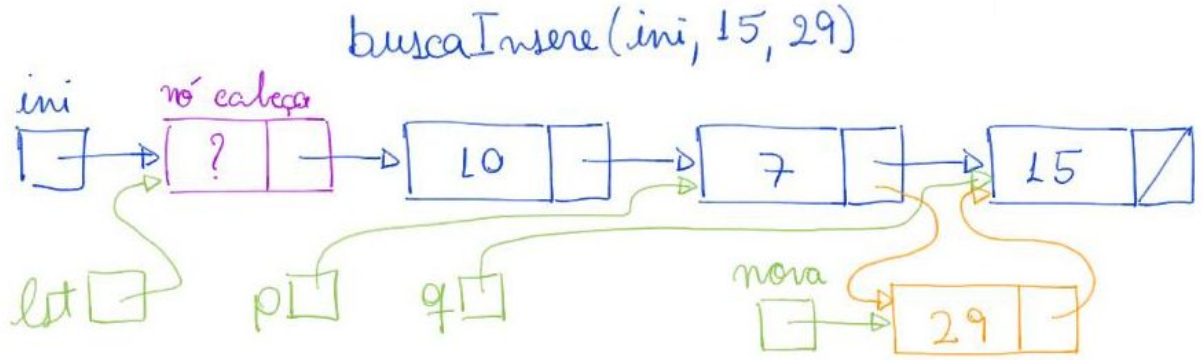



```c
// remove a celula sucessora de p
// supõe que p != NULL e p->prox != NULL
void remover(Celula *p)
{
    Celula *morta;
    morta = p->prox;
    p->prox = morta->prox;
    free(morta);
}
```

- Exemplos de uso

```c
    printf("remover(ini)\n");
    remover(ini);
    printf("remover(ini->prox)\n");
    remover(ini->prox);
```

Busca e insere, buscar um elemento x e inserir y logo antes dele leva tempo O(n).




```c
// busca x na lista lst e insere y logo antes de x
// se x não está na lista insere y no final
void buscaInsere(Celula *lst, int x, int y)
{
    Celula *p, *q, *nova;
    nova = malloc(sizeof(Celula));
```

```
    nova->conteudo = y;
    p = lst;
    q = p->prox;
    while (q != NULL && q->conteudo != x)
    {
        p = q;
        q = p->prox;
    }
    p->prox = nova;
    nova->prox = q;
}
```

- Exemplos de uso

```
    printf("buscaInsere(ini, 2, 3)\n");
    buscaInsere(ini, 2, 3);
    printf("buscaInsere(ini, ini->prox->conteudo, 49)\n");
    buscaInsere(ini, ini->prox->conteudo, 49);
    printf("buscaInsere(ini, 15, 17)\n");
    buscaInsere(ini, 15, 17);
```

Busca e remove, buscar um elemento x e removê-lo leva tempo O(n).

```
void buscaRemove(Celula *lst, int x)
{
    Celula *p, *morta;
    p = lst;
    while (p->prox != NULL && p->prox->conteudo != x)
        p = p->prox;
    if (p->prox != NULL)
    {
        morta = p->prox;
        p->prox = morta->prox;
        free(morta);
    }
}
```

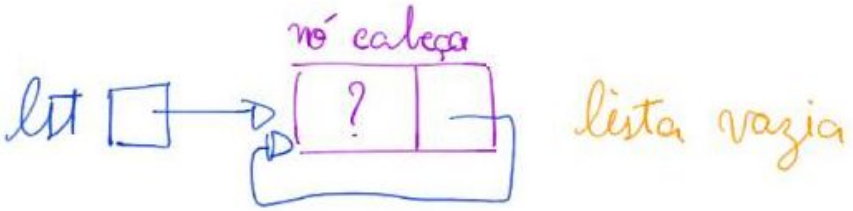
- Exemplos de uso

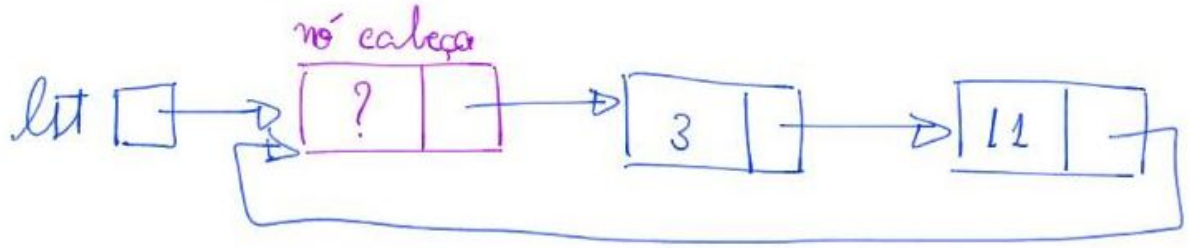
```
    printf("buscaRemove(ini, 2)\n");
    buscaRemove(ini, 2);
```

```
    printf("buscaRemove(ini, ini->prox->conteudo)\n");
    buscaRemove(ini, ini->prox->conteudo);
    printf("buscaRemove(ini, 17)\n");
    buscaRemove(ini, 17);
```

Listas ligadas circulares

- A última célula desta lista aponta para a primeira.
- Se for uma lista encadeada circular com nó cabeça,
    - a última célula aponta para o nó cabeça.





- Para verificar se chegou ao final da lista
    - é necessário verificar se o apontador para o próximo
        - tem o endereço da primeira célula,
            - ou para o nó cabeça.

Declaração da estrutura da célula:

```
typedef struct celula Celula;
struct celula
{
    int conteudo;
    Celula *prox;
};
```

- Note que é a mesma da lista ligada simples.

Exemplos de inicialização de uma lista circular com nó cabeça.

- Alocação estática:

```
    Celula *ini;
```

```
    Celula cabeca;
    ini = &cabeca;
    cabeca.prox = &cabeca;
```

- Alocação dinâmica:

```
    Celula *ini;
    ini = malloc(sizeof(Celula));
    ini->prox = ini;
```

Imprime conteúdo de uma lista circular com nó cabeça

```
void imprime(Celula *lst)
{
    Celula *p = lst->prox;
    while (p != lst)
    {
        printf("%d ", p->conteudo);
        p = p->prox;
    }
    printf("\n");
}
```

Busca, encontrar um elemento leva tempo O(n) no pior caso.

```
Celula *busca(Celula *lst, int x)
{
    Celula *p = lst->prox;
    while (p != lst && p->conteudo != x)
        p = p->prox;
    return p;
}
```

Seleção, pegar o conteúdo do k-ésimo item leva tempo O(k).

```
Celula *selecao(Celula *lst, int k)
{
    Celula *p = lst->prox;
    int pos = 0;
    while (p != lst && pos < k)
    {
        p = p->prox;
```

```
        pos++;
    }
    return p;
}
```

Inserção, inserir um elemento no início da lista, ou na frente de uma célula

- para a qual já temos um apontador, leva tempo constante, i.e., O(1).

```
void insere(Celula *lst, int x)
{
    Celula *nova;
    nova = malloc(sizeof(Celula));
    nova->conteudo = x;
    nova->prox = lst->prox;
    lst->prox = nova;
}
```

Remoção, remover um elemento no início da lista, ou a célula seguinte

- leva tempo constante, i.e., O(1).

```
// remove a celula sucessora de p
// supõe que p != NULL e p->prox != NULL
void remover(Celula *p)
{
    Celula *morta;
    morta = p->prox;
    p->prox = morta->prox;
    free(morta);
}
```

Busca e insere, buscar um elemento x e inserir y logo antes dele leva tempo O(n).

```
// busca x na lista lst e insere y logo antes de x
// se x não está na lista insere y no final
void buscaInsere(Celula *lst, int x, int y)
{
    Celula *p, *q, *nova;
    nova = malloc(sizeof(Celula));
    nova->conteudo = y;
    p = lst;
```

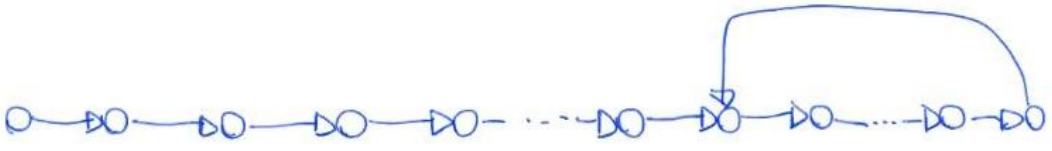
```
    q = p->prox;
    while (q != lst && q->conteudo != x)
    {
        p = q;
        q = p->prox;
    }
    p->prox = nova;
    nova->prox = q;
}
```

Busca e remove, buscar um elemento x e removê-lo leva tempo O(n).

```
void buscaRemove(Celula *lst, int x)
{
    Celula *p, *morta;
    p = lst;
    while (p->prox != lst && p->prox->conteudo != x)
        p = p->prox;
    if (p->prox != lst)
    {
        morta = p->prox;
        p->prox = morta->prox;
        free(morta);
    }
}
```

Quiz:

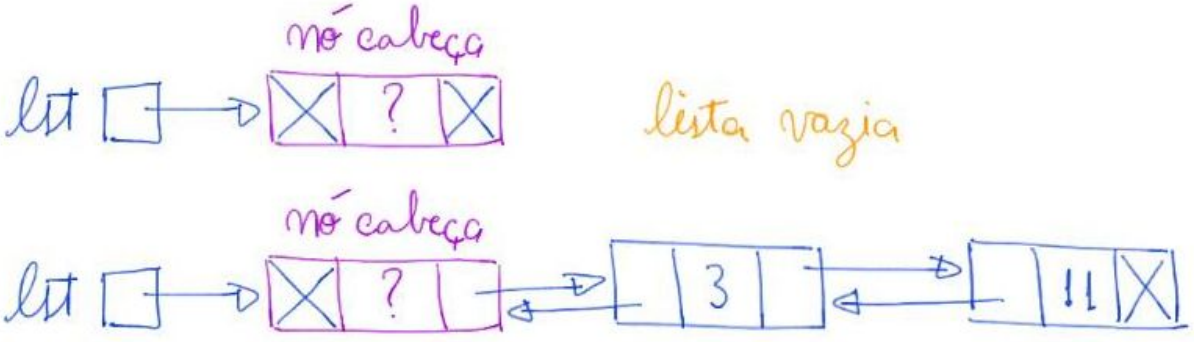
- Como detectar se uma lista é circular?
- E como verificar se uma lista tem algum laço,
    - ou seja, se a última célula aponta para qualquer outra célula?



    - Dica: considere usar dois apontadores.

Listas duplamente ligadas

- Esta lista pode ser percorrida em ambos os sentidos.
- Para tanto, suas células tem dois campos apontadores
    - um campo prox que guarda o endereço da próxima célula
    - e um campo ante que guarda o endereço da célula anterior.
- Desvantagem:
    - maior consumo de memória por célula.



Registro de uma célula de lista encadeada duplamente ligada:

```
typedef struct celula Celula;
struct celula
{
    int conteudo;
    Celula *ante;
    Celula *prox;
};
```

Exemplos de inicialização de uma lista duplamente ligada com nó cabeça.

- Alocação estática:

```
    Celula *ini;
    Celula cabeca;
    ini = &cabeca;
    cabeca.ante = NULL; // necessario?
    cabeca.prox = NULL;
```

- Alocação dinâmica:

```
    Celula *ini;
    ini = malloc(sizeof(Celula));
    ini->ante = NULL; // necessario?
    ini->prox = NULL;
```

Imprime lista.

```
void imprime(Celula *lst)
```

```
{
    Celula *p = lst->prox;
    while (p != NULL)
    {
        printf("%d ", p->conteudo);
        p = p->prox;
    }
    printf("\n");
}
```

Busca.

```
Celula *busca(Celula *lst, int x)
{
    Celula *p = lst->prox;
    while (p != NULL && p->conteudo != x)
        p = p->prox;
    return p;
}
```

Seleção.

```
Celula *selecao(Celula *lst, int k)
{
    Celula *p = lst->prox;
    int pos = 0;
    while (p != NULL && pos < k)
    {
        p = p->prox;
        pos++;
    }
    return p;
}
```

Inserção.

```
void insere(Celula *lst, int x)
{
    Celula *nova;
```

```
    nova = malloc(sizeof(Celula));
    nova->conteudo = x;
    nova->prox = lst->prox;
    if (nova->prox != NULL)
        nova->prox->ante = nova;
    lst->prox = nova;
    nova->ante = lst;
}
```

Remoção.

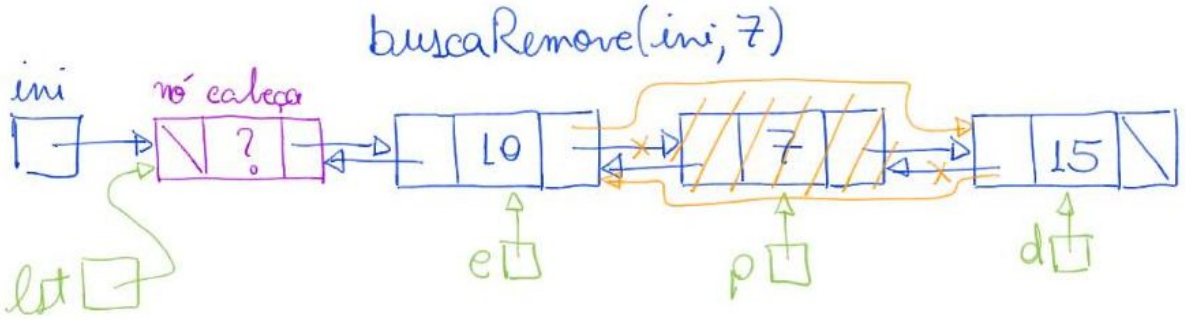
```
// remove a celula sucessora de p
// supõe que p != NULL e p->prox != NULL
void remover(Celula *p)
{
    Celula *morta;
    morta = p->prox;
    p->prox = morta->prox;
    if (p->prox != NULL)
        p->prox->ante = p;
    free(morta);
}
```

Busca e inserção.

```
// busca x na lista lst e insere y logo antes de x
// se x não está na lista insere y no final
void buscaInsere(Celula *lst, int x, int y)
{
    Celula *p, *q, *nova;
    nova = malloc(sizeof(Celula));
    nova->conteudo = y;
    p = lst;
    q = p->prox;
    while (q != NULL && q->conteudo != x)
    {
        p = q;
        q = p->prox;
```

```
    }
    p->prox = nova;
    nova->ante = p;
    nova->prox = q;
    if (nova->prox != NULL)
        nova->prox->ante = nova;
}
```

Busca e remoção.




```
void buscaRemove(Celula *lst, int x)
{
    Celula *p, *e, *d;
    p = busca(lst, x);
    if (p != NULL)
    {
        e = p->ante;
        d = p->prox;
        e->prox = d;
        if (d != NULL)
            d->ante = e;
        free(p);
    }
}
```

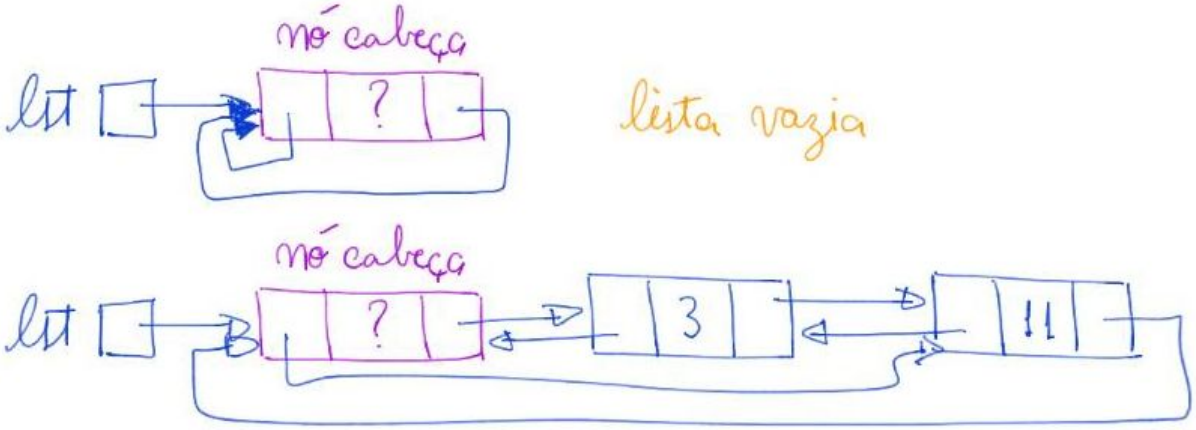
Quiz:

- Para quais células a função de remoção destas listas pode apontar?
    - E a função de inserção?

Extra:

- Podemos combinar as variantes obtendo

- listas duplamente encadeadas circulares com nó cabeça.



- Note que isso pode possibilitar novas operações,
  - como imprimir a lista do fim para o começo.