# Algoritmos
# e
# Estrutura de Dados

**Profª. Angela Abreu Rosa de Sá, Drª.**

*Contato: angelaabreu@gmail.com*

# Lembrar ...

**É um processo INDIVIDUAL!**

**Depende apenas de você!**

IMPORTANTE

- Não se compare com o colega;
- Pergunte QUALQUER dúvida;
- Implemente os exercícios propostos;
- Pratique sempre...inclusive em casa!
- Não desista!

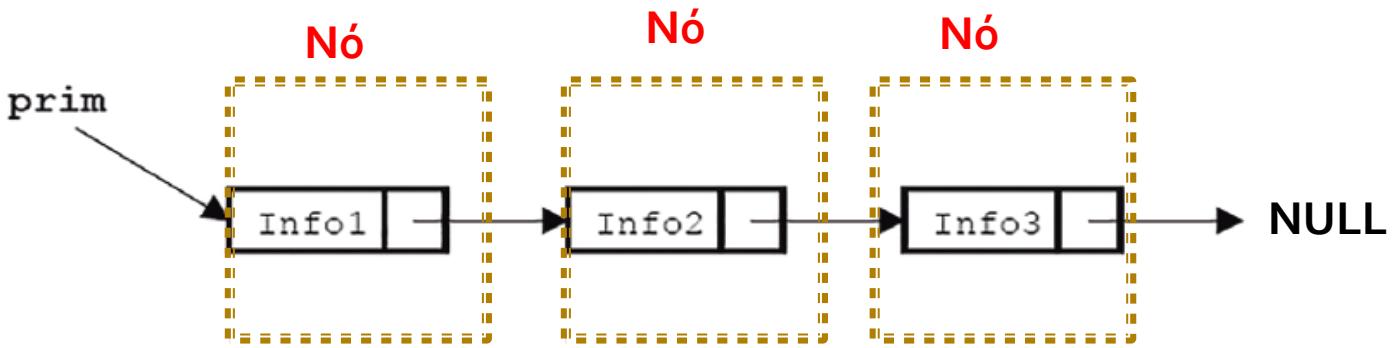
# Ementa


1
Listas Ligadas
Definição
Operações
Listas duplamente ligadas
2
Pilhas e filas
Definição
Operações com pilhas
Operações com filas
3
Tabelas de espalhamento
Definição
Operações
Otimização
4
Armazenamento associativo
Definição
Mapas com lista
Mapas com espalhamento

# Algoritmos e Estrutura de Dados

## Sumário

# Passo 1: criar a estrutura dos nós



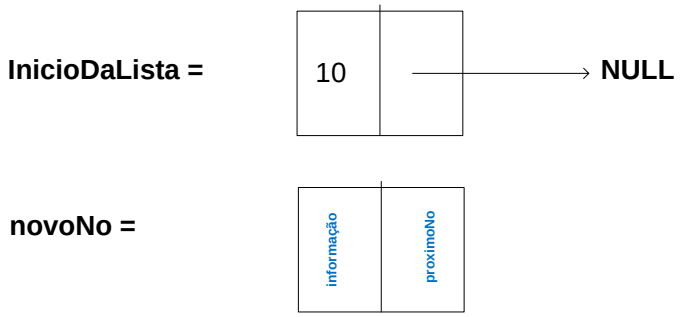

```
struct noLista
{
   int informacao;
   struct noLista *proximo;
};
```

# Inserir no INÍCIO da Lista

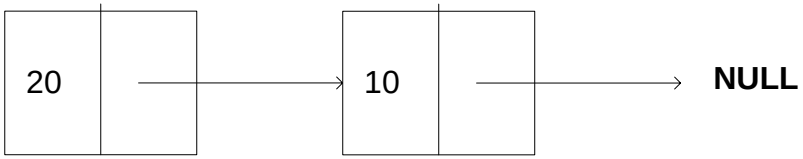
```c
struct noLista * inserirInicioLista (struct noLista *InicioDaLista, int NovoNumero)
 {
    //alocar memória para o novo Nó da Lista
    struct noLista* novoNo = (struct noLista*) malloc(sizeof(struct noLista));

    //Inserir as informação para o novo nó
    novoNo->informacao = NovoNumero;
    //Apontar o campo "próximo" do novo nó para o local que o InicioDaLista apontava
    novoNo->proximoNo = InicioDaLista;

    return novoNo;
}



int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;

    //inserir a informaçao "10" na lista
   //---- o Primeiro da lista irá apontar agora para o novo nó que foi criado
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 10);

}
```

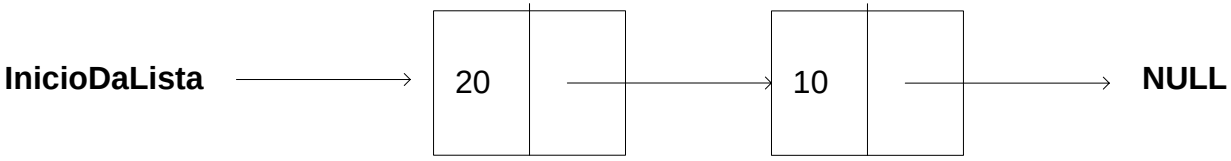
# Inserir no INÍCIO da Lista

```c
int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;

    //inserir a informaçao "10" na lista
   //---- o Primeiro da lista irá apontar agora para o novo nó que foi criado
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 10);

    //inserir a informaçao "20" na lista
    //---- o Primeiro da lista irá apontar agora para o novo nó que foi criado
    PrimeiroNoDaLista= inserirInicioLista (PrimeiroNoDaLista, 20);

}

    ImprimirLista(PrimeiroNoDaLista);
```




**InicioDaLista = novoNo**



# Imprimir a Lista

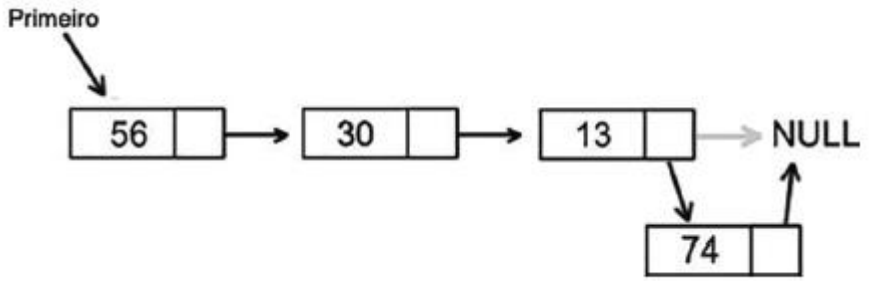
```c
void ImprimirLista(struct noLista  *InicioDaLista)
{
    noLista *NoAtual = InicioDaLista; //copiar o endereço do primeiro nó da lista

    while (NoAtual != NULL) //percorrer a lista até encontrar o último nó = NULL
    {
        printf("%d -> ",NoAtual->informacao); //mostrar a informação do nó
        NoAtual = NoAtual->proximoNo; //apotar para o próximo nó da lista
    }

    printf("NULL");

}
```

```c
int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;

    //inserir a informaçao "10" na lista
   //---- o Primeiro da lista irá apontar agora para o novo nó que foi criado
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 10);

    //inserir a informaçao "20" na lista
    //---- o Primeiro da lista irá apontar agora para o novo nó que foi criado
    PrimeiroNoDaLista= inserirInicioLista (PrimeiroNoDaLista, 20);

   //imprimir o conteúdo da lista
   ImprimirLista(PrimeiroNoDaLista);

}
```

```
20 -> 10 -> NULL
--------------------------------
```

# Inserir no FINAL da Lista

```
struct noLista* inserirFimLista(struct noLista* InicioLista, int NovoNumero)
{
    struct noLista *PercorreLista = InicioLista; //cópia do início da lista
    struct noLista* novoNo = (struct noLista*)malloc(sizeof(struct noLista)); //
    while (PercorreLista->proximoNo != NULL) //percorrer a lista até encontrar o último nó
    {
      PercorreLista = PercorreLista->proximoNo;

    }
    novoNo->informacao = NovoNumero; //inserir a nova informação
    novoNo->proximoNo = PercorreLista->proximoNo; //apontar para o próximo nó que o último estava apontando (NULL)
    PercorreLista->proximoNo = novoNo; // o que era o último nó (agora será o penúltimo!) aponta para o novo nó

    return InicioLista;
}
```

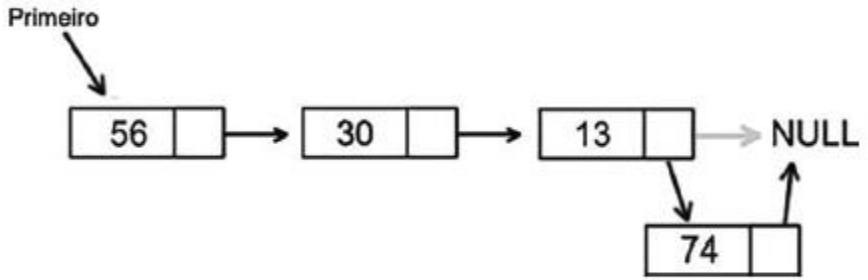



Elaborada pelo autor

# Inserir no FINAL da Lista

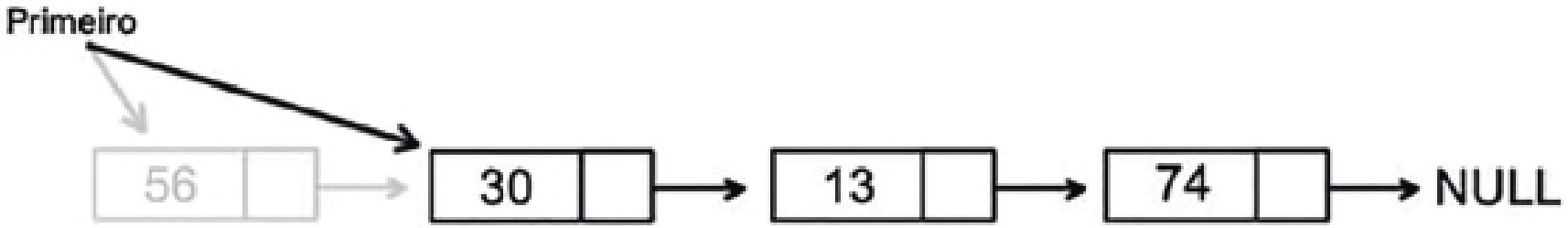
```
int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;

    //inserir a informaçao "10" na lista
   //---- o Primeiro da lista irá apontar agora para o novo nó que foi criado
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 13);

    //inserir a informaçao "20" na lista
    //---- o Primeiro da lista irá apontar agora para o novo nó que foi criado
    PrimeiroNoDaLista= inserirInicioLista (PrimeiroNoDaLista, 56);

    //inserir o número 30 após a posição 0
    PrimeiroNoDaLista= inserirPosicaoLista (PrimeiroNoDaLista, 30,0) ;

    //inserir o número 74 no fim da lista
    PrimeiroNoDaLista =  inserirFimLista(PrimeiroNoDaLista, 74);

   //imprimir o conteúdo da lista
   ImprimirLista(PrimeiroNoDaLista);

}
```

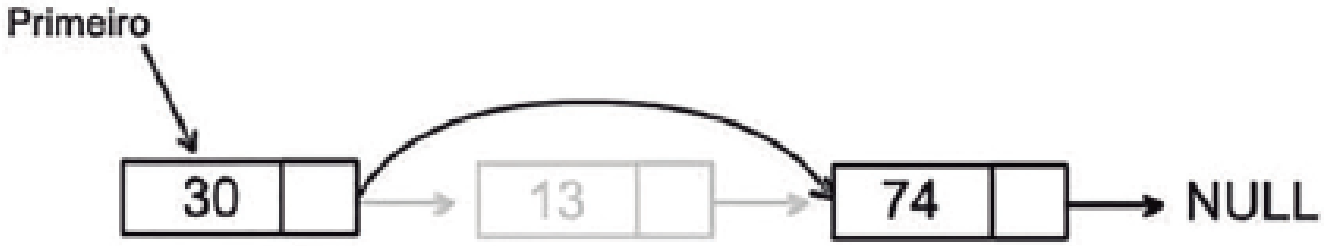
```
56 -> 30 -> 13 -> 74 -> NULL
--------------------------------
```




elaborada pelo autor

# REMOVER elementos da lista



# REMOVER elementos da lista

```c
struct noLista* remove (struct noLista* InicioLista, int ElementoParaRemover)
{
    struct noLista* NoAnterior = NULL; //ponteiro para o nó anterior
    struct noLista* PercorreLista = InicioLista; //cópia do início da lista

    //procurar o elementou ou até chegar no final da lista
    while (PercorreLista != NULL && PercorreLista->informacao != ElementoParaRemover)
    {
        NoAnterior = PercorreLista; //manter uma cópia do endereço do nó anterior
        PercorreLista = PercorreLista->proximoNo; //percorrer a lista
    }

    if (PercorreLista == NULL ) //não encontrou o número na lista para ser removido
        return InicioLista;

    if (NoAnterior == NULL) //significa que o elemento procurado é  o primeiro da lista
    {
        InicioLista = PercorreLista->proximoNo; //atualizar o início da lista
    }
    else {
            NoAnterior->proximoNo = PercorreLista->proximoNo; //"ligar" o próximo nó do anterior com o próximo do elemento que será removido
        }

    free(PercorreLista); //liberar região de memória do nó removido

  return InicioLista;
}
```

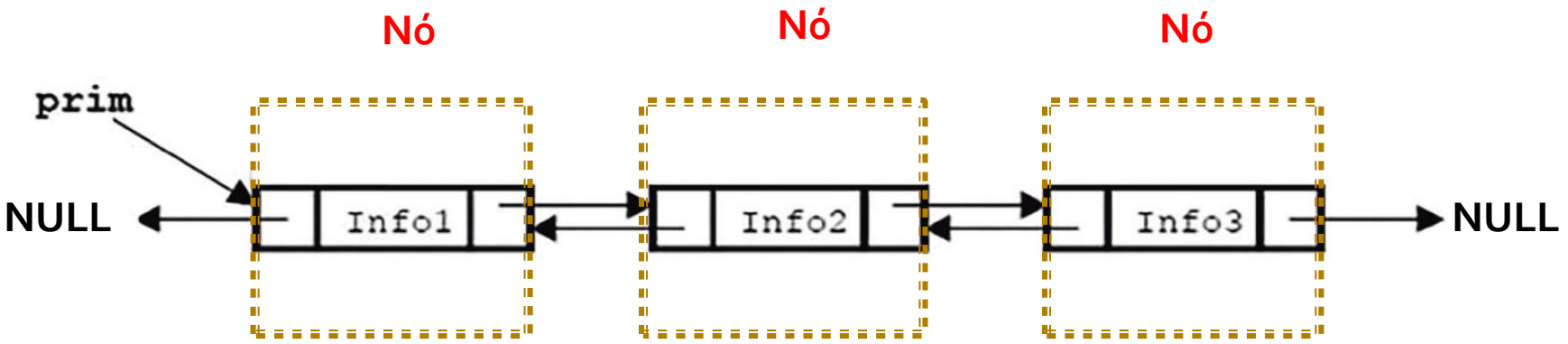
# PROCURAR elementos na lista

```c
struct noLista* ProcurarElementoLista(struct noLista* InicioLista, int ElementoProcurado)
{
    struct noLista* PercorreLista;
    PercorreLista = InicioLista;//copia do inicio da lista

    while(PercorreLista!= NULL) //enquanto existir nó na lista
    {
      if (PercorreLista->informacao == ElementoProcurado) //se encontrou o elemento procurado
        return PercorreLista; //retorna o nó encontrado

      else PercorreLista = PercorreLista->proximoNo; //continuar percorrendo a lista
    }

    //se saiu o while, é porque não encontrou o número! Retornar NULL;
    return NULL;
}
```

```c
//imprimir o conteúdo da lista
ImprimirLista(PrimeiroNoDaLista);


if (ProcurarElementoLista(PrimeiroNoDaLista, 30) == NULL)
 {
    printf("\n Elemento não encontrado");
}
 else
 {
   printf("\n Elemento encontrado");
 }
```

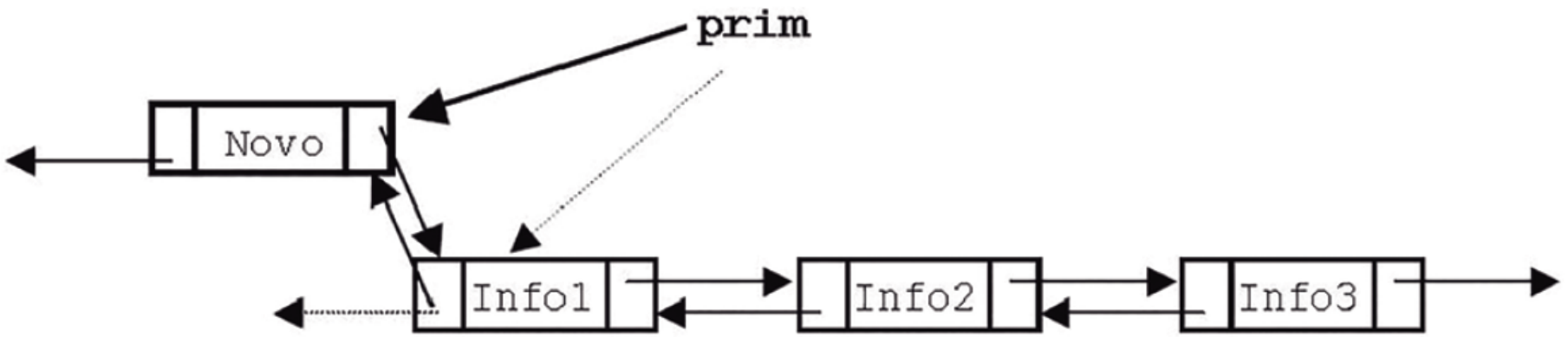
```
30 -> 13 -> NULL
 Elemento encontrado
--------------------------------
```

# Lista DUPLAMENTE Encadeada




```
struct noLista
{
   struct noLista *NoAnterior;
   int informacao;
   struct noLista *proximoNo;
};
```

# Inserir no INÍCIO da Lista DUPLAMENTE Encadeada



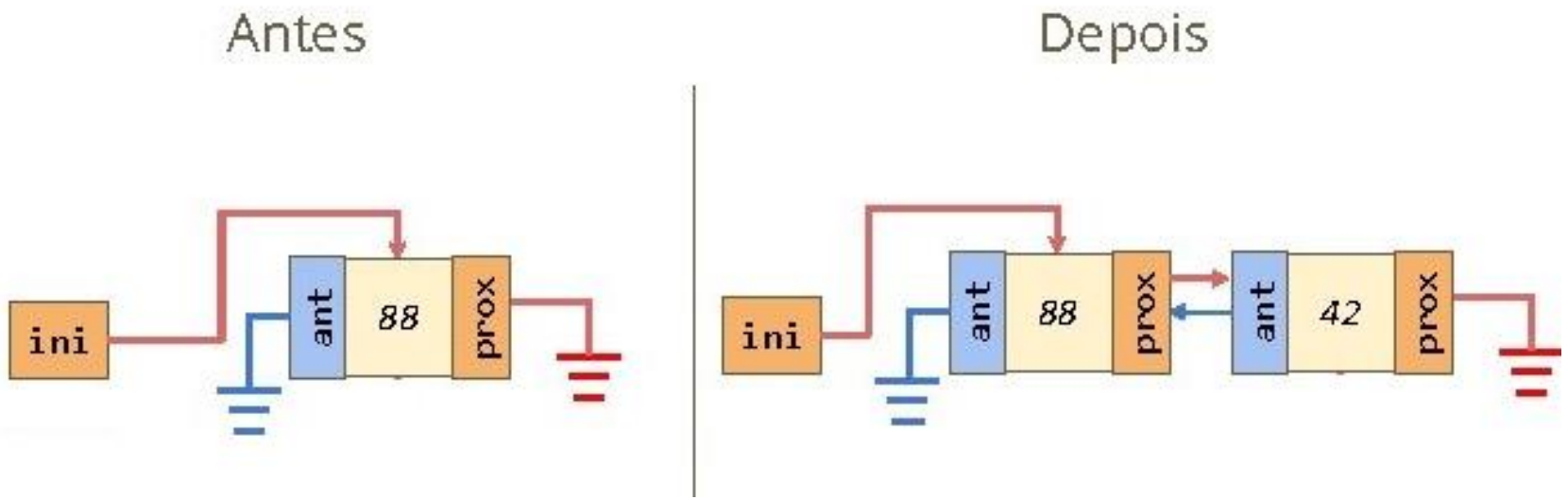
# Inserir no INÍCIO da Lista DUPLAMENTE Encadeada

```c
struct noLista * inserirInicioLista (struct noLista *InicioDaLista, int NovoNumero)
 {
    //alocar memória para o novo Nó da Lista
    struct noLista* novoNo = (struct noLista*) malloc(sizeof(struct noLista));

    //O campo NoAnterior será NULL, pois agora ele será o primeiro da lista
    novoNo->NoAnterior = NULL;
    //Inserir as informação para o novo nó
    novoNo->informacao = NovoNumero;
    //Apontar o campo "próximo" do novo nó para o local que o InicioDaLista apontava
    novoNo->proximoNo = InicioDaLista;


    if(InicioDaLista != NULL) // se a lista não estiver vazia, ligar o nó anterior do início da lista ao novo nó
       InicioDaLista->NoAnterior = novoNo;


    return novoNo;
}
```

```c
int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;

    //inserir a informaçao 88 no início da lista
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 88);
    //inserir a informaçao 42 no início da lista
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 42);


    //imprimir o conteúdo da lista
   ImprimirLista(PrimeiroNoDaLista);


}
```

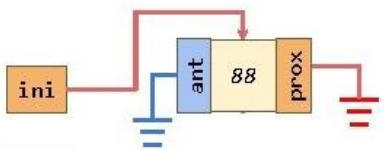
```
42 -> 88 -> NULL
--------------------
```

# Inserir no FINAL da Lista DUPLAMENTE Encadeada

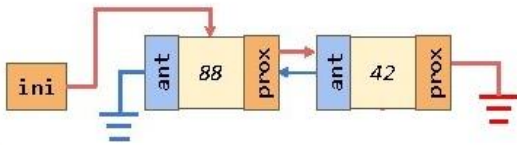


```c
struct noLista* inserirFimLista(struct noLista* InicioLista, int NovoNumero)
{
    struct noLista *PercorreLista = InicioLista; //cópia do início da lista
    struct noLista* novoNo = (struct noLista*)malloc(sizeof(struct noLista)); // alocar memória para o novo nó

    if(PercorreLista != NULL) //se a lista não estiver vazia
    {

        while (PercorreLista->proximoNo != NULL) //percorrer a lista até encontrar o último nó
        {
          PercorreLista = PercorreLista->proximoNo;

        }

        //o que era o último nó, agora será o penúltimo: aponta para o novoNo
        PercorreLista->proximoNo = novoNo; // o que era o último nó (agora será o penúltimo!) aponta para o novo nó

   }

    novoNo->NoAnterior = PercorreLista; //apontar o campo NoAnterior para o nó PercorreLista
    novoNo->informacao = NovoNumero; //inserir a nova informação
    novoNo->proximoNo = NULL; //o novoNo será o último da lista


    if(InicioLista == NULL)//a lista estava vazia
        return novoNo;
    else return InicioLista;
}
```

Antes



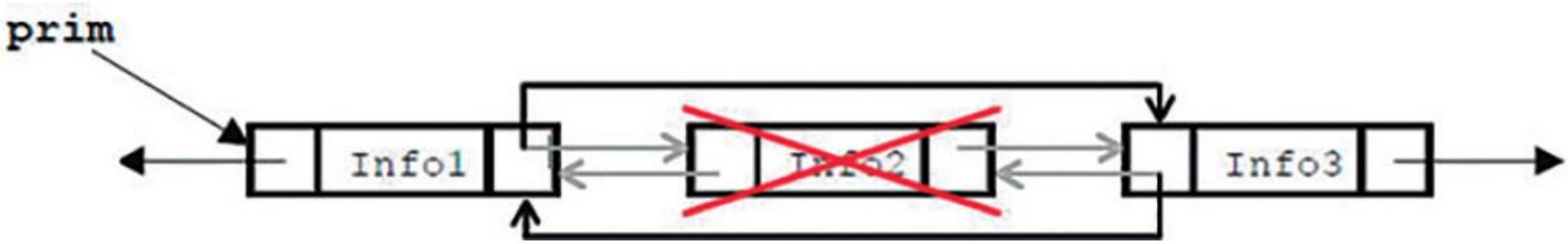

Depois




```c
int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;


    //inserir a informaçao 88 no fim da lista
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 88);
    //inserir a informaçao 42 no fim da lista
   PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 42);


    //imprimir o conteúdo da lista
   ImprimirLista(PrimeiroNoDaLista);

}
```

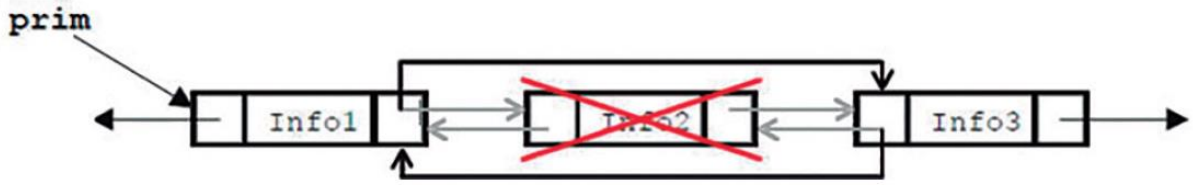
```
88 -> 42 -> NULL
```

# Remover da Lista DUPLAMENTE Encadeada



```c
struct noLista* remove (struct noLista* InicioLista, int ElementoParaRemover)
{
    struct noLista* PercorreLista = InicioLista; //cópia do início da lista

    //procurar o elementou ou até chegar no final da lista
    while (PercorreLista != NULL && PercorreLista->informacao != ElementoParaRemover)
    {
        PercorreLista = PercorreLista->proximoNo; //percorrer a lista
    }

    if (PercorreLista == NULL ) //não encontrou o número na lista para ser removido
        return InicioLista;


    if (PercorreLista->NoAnterior == NULL) //significa que o elemento procurado é  o primeiro da lista
    {
        InicioLista = PercorreLista->proximoNo; //atualizar o início da lista
    }
    else {  //"ligar" o próximo nó do anterior com o próximo do elemento que será removido
          PercorreLista->NoAnterior->proximoNo = PercorreLista->proximoNo;
          }

    if(PercorreLista->proximoNo != NULL) //se não for o último da lista
    {
        //ligar o  o campo "noAnterior" do proximo nó ao nó anterior daquele que será removido
        PercorreLista->proximoNo->NoAnterior = PercorreLista->NoAnterior;
    }
    free(PercorreLista); //liberar região de memória do nó removido

  return InicioLista;
}
```

```c
int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;


    //inserir a informaçao 88 no fim da lista
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 88);
    //inserir a informaçao 42 no fim da lista
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 42);

    //inserir a informaçao 88 no fim da lista
    PrimeiroNoDaLista = inserirFimLista (PrimeiroNoDaLista, 53);

    //inserir a informaçao 42 no fim da lista
    PrimeiroNoDaLista = inserirFimLista (PrimeiroNoDaLista, 44);

  PrimeiroNoDaLista = remove (PrimeiroNoDaLista, 42);

    //imprimir o conteúdo da lista
  ImprimirLista(PrimeiroNoDaLista);
```




```
88 -> 53 -> 44 -> NULL
--------------------------------
```

# Procurar na Lista DUPLAMENTE Encadeada

```c
struct noLista* ProcurarElementoLista(struct noLista* InicioLista, int ElementoProcurado)
{
    struct noLista* PercorreLista;
    PercorreLista = InicioLista;//copia do inicio da lista

    while(PercorreLista!= NULL) //enquanto existir nó na lista
    {
      if (PercorreLista->informacao == ElementoProcurado) //se encontrou o elemento procurado
        return PercorreLista; //retorna o nó encontrado

      else PercorreLista = PercorreLista->proximoNo; //continuar percorrendo a lista
    }

    //se saiu o while, é porque não encontrou o número! Retornar NULL;
    return NULL;
}
```

```c
//inserir a informaçao 88 no fim da lista
PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 88);
//inserir a informaçao 42 no fim da lista
PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 42);

//inserir a informaçao 88 no fim da lista
PrimeiroNoDaLista = inserirFimLista (PrimeiroNoDaLista, 53);

//inserir a informaçao 42 no fim da lista
PrimeiroNoDaLista = inserirFimLista (PrimeiroNoDaLista, 44);

PrimeiroNoDaLista = remove (PrimeiroNoDaLista, 42);


if (ProcurarElementoLista(PrimeiroNoDaLista, 53) == NULL)
{
   printf("\n Elemento não encontrado");
}
else
{
  printf("\n Elemento encontrado");
}
```

```
Elemento encontrado
--------------------------------
```

# Programa completo...

```c
#include <stdio.h>
#include <stdlib.h>

struct noLista
{
  struct noLista *NoAnterior;
  int informacao;
  struct noLista *proximoNo;
};


struct noLista* ProcurarElementoLista(struct noLista* InicioLista, int ElementoProcurado)
{
    struct noLista* PercorreLista;
    PercorreLista = InicioLista;//copia do inicio da lista

    while(PercorreLista!= NULL) //enquanto existir nó na lista
    {
      if (PercorreLista->informacao == ElementoProcurado) //se encontrou o elemento procurado
        return PercorreLista; //retorna o nó encontrado

      else PercorreLista = PercorreLista->proximoNo; //continuar percorrendo a lista
    }

    //se saiu o while, é porque não encontrou o número! Retornar NULL;
    return NULL;
}
```

```c
struct noLista* remove (struct noLista* InicioLista, int ElementoParaRemover)
{
    struct noLista* PercorreLista = InicioLista; //cópia do início da lista

    //procurar o elementou ou até chegar no final da lista
    while (PercorreLista != NULL && PercorreLista->informacao != ElementoParaRemover)
    {
        PercorreLista = PercorreLista->proximoNo; //percorrer a lista
    }

    if (PercorreLista == NULL ) //não encontrou o número na lista para ser removido
        return InicioLista;


    if (PercorreLista->NoAnterior == NULL) //significa que o elemento procurado é  o primeiro da lista
    {
        InicioLista = PercorreLista->proximoNo; //atualizar o início da lista
    }
    else {  //"ligar" o próximo nó do anterior com o próximo do elemento que será removido
          PercorreLista->NoAnterior->proximoNo = PercorreLista->proximoNo;
          }

    if(PercorreLista->proximoNo != NULL) //se não for o último da lista
    {
        //ligar o  o campo "noAnterior" do proximo nó ao nó anterior daquele que será removido
        PercorreLista->proximoNo->NoAnterior = PercorreLista->NoAnterior;
    }
    free(PercorreLista); //liberar região de memória do nó removido

  return InicioLista;
}
```

```c
void ImprimirLista(struct noLista  *InicioDaLista)
{
    noLista *NoAtual = InicioDaLista; //copiar o endereço do primeiro nó da lista

    while (NoAtual != NULL) //percorrer a lista até encontrar o último nó = NULL
    {
        printf("%d -> ",NoAtual->informacao); //mostrar a informação do nó
        NoAtual = NoAtual->proximoNo; //apotar para o próximo nó da lista
    }

    printf("NULL");

}
```

```c
struct noLista* inserirFimLista(struct noLista* InicioLista, int NovoNumero)
{
    struct noLista *PercorreLista = InicioLista; //cópia do início da lista
    struct noLista* novoNo = (struct noLista*)malloc(sizeof(struct noLista)); // alocar memória para o novo nó

    if(PercorreLista != NULL) //se a lista não estiver vazia
    {

            while (PercorreLista->proximoNo != NULL) //percorrer a lista até encontrar o último nó
            {
              PercorreLista = PercorreLista->proximoNo;

            }

            //o que era o último nó, agora será o penúltimo: aponta para o novoNo
            PercorreLista->proximoNo = novoNo; // o que era o último nó (agora será o penúltimo!) aponta para o novo nó

   }

    novoNo->NoAnterior = PercorreLista; //apontar o campo NoAnterior para o nó PercorreLista
    novoNo->informacao = NovoNumero; //inserir a nova informação
    novoNo->proximoNo = NULL; //o novoNo será o último da lista


    if(InicioLista == NULL)//a lista estava vazia
        return novoNo;
    else return InicioLista;
}
```

```c
struct noLista * inserirInicioLista (struct noLista *InicioDaLista, int NovoNumero)
 {
    //alocar memória para o novo Nó da Lista
    struct noLista* novoNo = (struct noLista*) malloc(sizeof(struct noLista));

    //O campo NoAnterior será NULL, pois agora ele será o primeiro da lista
    novoNo->NoAnterior = NULL;
    //Inserir as informação para o novo nó
    novoNo->informacao = NovoNumero;
    //Apontar o campo "próximo" do novo nó para o local que o InicioDaLista apontava
    novoNo->proximoNo = InicioDaLista;


    if(InicioDaLista != NULL) // se a lista não estiver vazia, ligar o nó anterior do início da lista ao novo nó
       InicioDaLista->NoAnterior = novoNo;


    return novoNo;
}
```

```c
int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;

    //inserir a informaçao 88 no fim da lista
    PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 88);
    //inserir a informaçao 42 no fim da lista
   PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, 42);

    //inserir a informaçao 53 no fim da lista
    PrimeiroNoDaLista = inserirFimLista (PrimeiroNoDaLista, 53);

    //inserir a informaçao 44 no fim da lista
    PrimeiroNoDaLista = inserirFimLista (PrimeiroNoDaLista, 44);

   //remover o número 42
   PrimeiroNoDaLista = remove (PrimeiroNoDaLista, 42);


    //imprimir o conteúdo da lista
   ImprimirLista(PrimeiroNoDaLista);

   if (ProcurarElementoLista(PrimeiroNoDaLista, 53) == NULL)
    {
       printf("\n Elemento não encontrado");
   }
    else
    {
      printf("\n Elemento encontrado");
    }
}
```

```
*** INSERIR NUMEROS NO INICIO DA LISTA ****

 -- Digite um numero:2

 -- Digite um numero:4


 *** LISTA GERADA ****
4 -> 2 -> NULL

 *** INSERIR NUMEROS NO FIM DA LISTA ****

 -- Digite um numero:6

 -- Digite um numero:5


 *** LISTA GERADA ****
4 -> 2 -> 6 -> 5 -> NULL

 *** REMOVER NUMEROS DA LISTA ****

 -- Digite um numero:6


 *** LISTA GERADA ****
4 -> 2 -> 5 -> NULL

 *** PROCURAR ELEMENTO NA LISTA  ****

 -- Digite um numero:4

 Elemento encontrado.
```

```c
int main()
{
    struct noLista *PrimeiroNoDaLista = NULL;

    printf("*** INSERIR NUMEROS NO INICIO DA LISTA **** \n");
    int numero;
    for(int i = 0; i < 2; i++)
    {
        printf("\n -- Digite um numero:");
        scanf("%d", &numero);

        //inserir a informaçao 88 no fim da lista
        PrimeiroNoDaLista = inserirInicioLista (PrimeiroNoDaLista, numero);
    }

    printf("\n\n *** LISTA GERADA **** \n");
    //imprimir o conteúdo da lista
   ImprimirLista(PrimeiroNoDaLista);

    printf("\n\n *** INSERIR NUMEROS NO FIM DA LISTA **** \n");
    for(int i = 0; i < 2; i++)
    {
        printf("\n -- Digite um numero:");
        scanf("%d", &numero);

        //inserir a informaçao 88 no fim da lista
        PrimeiroNoDaLista = inserirFimLista (PrimeiroNoDaLista, numero);
    }
```

```
    printf("\n\n *** LISTA GERADA **** \n");
    //imprimir o conteúdo da lista
   ImprimirLista(PrimeiroNoDaLista);

  printf("\n\n *** REMOVER NUMEROS DA LISTA **** \n");
  printf("\n -- Digite um numero:");
  scanf("%d", &numero);
  PrimeiroNoDaLista = remove (PrimeiroNoDaLista, numero);

  printf("\n\n *** LISTA GERADA **** \n");
  //imprimir o conteúdo da lista
   ImprimirLista(PrimeiroNoDaLista);

   printf("\n\n *** PROCURAR ELEMENTO NA LISTA  **** \n");
   printf("\n -- Digite um numero:");
   scanf("%d", &numero);

   if (ProcurarElementoLista(PrimeiroNoDaLista, numero) == NULL)
    {
       printf("\n Elemento nao encontrado.");
   }
    else
    {
      printf("\n Elemento encontrado.");
    }
}
```

# Exercícios 😀

## LISTAS DUPLAMENTE ENCADEADAS

1- Implemente uma estrutura de um nó de uma lista duplamente encadeada para armazenar a idade de uma pessoa.

2- Implemente uma função para inserir um nó no início de uma lista duplamente encadeada.

3- Implemente uma função para imprimir todos os elementos de uma lista duplamente encadeada.

4- No programa principal, inicie o Primeiro nó da lista com NULL e insira 2 idades no início da lista duplamente encadeada. Em seguida, imprima a lista.
Obs.: solicite para o usuário digitar uma idade por vez.

5- Implemente uma função para inserir um nó no final da lista duplamente encadeada.

6- No programa principal, insira 2 idades no final da lista duplamente encadeada. Em seguida, imprima a lista.
Obs.: solicite para o usuário digitar uma idade por vez.

7- Implemente uma função para remover uma determinada idade da lista duplamente encadeada.

8- No programa principal, remova uma determinada idade da lista duplamente encadeada. Em seguida, imprima a lista após a remoção.
Obs.: solicite para o usuário digitar a idade que será removida.

9- Implemente uma função para procurar uma determinada idade em uma lista duplamente encadeada. A função deve retornar o nó contendo a idade encontrada ou NULL caso a idade procurada não esteja na lista.

10- No programa principal, solicite para o usuário digitar uma idade para ser procurada na lista duplamente encadeada. Exiba uma mensagem dizendo se o número está ou não na lista. Em seguida, imprima todo o conteúdo da lista.

11- Implemente uma função para procurar uma determinada idade em uma lista duplamente encadeada e retornar a POSIÇÃO em que esta idade está na lista. Caso a idade não esteja na lista, a função deve retornar -1.

12 – Faça uma função que percorra a lista duplamente encadeada e imprima somente as idades acima de 18 anos.

# Muito Obrigada!

**Profª. Angela Abreu Rosa de Sá, Drª.**

*Contato: angelaabreu@gmail.com*