

Listas

Murilo Dantas

Tópicos desta aula

1. Motivação
2. Listas simplesmente encadeadas
3. Listas duplamente encadeadas
4. Listas circulares

Motivação

- Estrutura de dados conhecida: **vetores**
- Exemplo de uso:
 - ▶ Manter uma lista de valores ordenada.
- Se há mais de um valor, usa-se **struct**

```
typedef struct lista
{
    int cliente;
    float valor;
} lista;

lista v[MAX];
```

0003	0018	0505	2122	0018	
95.00	84.50	71.90	65.00	58.00	

Motivação

- Para alguns problemas, o vetor não ajuda
 - ▶ Inserir novo elemento, mantendo a ordem.

▶ cliente: 0101; pedido: 99.00



0003	0018	0505	2122	0018	
95.00	84.50	71.90	65.00	58.00	

	0003	0018	0505	2122	0018
	95.00	84.50	71.90	65.00	58.00


0101	0003	0018	0505	2122	0018
99.00	95.00	84.50	71.90	65.00	58.00

Problemas

1. Muita movimentação de dados.
2. O esforço computacional depende dos dados a serem inseridos.

Motivação

- No caso de exclusão, também
 - ▶ Excluir, mantendo a ordem.
 - ▶ cliente: 0003; pedido: 95.00



0101	0003	0018	0505	2122	0018
99.00	95.00	84.50	71.90	65.00	58.00

0101	0018	0505	2122	0018	
99.00	84.50	71.90	65.00	58.00	

Solução: dados encadeados

1. Não movimenta os dados.
2. Esforço computacional constante.

Definição

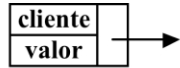
- Lista encadeada
 - ▶ É uma estrutura de dados que armazena a informação do problema (como **cliente** e **valor**), e um ponteiro para a própria estrutura.

```
typedef struct lista
{
    int cliente;
    float valor;
    struct lista *prox;
} lista;
```

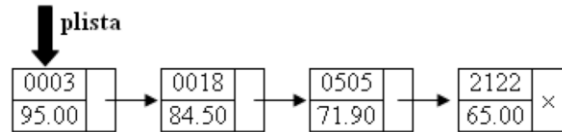
A variável **prox** é do
tipo **struct lista ***

Visualização

- Variável do tipo lista



- Lista de pedidos dos clientes



Ponteiros importantes

- **plista**
 - ▶ É o ponteiro que referencia a lista.
- **prox**
 - ▶ Aponta para o endereço do próximo elemento.
- **NULL**
 - ▶ Indica o final da lista.

Vetor x Lista

- Vetor
 - ▶ Também chamado de lista estática.
 - ▶ Existe uma **ordenação física** dos elementos.
 - ▶ Posições consecutivas de memória (dados contíguos).
 - ▶ As movimentações de dados são necessárias para manter o vetor ordenado “sem buracos”.

Vetor x Lista

- Lista
 - ▶ Existe uma **ordenação lógica** dos elementos da lista (alocação dinâmica).
 - ▶ Os elementos da lista ocupam posições quaisquer de memória, não necessariamente consecutivas.
 - ▶ É preciso indicar (por meio de outro ponteiro) qual é o “primeiro” elemento da lista.

Vetor x Lista

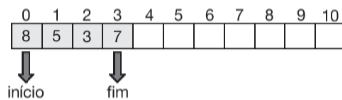
- Lista
 - ▶ Representa um conjunto de dados dispostos linearmente.
 - ▶ Pode crescer ou diminuir ao longo do tempo.
 - ▶ Principais operações:
 - ▶ Inserir, excluir, buscar, encontrar o maior ou o menor, contar os elementos, alterá-los, buscar o sucessor e o predecessor.

Vetor x Lista

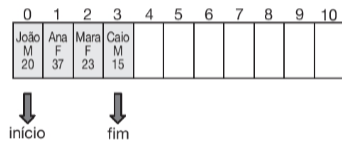
- Lista
 - ▶ Pode ser:
 - ▶ Homogênea: contém apenas um dado primitivos.
 - ▶ Heterogênea: contém dados compostos.
 - ▶ Tipos:
 - ▶ Simples ordenada/não ordenada, dupla orden./ não ordenada e circulares.

Representação dos tipos de listas

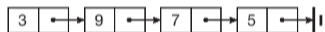
Lista estática homogênea



Lista estática heterogênea



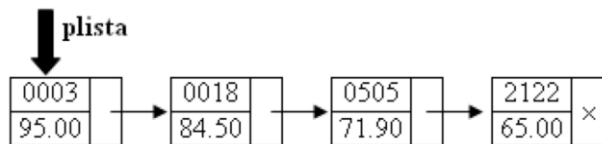
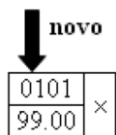
Lista dinâmica homogênea



Lista dinâmica heterogênea

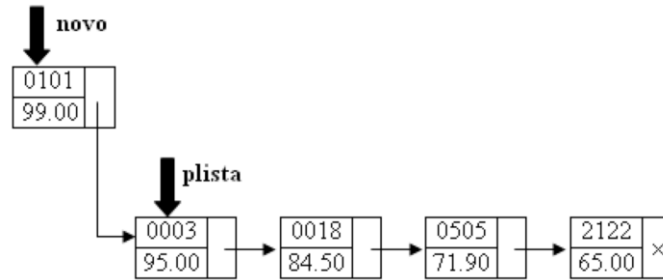


Inclusão na lista

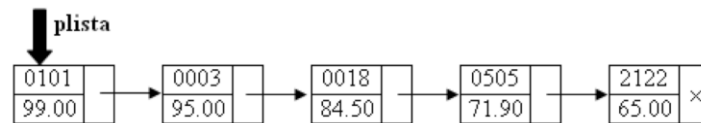


Inclusão: operações

novo → prox = lista;



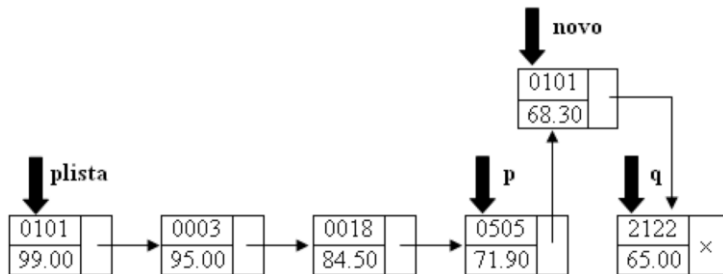
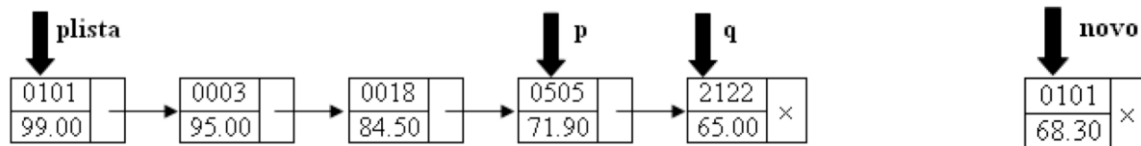
lista = novo;



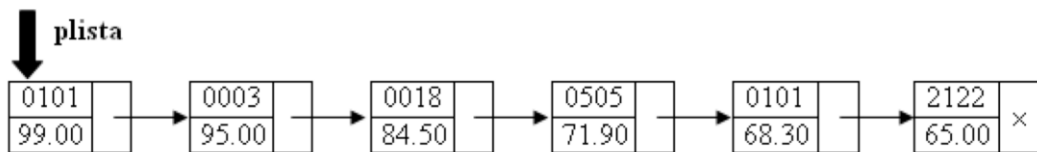
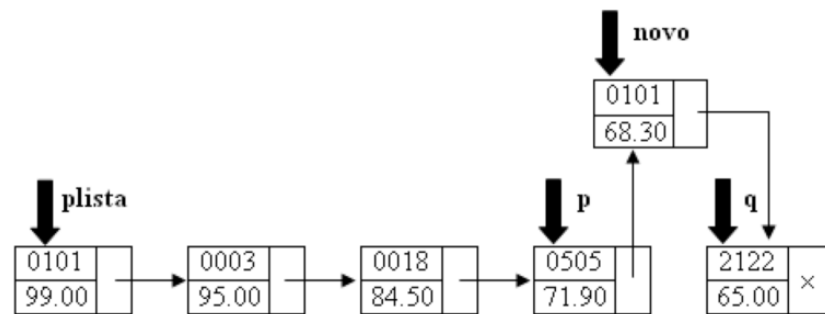
Inclusão no “meio” da lista

$p \rightarrow \text{prox} = \text{novo};$

$\text{novo} \rightarrow \text{prox} = q;$



Logicamente iguais!



Alguns detalhes

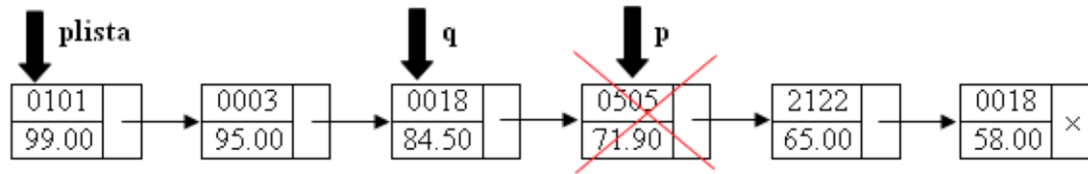
- Inclusão no “final” ($q = \text{NULL}$)
 - ▶ O algoritmo de inclusão é o mesmo.
- Inclusão no “início” ($p = \text{NULL}$ e $q = \text{plista}$)
 - ▶ Cuidado, pois o ponteiro p não acessa um prox!

Algoritmo de inclusão em lista

```
void InclusaoLista(lista novo, lista p, lista q)
{
    if (p == NULL)
        plista = novo;
    else
        p->prox = novo;
    novo->prox = q;
}
```

Exclusão de lista

- Método seguro
 - ▶ Para a exclusão devemos ter um ponteiro para o elemento a ser excluído (ponteiro **p**) e um outro ponteiro para o elemento anterior (ponteiro **q**).

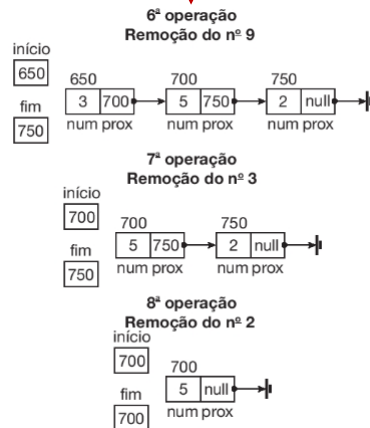
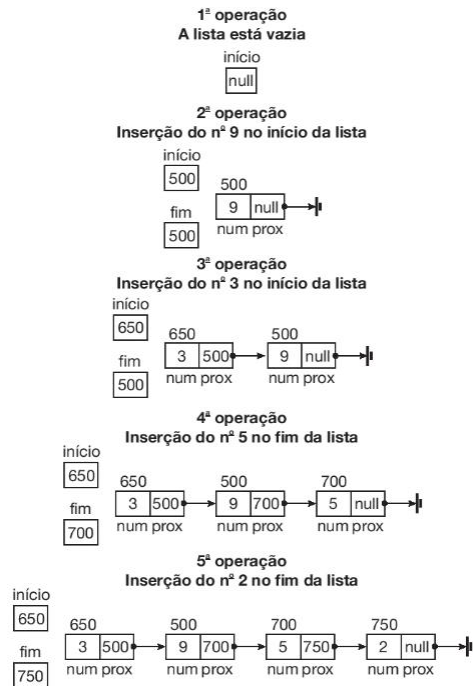


Exclusão de lista: atenção!

- Exclusão do “último”
 - ▶ $q = \text{NULL}$ e $p = \text{plista}$
 - ▶ q não tem prox.
- Algoritmo de exclusão

```
void ExclusaoLista(lista p, lista q)
{
    if (q == NULL)
        plista = p->prox;
    else
        q->prox = p->prox;
    free(p);
}
```

Lista não ordenada



Lista não ordenada

```
#include <iostream.h>
#include <conio.h>

void main()
{
    //Definindo o registro que representará
    //cada elemento da lista
    struct LISTA
    {
        int num;
        LISTA *prox;
    };

    // a lista está vazia, logo,
    // o ponteiro inicio tem o valor NULL
    // o ponteiro inicio conterá o endereço
    // do primeiro elemento da lista
    LISTA *inicio = NULL;
    // o ponteiro fim conterá o endereço
    // do último elemento da lista
    LISTA *fim = NULL;
    // o ponteiro aux é um ponteiro auxiliar
    LISTA *aux;
    // o ponteiro anterior é um ponteiro auxiliar
    LISTA *anterior;
    // apresentando o menu de opções
    int op, numero, achou;
```

```
do
{
    clrscr();
    cout<<"\nMENU DE OPÇÕES\n";
    cout<<"\n1 - Inserir no início da
    ↳ lista";
    cout<<"\n2 - Inserir no fim da lista";
    cout<<"\n3 - Consultar toda a lista";
    cout<<"\n4 - Remover da lista";
    cout<<"\n5 - Esvaziar a lista";
    cout<<"\n6 - Sair";
    cout<<"\nDigite sua opção: ";
    cin>>op;
    if (op < 1 || op > 6)
        cout<<"Opção inválida!!";
    if (op == 1)
    {
        cout<<"Digite o número a ser
        ↳ inserido no início da lista:";
        LISTA *novo = new LISTA();
        cin>>novo->num;
        if (inicio == NULL)
        {
            // a lista estava vazia
            // e o elemento inserido será
            // o primeiro e o último
            inicio = novo;
            fim = novo;
            fim->prox = NULL;
        }
        else
        {
```

Lista não ordenada

```
// a lista já contém elementos
// e o novo elemento
// será inserido no início da
// lista
novo->prox = inicio;
inicio = novo;
}
cout<<"Número inserido no
↳ início da lista!!";
}
if (op == 2)
{
    cout<<"Digite o número a ser
    ↳ inserido no fim da lista: ";
    LISTA *novo = new LISTA();
    cin>>novo->num;
    if (inicio == NULL)
    {
        // a lista estava vazia
        // e o elemento inserido será
        // o primeiro e o último
        inicio = novo;
        fim = novo;
        fim->prox = NULL;
    }
}
```

```
else
{
    // a lista já contém elementos e
    // o novo elemento
    // será inserido no fim da lista
    fim->prox = novo;
    fim = novo;
    fim->prox=NULL;
}
cout<<"Número inserido no fim da
↳ lista!!";
}
if (op == 3)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
```

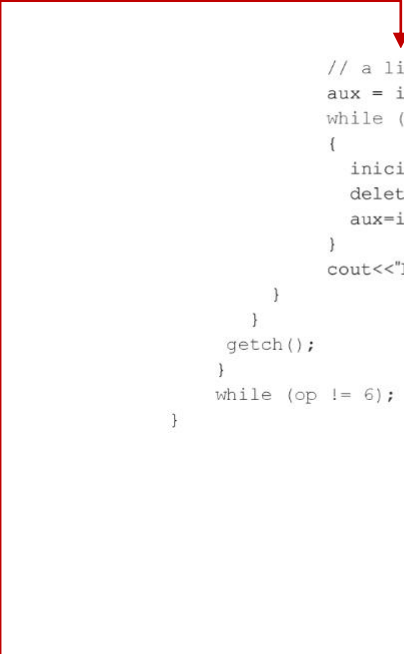

Lista não ordenada

```
// a lista contém elementos e estes serão
// mostrados do início ao fim
cout<<"\nConsultando toda a lista\n";
aux = inicio;
while (aux != NULL)
{
    cout<<aux->num<<" ";
    aux = aux->prox;
}
}
if (op == 4)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
        // a lista contém elementos e o elemento
        // a ser removido deve ser digitado
        cout<<"\nDigite o elemento a ser removido:";
        cin>>numero;
        // todas as ocorrências da lista, iguais ao
        // número digitado, serão removidas
        aux = inicio;
        anterior = NULL;
        achou = 0;
```

```
while (aux != NULL)
{
    if (aux->num == numero)
    {
        // o número digitado
        // foi encontrado na lista
        // e será removido
        achou = achou + 1;
        if (aux == inicio)
        {
            // o número a ser removido
            // é o primeiro da lista
            inicio = aux->prox;
            delete(aux);
            aux = inicio;
        }
    }
    else if (aux == fim)
    {
        // o número a ser removido
        // é o último da lista
        anterior->prox = NULL;
        fim = anterior;
        delete(aux);
        aux = NULL;
    }
    else
    {
        // o número a ser
        // removido
        // está no meio da
        // lista
        anterior->prox = aux->prox;
        delete(aux);
        aux = anterior->prox;
    }
}
```

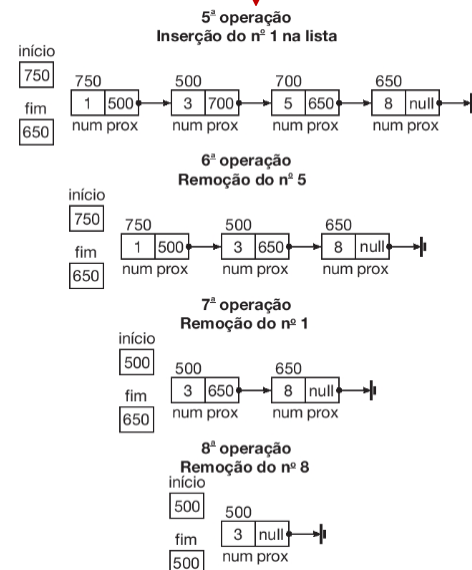
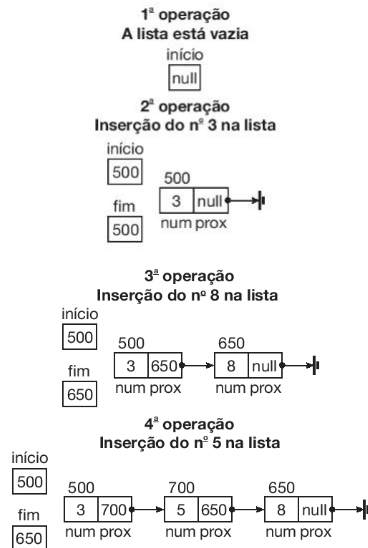
Lista não ordenada

```
        else
        {
            anterior = aux;
            aux = aux->prox;
        }
    }
    if (achou == 0)
        cout<<"Número não encontrado";
    else if (achou == 1)
        cout<<"Número removido 1 vez";
    else
        cout<<"Número removido "<<achou<<"
        ↳ vezes";
    }
}
if (op == 5)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!";
    }
    else
    {
```



```
        // a lista será esvaziada
        aux = inicio;
        while (aux != NULL)
        {
            inicio = inicio->prox;
            delete(aux);
            aux=inicio;
        }
        cout<<"Lista esvaziada";
    }
    getch();
}
while (op != 6);
}
```

Lista ordenada



Lista ordenada

```
#include <iostream.h>
#include <conio.h>

void main()
{
    //Definindo o registro que
    //representará cada elemento da lista
    struct LISTA
    {
        int num;
        LISTA *prox;
    };

    // a lista está vazia, logo,
    // o ponteiro inicio têm o valor NULL
    // o ponteiro inicio conterá o endereço
    // do primeiro elemento da lista
    LISTA *inicio = NULL;
    // o ponteiro fim conterá o endereço
    // do último elemento da lista
    LISTA *fim = NULL;
    // o ponteiro aux é um ponteiro auxiliar
    LISTA *aux;
    // o ponteiro anterior é um ponteiro auxiliar
    LISTA *anterior;
    // apresentando o menu de opções
    int op, numero, achou;
    do
```

```
{
    clrscr();
    cout<<"\nMENU DE OPÇÕES\n";
    cout<<"\n1 - Inserir na lista";
    cout<<"\n2 - Consultar toda a lista";
    cout<<"\n3 - Remover da lista";
    cout<<"\n4 - Esvaziar a lista";
    cout<<"\n5 - Sair";
    cout<<"\nDigite sua opção: ";
    cin>>op;
    if (op < 1 || op > 5)
        cout<<"Opção inválida!!";
    if (op == 1)
    {
        cout<<"Digite o número a ser
        ↳ inserido na lista: ";
        LISTA *novo = new LISTA();
        cin>>novo->num;
        if (inicio == NULL)
        {
            // a lista estava vazia
            // e o elemento inserido será
            // o primeiro e o último
            inicio = novo;
            fim = novo;
            novo->prox = NULL;
        }
    }
}
```

Lista ordenada

```
else
{
    // a lista já contém elementos
    // e o novo elemento
    // será inserido na lista
    // respeitando a ordenação
    // crescente
    anterior = NULL;
    aux = inicio;
    while (aux != NULL
    && novo->num > aux->num)
    {
        anterior = aux;
        aux = aux->prox;
    }
    if (anterior == NULL)
    {
        // o novo número a ser inserido
        // é menor que todos os
        // números da lista,
        // logo, será inserido no
        // início
        novo->prox = inicio;
        inicio = novo;
    }
    else if (aux == NULL)
    {
        // o novo número a ser
        // inserido
        // é maior que todos os
        // números da
        // lista, logo, será
        // inserido no fim
```

```
        fim->prox = novo;
        fim = novo;
        fim->prox=NULL;
    }
    else
    {
        // o novo número a ser
        // inserido
        // será inserido entre
        // dois
        // números que já estão na
        // lista
        anterior->prox =
        ~ novo;
        novo->prox = aux;
    }
    cout<<"Número inserido na lista!!";
}
if (op == 2)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
```

Lista ordenada

```
}
else
{
    // a lista contém elementos e
    // estes serão
    // mostrados do início ao fim
    cout<<"\nConsultando toda a
    ↳ lista\n";
    aux = inicio;
    while (aux != NULL)
    {
        cout<<aux->num<<" ";
        aux = aux->prox;
    }
}
if (op == 3)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!";
    }
    else
    {
        // a lista contém elementos
        // e o elemento a ser
        // removido deve ser digitado
        cout<<"\nDigite o elemento a
        ↳ ser removido:";
```


```
cin>>numero;
// todas as ocorrências da
// lista,
// iguais ao número digitado,
// serão removidas
aux = inicio;
anterior = NULL;
achou = 0;
while (aux != NULL)
{
    if (aux->num == numero)
    {
        // o número digitado
        // foi encontrado na lista
        // e será removido
        achou = achou + 1;
        if (aux == inicio)
        {
            // o número a ser
            // removido
            // é o primeiro da
            // lista
            inicio = aux->prox;
            delete(aux);
            aux = inicio;
        }
        else if (aux == fim)
        {
            // o número a ser
            // removido
            // é o último da
            // lista
            anterior->prox =
            ↳ NULL;
            fim = anterior;
            delete(aux);
            aux = NULL;
        }
    }
}
```

Lista ordenada

```
        else
        {
            // o número a ser
            // removido
            // está no meio
            // da lista
            anterior->prox
            ↳ =aux->prox;
            delete(aux);
            aux = anterior->
            ↳ >prox;
        }
    }
    else
    {
        anterior = aux;
        aux = aux->prox;
    }
}

if (achou == 0)
    cout<<"Número não encontrado";
else if (achou == 1)
    cout<<"Número removido 1 vez";
else
    cout<<"Número removido "<<achou<<"
    ↳ vezes";
}

if (op == 4)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
}
```



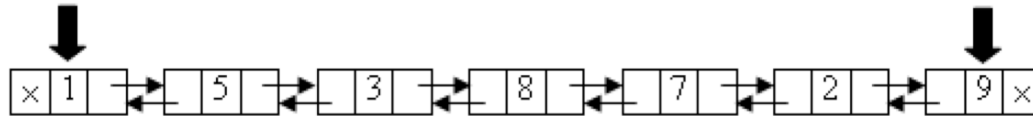
```
    else
    {
        // a lista será esvaziada
        aux=inicio;
        while (aux != NULL)
        {
            inicio =inicio->prox;
            delete(aux);
            aux = inicio;
        }
        cout<<"Lista esvaziada";
    }
}

getch();
}

while (op != 5);
}
```

Lista duplamente encadeada

- Definição prática
 - ▶ Contém ponteiros para o “próximo” elemento e para o “anterior”.
 - ▶ Pode tornar o “trânsito” entre os nós mais rápido.



Lista duplamente encadeada

- Estrutura

```
typedef struct lista
{
    int digito;
    struct lista *prox;
    struct lista *prev;
} lista;
```

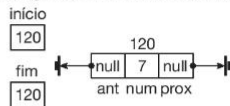
- ▶ **prox** aponta para o próximo elemento.
- ▶ **prev** aponta para o elemento anterior da lista.

Lista duplamente encadeada

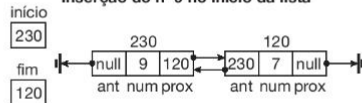
1ª operação
A lista está vazia



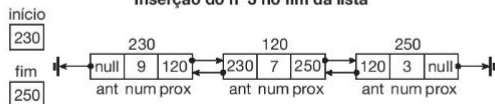
2ª operação
Inserção do nº 7 no início da lista



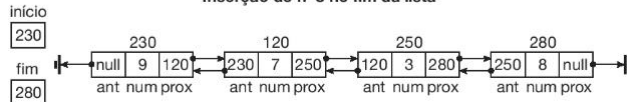
3ª operação
Inserção do nº 9 no início da lista



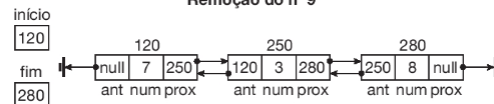
4ª operação
Inserção do nº 3 no fim da lista



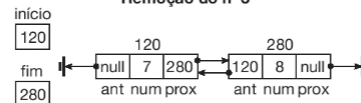
5ª operação
Inserção do nº 8 no fim da lista



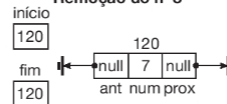
6ª operação
Remoção do nº 9



7ª operação
Remoção do nº 3



8ª operação
Remoção do nº 8




Lista duplamente encadeada

```
#include <iostream.h>
#include <conio.h>

void main()
{
    //Definindo o registro representará
    //cada elemento da lista
    struct LISTA
    {
        int num;
        LISTA *prox;
        LISTA *ant;
    };

    // a lista está vazia, logo,
    // o ponteiro inicio tem o valor NULL
    // o ponteiro inicio conterá o endereço
    // do primeiro elemento da lista
    LISTA *inicio = NULL;
    // o ponteiro fim conterá o endereço
    // do último elemento da lista
    LISTA *fim = NULL;
    // o ponteiro aux é um ponteiro auxiliar
    LISTA *aux;
    // apresentando o menu de opções
    int op, numero, achou;
    do
    {
        clrscr();
        cout<<"\nMENU DE OPÇÕES\n";
        cout<<"\n1 - Inserir no início da
        ↳ lista";
```



```
        cout<<"\n2 - Inserir no fim da lista";
        cout<<"\n3 - Consultar a lista do
        ↳ início ao fim";
        cout<<"\n4 - Consultar a lista do fim
        ↳ ao início";
        cout<<"\n5 - Remover da lista";
        cout<<"\n6 - Esvaziar a lista";
        cout<<"\n7 - Sair";
        cout<<"\nDigite sua opção: ";
        cin>>op;
        if (op < 1 || op > 7)
            cout<<"Opção inválida!!";
        if (op == 1)
        {
            cout<<"Digite o número a ser inserido
            ↳ no início da lista: ";
            LISTA *novo = new LISTA();
            cin>>novo->num;
            if (inicio == NULL)
            {
                // a lista estava vazia
                // e o elemento inserido será
                // o primeiro e o último
                inicio = novo;
                fim = novo;
                novo->prox = NULL;
                novo->ant = NULL;
```

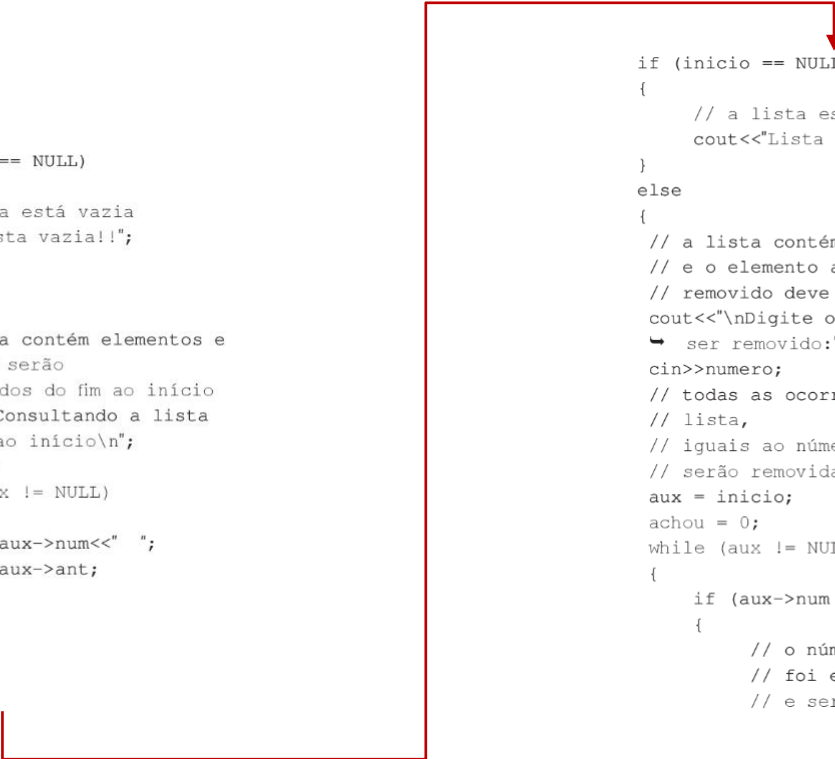
Lista duplamente encadeada

```
}
else
{
    // a lista já contém elementos
    // e o novo elemento
    // será inserido no início da lista
    novo->prox = inicio;
    inicio->ant = novo;
    novo->ant = NULL;
    inicio = novo;
}
cout<<"Número inserido no início da lista!!";
}
if (op == 2)
{
    cout<<"Digite o número a ser
    ↳ inserido no fim da lista: ";
    LISTA *novo = new LISTA();
    cin>>novo->num;
    if (inicio == NULL)
    {
        // a lista estava vazia
        // e o elemento inserido será
        // o primeiro e o último
        inicio = novo;
        fim = novo;
        novo->prox = NULL;
        novo->ant = NULL;
    }
}
```

```
else
{
    // a lista já contém elementos
    // e o novo elemento
    // será inserido no fim da lista
    fim->prox = novo;
    novo->ant = fim;
    novo->prox = NULL;
    fim = novo;
}
cout<<"Número inserido no fim da
↳ lista!!";
}
if (op == 3)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
        // a lista contém elementos e
        // estes serão
        // mostrados do início ao fim
        cout<<"\nConsultando a lista
        ↳ do início ao fim\n";
        aux = inicio;
        while (aux != NULL)
        {
            cout<<aux->num<<" ";
            aux = aux->prox;
        }
    }
}
```

Lista duplamente encadeada

```
if (op == 4)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
        // a lista contém elementos e
        // estes serão
        // mostrados do fim ao início
        cout<<"\nConsultando a lista
        ↳ do fim ao início\n";
        aux = fim;
        while (aux != NULL)
        {
            cout<<aux->num<<" ";
            aux = aux->ant;
        }
    }
}
if (op == 5)
{
```



```
if (inicio == NULL)
{
    // a lista está vazia
    cout<<"Lista vazia!!";
}
else
{
    // a lista contém elementos
    // e o elemento a ser
    // removido deve ser digitado
    cout<<"\nDigite o elemento a
    ↳ ser removido:";
    cin>>numero;
    // todas as ocorrências da
    // lista,
    // iguais ao número digitado,
    // serão removidas
    aux = inicio;
    achou = 0;
    while (aux != NULL)
    {
        if (aux->num == numero)
        {
            // o número digitado
            // foi encontrado na lista
            // e será removido
```

Lista duplamente encadeada

```
achou = achou + 1;
if (aux == inicio)
{
    // o número a
    // ser removido
    // é o primeiro da
    // lista
    inicio = aux->prox;
    if (inicio != NULL)
    {
        inicio->ant = NULL;
    }
    delete (aux);
    aux = inicio;
}
else if (aux == fim)
{
    // o número a ser
    // removido
    // é o último da
    // lista
    fim = fim->ant;
    fim->prox = NULL;
    delete (aux);
    aux = NULL;
}
```

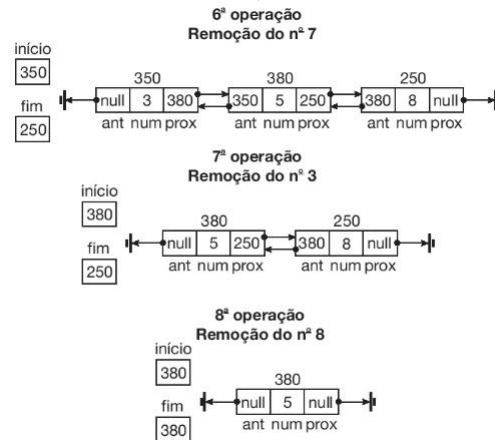
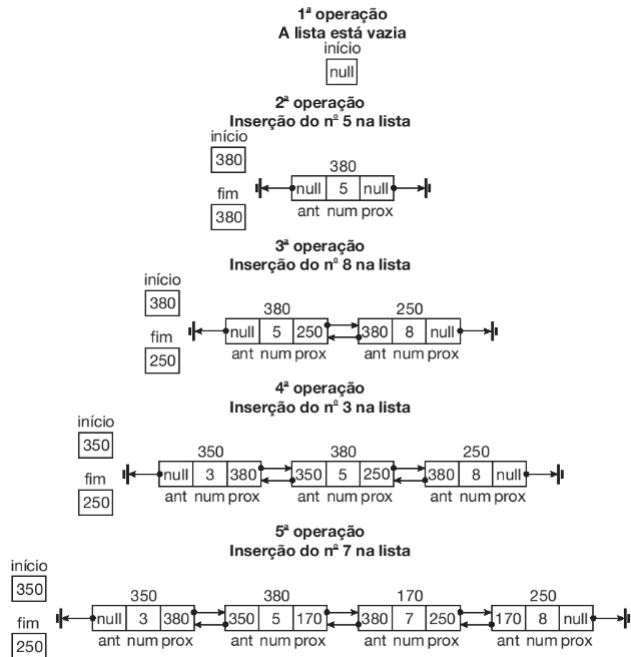
```
    else {
        // o número a ser
        // removido
        // está no meio
        // da lista
        aux->ant->prox =
            ↪ aux->prox;
        aux->prox->ant =
            ↪ aux->ant;
        LISTA *aux2;
        aux2 = aux->prox;
        delete(aux);
        aux = aux2;
    }
    else
    {
        aux = aux->prox;
    }
}
if (achou == 0)
    cout<<"Número não encontrado";
else if (achou == 1)
    cout<<"Número removido 1 vez";
else
    cout<<"Número removido
    ↪ "<<achou<<" vezes";
}
```

```
}
```

Lista duplamente encadeada

```
if (op == 6)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
        // a lista será esvaziada
        aux = inicio;
        while (aux != NULL)
        {
            inicio = inicio->prox;
            delete(aux);
            aux = inicio;
        }
        cout<<"Lista esvaziada";
    }
}
getch();
}
while (op != 7);
}
```

Lista dupla enc. ordenada



Lista dup. ord.

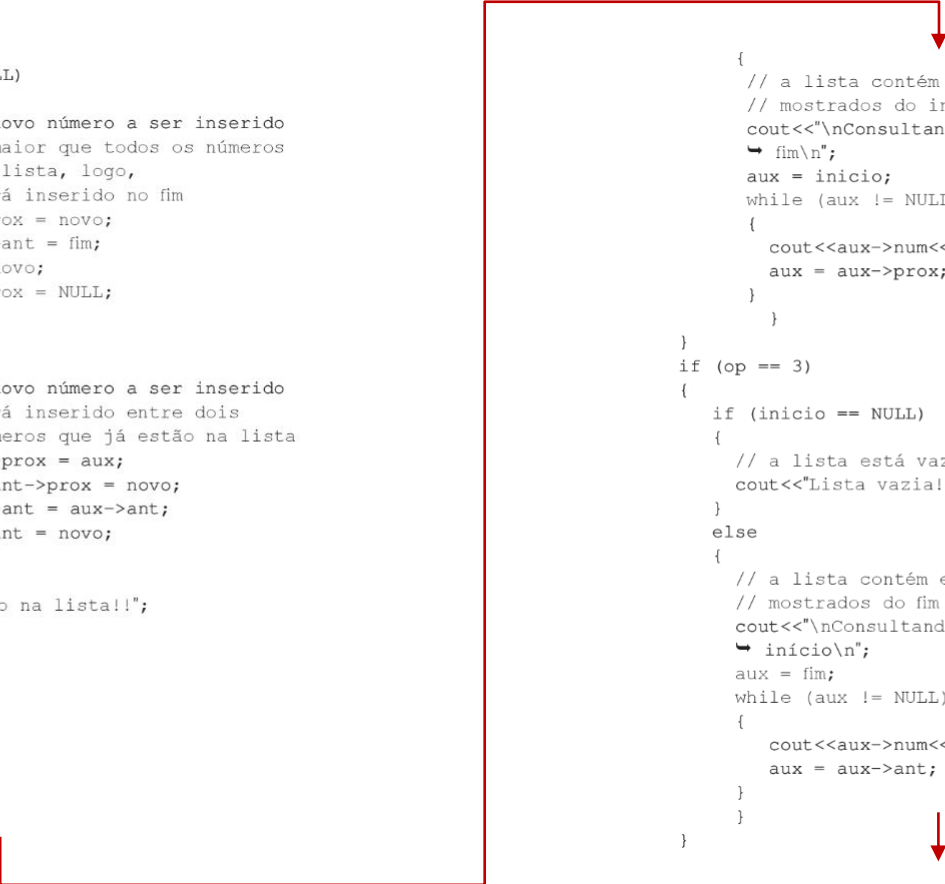
```
#include <iostream.h>
#include <conio.h>

void main()
{
    //Definindo o registro que representará
    //cada elemento da lista
    struct LISTA
    {
        int num;
        LISTA *prox;
        LISTA *ant;
    };
    // a lista está vazia, logo,
    // o ponteiro inicio tem o valor null
    // o ponteiro inicio conterá o endereço
    // do primeiro elemento da lista
    LISTA *inicio = NULL;
    // o ponteiro fim conterá o endereço
    // do último elemento da lista
    LISTA *fim = NULL;
    // o ponteiro aux é um ponteiro auxiliar
    LISTA *aux;
    // apresentando o menu de opções
    int op, numero, achou;
    do
    {
        clrscr();
        cout<<"\nMENU DE OPÇÕES\n";
        cout<<"\n1 - "Inserir na lista";
        cout<<"\n2 - Consultar a lista do início ao fim";
        cout<<"\n3 - Consultar a lista do fim ao início";
        cout<<"\n4 - Remover da lista";
        cout<<"\n5 - Esvaziar a lista";
        cout<<"\n6 - Sair";
        cout<<"\nDigite sua opção: ";
        cin>>op;
```

```
if (op < 1 || op > 6)
    cout<<"Opção inválida!!";
if (op == 1)
{
    cout<<"Digite o número a ser inserido na
    ↳ lista: ";
    LISTA *novo = new LISTA();
    cin>>novo->num;
    if (inicio == NULL)
    {
        // a lista estava vazia
        // e o elemento inserido será
        // o primeiro e o último
        novo->prox = NULL;
        novo->ant = NULL;
        inicio = novo;
        fim = novo;
    }
    else
    {
        // a lista já contém elementos
        // e o novo elemento
        // será inserido na lista
        // respeitando a ordenação crescente
        aux = inicio;
        while (aux != NULL && novo->num > aux->num)
        {
            aux = aux->prox;
        }
        if (aux == inicio)
        {
            // o novo número a ser inserido
            // é menor que todos os números da lista,
            // logo, será inserido no início
            novo->prox = inicio;
            novo->ant = NULL;
            inicio->ant = novo;
            inicio = novo;
        }
    }
}
```

Lista dupla enc. ordenada

```
else if (aux == NULL)
{
    // o novo número a ser inserido
    // é maior que todos os números
    // da lista, logo,
    // será inserido no fim
    fim->prox = novo;
    novo->ant = fim;
    fim = novo;
    fim->prox = NULL;
}
else
{
    // o novo número a ser inserido
    // será inserido entre dois
    // números que já estão na lista
    novo->prox = aux;
    aux->ant->prox = novo;
    novo->ant = aux->ant;
    aux->ant = novo;
}
}
cout<<"Número inserido na lista!!";
}
if (op == 2)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
```



```
{
    // a lista contém elementos e estes serão
    // mostrados do início ao fim
    cout<<"\nConsultando a lista do início ao
    fim\n";
    aux = inicio;
    while (aux != NULL)
    {
        cout<<aux->num<<" ";
        aux = aux->prox;
    }
}
}
if (op == 3)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
        // a lista contém elementos e estes serão
        // mostrados do fim ao início
        cout<<"\nConsultando a lista do fim ao
        início\n";
        aux = fim;
        while (aux != NULL)
        {
            cout<<aux->num<<" ";
            aux = aux->ant;
        }
    }
}
```

Lista dupla enc. ordenada

```
if (op == 4)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
        // a lista contém elementos
        // e o elemento a ser
        // removido deve ser digitado
        cout<<"\nDigite o elemento a ser
        ↳ removido: ";
        cin>>numero;
        // todas as ocorrências da lista,
        // iguais ao número digitado,
        // serão removidas
        aux = inicio;
        achou = 0;
        while (aux != NULL)
        {
            if (aux->num == numero)
            {
                // o número digitado
                // foi encontrado na lista
                // e será removido
                achou = achou + 1;
```

```
if (aux == inicio)
{
    // o número a ser
    // removido
    // é o primeiro da
    // lista
    inicio = aux->prox;
    if (inicio != NULL)
    {
        inicio->ant = NULL;
    }
    delete(aux);
    aux = inicio;
}
else if (aux == fim)
{
    // o número a ser
    // removido
    // é o último da lista
    fim = fim->ant;
    fim->prox = NULL;
    delete(aux);
    aux = NULL;
}
else
{
    // o número a ser
    // removido
    // está no meio da
    // lista
    aux->ant->prox = aux-
    ↳ >prox;
    aux->prox->ant = aux-
    ↳ >ant;
```

Lista dupla enc. ordenada

```
        LISTA *aux2;
        aux2 = aux->prox;
        delete(aux);
        aux=aux2;
    }
    else
    {
        aux = aux->prox;
    }
}

if (achou == 0)
cout<<"Número não encontrado";
else if (achou == 1)
    cout<<"Número removido 1 vez";
else
    cout<<"Número removido "<<achou<<"
    ↳ vezes";
}
}

if (op == 5)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }

    LISTA *aux2;
    aux2 = aux->prox;
    delete(aux);
    aux=aux2;
}
else
{
    // a lista será esvaziada
    aux = inicio;
    while (aux != NULL)
    {
        inicio = inicio->prox;
        delete(aux);
        aux = inicio;
    }
    cout<<"Lista esvaziada";
}

}

getch();
}

while (op != 6);
}
```

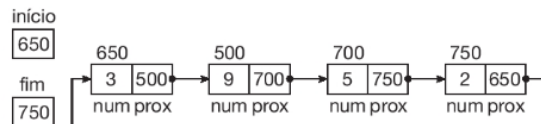


Lista circular

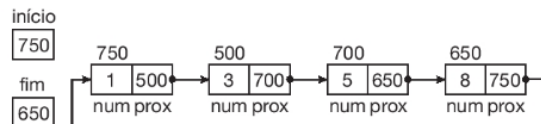
- Caso especial de lista dupla
 - ▶ O ponteiro **prox** do “último” elemento aponta para o “primeiro” elemento da lista.
 - ▶ O ponteiro **prev** do “primeiro” elemento aponta para o “último” elemento da lista.

Lista circular

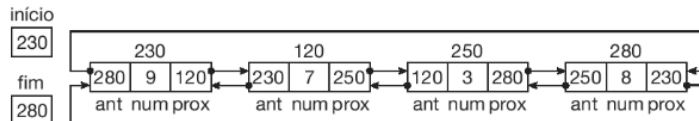
Lista circular simplesmente encadeada e não ordenada



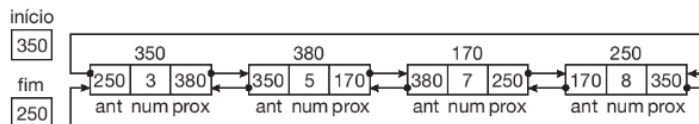
Lista circular simplesmente encadeada e ordenada



Lista circular duplamente encadeada e não ordenada



Lista circular duplamente encadeada e ordenada




Lista circular simp. enc. não ord.

```
#include<iostream.h>
#include<conio.h>

void main()
{
    //Definindo o registro que representará
    //cada elemento da lista
    struct LISTA
    {
        int num;
        LISTA *prox;
    };

    // a lista está vazia, logo,
    // o ponteiro inicio têm o valor null
    // o ponteiro inicio conterá o endereço
    // do primeiro elemento da lista
    LISTA *inicio = NULL;
    // o ponteiro fim conterá o endereço
    // do último elemento da lista
    LISTA *fim = NULL;
    // o ponteiro aux é um ponteiro auxiliar
    LISTA *aux;
    // o ponteiro anterior é um ponteiro auxiliar
    LISTA *anterior;
    // apresentando o menu de opções
    int op, numero, achou;
    do
    {
        clrscr();
        cout<<"\nMENU DE OPÇÕES\n";
        cout<<"\n1 - Inserir no início da lista";
```



```
        cout<<"\n2 - Inserir no fim da lista";
        cout<<"\n3 - Consultar toda a lista";
        cout<<"\n4 - Remover da lista";
        cout<<"\n5 - Esvaziar a lista";
        cout<<"\n6 - Sair";
        cout<<"\nDigite sua opção: ";
        cin>>op;
        if (op < 1 || op > 6)
            cout<<"Opção inválida!";
        if (op == 1)
        {
            cout<<"Digite o número a ser inserido no
            ↳ início da lista: ";
            LISTA *novo = new LISTA();
            cin>>novo->num;
            if (inicio == NULL)
            {
                // a lista estava vazia
                // e o elemento inserido será
                // o primeiro e o último
                inicio = novo;
                fim = novo;
                fim->prox = inicio;
            }
            else
            {
                // a lista já contém elementos
                // e o novo elemento
                // será inserido no início da lista
```

Lista circular simp. enc. não ord.

```
    novo->prox = inicio;
    inicio = novo;
    fim->prox = inicio;
}
cout<<"Número inserido no início da
↳ lista!!";
}
if (op == 2)
{
    cout<<"Digite o número a ser inserido
↳ no fim da lista: ";
    LISTA *novo = new LISTA();
    cin>>novo->num;
    if (inicio == NULL)
    {
        // a lista estava vazia
        // e o elemento inserido será
        // o primeiro e o último
        inicio = novo;
        fim = novo;
        fim->prox = inicio;
    }
    else
    {
        // a lista já contém elementos
        // e o novo elemento
        // será inserido no fim da lista
        fim->prox = novo;
        fim = novo;
        fim->prox = inicio;
    }
}
```

```
    cout<<"Número inserido no fim da
↳ lista!!";
}
if (op == 3)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
        // a lista contém elementos
        // e estes serão mostrados
        // do início ao fim
        cout<<"\nConsultando toda a
↳ lista\n";
        aux = inicio;
        do
        {
            cout<<aux->num<<" ";
            aux = aux->prox;
        }
        while (aux != inicio);
    }
}
if (op == 4)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
}
```


Lista circular simp. enc. não ord.

```
else
{
    // a lista contém elementos
    // e o elemento a ser
    // removido deve ser digitado
    cout<<"\nDigite o elemento a ser
    ↪ removido:";
    cin>>numero;
    // todas as ocorrências da lista,
    // iguais ao número digitado,
    // serão removidas
    aux = inicio;
    anterior = NULL;
    achou = 0;
    // descobrindo a quantidade de
    // elementos da lista
    int quantidade = 0;
    aux = inicio;
    do
    {
        quantidade = quantidade + 1;
        aux = aux->prox;
    }
    while (aux != inicio);
```

```
int elemento = 1;
do
{
    // se a lista possui apenas
    // um elemento
    if (inicio == fim &&
        inicio->num == numero)
    {
        delete(inicio);
        inicio = NULL;
        achou = achou + 1;
    }
    else
    {
        if (aux->num == numero)
        {
            // o número digitado
            // foi encontrado na lista
            // e será removido
            achou = achou + 1;
            if (aux == inicio)
            {
                // o número a ser
                // removido
                // é o primeiro da lista
                inicio = aux->prox;
                fim->prox = inicio;
                delete(aux);
                aux = inicio;
            }
        }
    }
}
```

Lista circular simp. enc. não ord.

```
else if (aux == fim)
{
    // o número a ser
    // removido
    // é o último da
    // lista
    fim = anterior;
    fim->prox = inicio;
    delete(aux);
    aux = NULL;
}
else
{
    // o número a ser
    // removido
    // está no meio da
    // lista
    anterior->prox =
        aux->prox;
    delete(aux);
    aux = anterior-
        >prox;
}
}
else
{
    anterior = aux;
    aux = aux->prox;
}
}
```

```
elemento = elemento + 1;
}
while (elemento <= quantidade);
if (achou == 0)
    cout<<"Número não encontrado";
else if (achou == 1)
    cout<<"Número removido 1 vez";
else
    cout<<"Número removido "<<achou<<"
        vezes";
}
}
if (op == 5)
{
    if (inicio == NULL)
    {
        // a lista está vazia
        cout<<"Lista vazia!!";
    }
    else
    {
        // a lista será esvaziada
        aux = inicio;
        do
        {
            inicio = inicio ->prox;
            delete(aux);
        }
```

Lista circular simp. enc. não ord.

```
        aux = inicio;
    }
    while (aux != fim);
    delete(fim);
    inicio=NULL;
    cout<<"Lista esvaziada";
}
}
getch();
}
while (op != 6);
}
```

Perguntas?

Bibliografia da aula

- Notas de aula do Prof. Edson L.F. Senne (UNESP/INPE) em 2010.
- ASCENCIO, A. F. G.; ARAÚJO, G. S. Estrutura de dados. Algoritmos, análise da complexidade e implementação em Java e C/C++. Pearson. 2010.