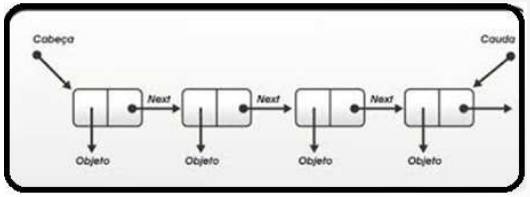
ESTRUTURA E RECUPERAÇÃO DE DADOS A

ENGENHARIA DA COMPUTAÇÃO

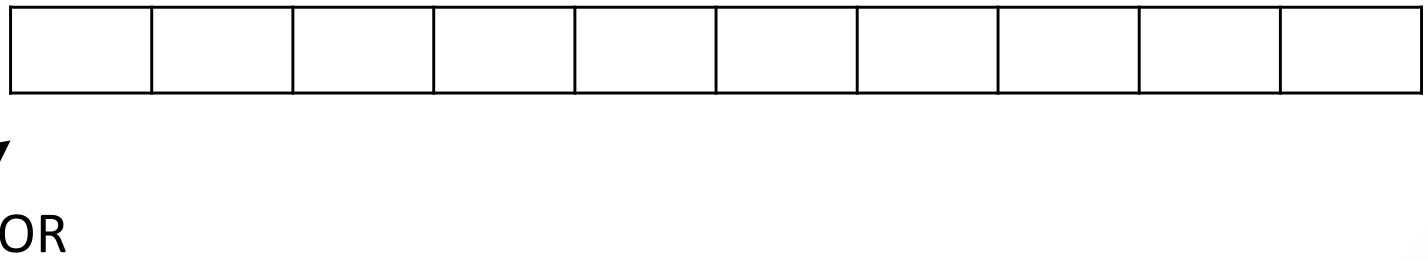
AULA 08– Listas Lineares - Continuação

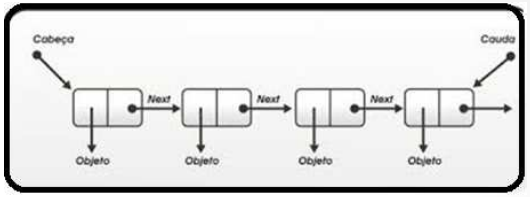
Profa. Dra. Lúcia F. A. Guimarães

- **Listas Encadeadas**

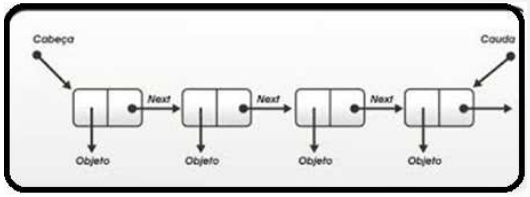


  - Vetor:
    - Ocupa espaço contíguo de memória
    - Permite acesso randômico aos elementos
    - Deve ser dimensionado com um número máximo de elementos

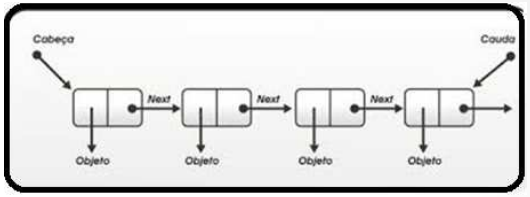




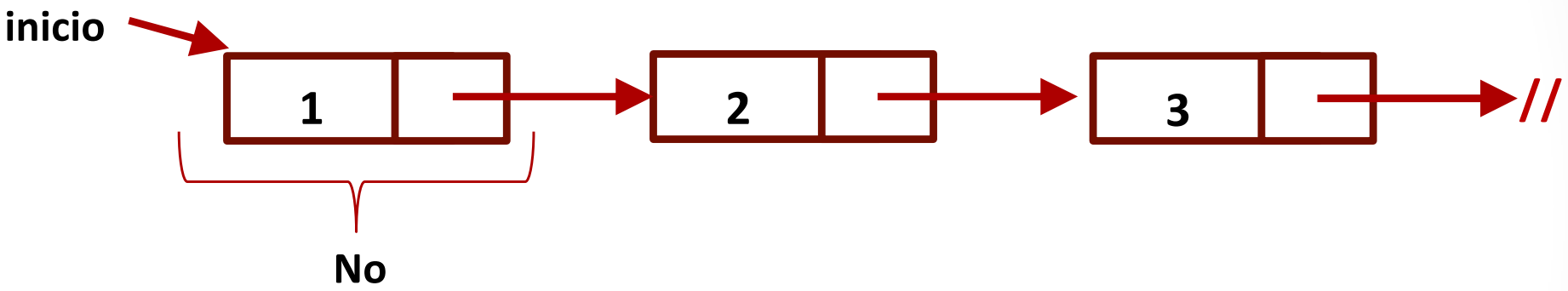
- **Listas Encadeadas**
  - Estrutura onde **para cada novo elemento inserido, aloca-se um espaço de memória para armazená-lo**.
  - O espaço total de memória gasto pela estrutura é proporcional ao número de elementos nela armazenado.
  - Não é possível garantir que os elementos armazenados na lista ocuparão um espaço de memória contíguo.



- **Listas Encadeadas**
  - Portanto **não há acesso direto aos elementos da lista.**
  - Para que seja possível percorrer todos os elementos da lista, é necessário explicitamente guardar o encadeamento dos elementos,
  - O que é feito armazenando-se, junto com a informação de cada elemento, um ponteiro para o próximo elemento da lista



- **Listas Encadeadas**
  - Exemplo:

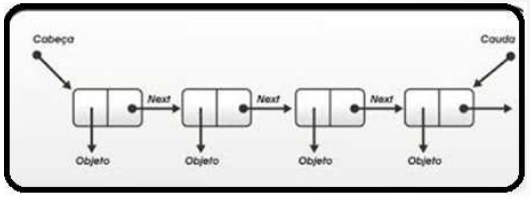


A Lista tem como característica principal a inserção e remoção em qualquer posição.

- **Para trabalharmos com lista considere a seguinte estrutura:**

```
struct no
{
        int info;
        struct no * prox;
};
typedef struct no  Lista;
```

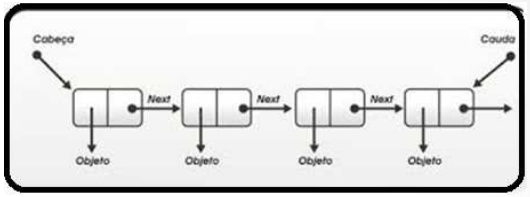
Caso você deseje, você denominar esta estrutura de No



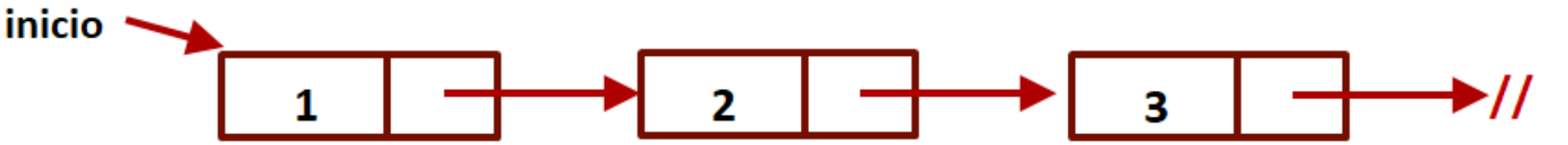
- **Função Incialização**

/* função de inicialização: retorna uma lista vazia */

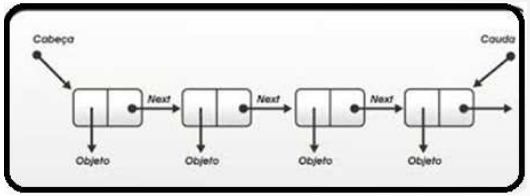
```
Lista* inicializa ( )
{
        return NULL;
}
```



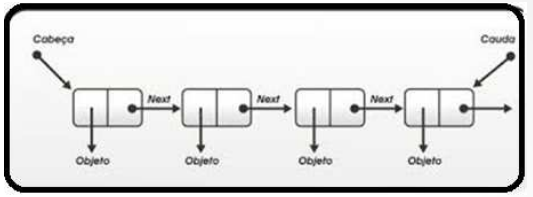
- **Inserção:**
  - A função de inserção mais simples insere o novo elemento no início da lista.
  - Note que **o ponteiro que representa a lista deve ter seu valor atualizado**



- **Inserção (insere no início)**

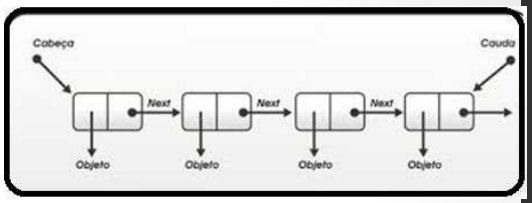



```
/* inserção no início: retorna a lista atualizada */
    Lista* insere (Lista* recebida, int valor)
    {
        Lista  *novo ;
        novo= (Lista*) malloc(sizeof(Lista));
        novo->info = valor;
        novo->prox = recebida;
        return novo;
    }
```



- **Percorrimento**
  - Uma possível implementação dessa função seria:

```
void imprime (Lista* recebida)
{
        Lista* p; /* variável auxiliar para percorrer a lista */
        for (p = recebida; p != NULL; p = p->prox)
                printf("info = %d\n", p->info);
}
```



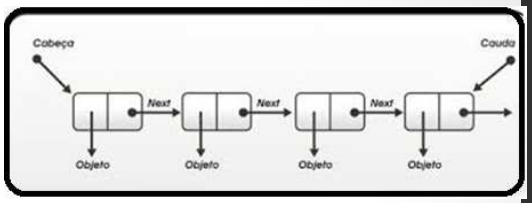
- **Função Verifica se a Lista Está Vazia**
  - Pode ser útil implementarmos uma função que verifique se uma lista está vazia ou não.
  - A função recebe a lista e
    - retorna 1 se estiver vazia ou
    - retorna 0 se não estiver vazia.

- **Função Verifica se a Lista Está Vazia**

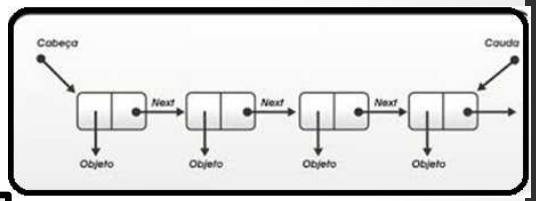
```
/* função vazia: retorna 1 se vazia ou 0 se não vazia */
int vazia (Lista* recebida)
{
        if (recebida == NULL)
        {
                return 1;
        }
        else
        {
                return 0;
        }
}
```

- **Função Verifica se a Lista Está Vazia (*simplificada*)**

```
/* função vazia: retorna 1 se vazia ou 0 se não vazia */
int vazia (Lista* recebida)
{
        if (recebida == NULL)
        {
                return 1;
        }
        return 0;
}
```

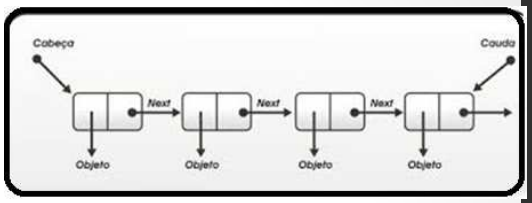


- **Função Busca**
  - Verifica se um determinado elemento está presente na lista.
  - A função recebe a informação referente ao elemento que queremos buscar e fornece como valor de retorno
    - O ponteiro do nó da lista que representa o elemento.
    - Caso o elemento não seja encontrado na lista, o valor retornado é NULL



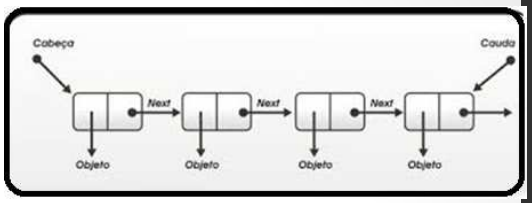
- **Função Busca**

```
/* função busca: busca um elemento na lista */
Lista* busca (Lista* recebida, int v)
{
        Lista* p;
        for (p= recebida; p!=NULL; p=p->prox)
        {
                if (p->info == v)
                                return p;
        }
        return NULL; /* não achou o elemento */
}
```



- **Função Libera a Lista**
  - Para apagarmos a Lista todos os elementos da mesma tem que ser liberados usando o comando free
  - E por último, atribuímos NULL para o ponteiro que representa o início da lista

- **Função Libera a Lista**




```
Lista* libera(Lista *receb)
{
   Lista *aux;
   while(receb!= NULL)
   {
      aux=receb;
      receb=receb->prox;
      free(aux);
   }
   return NULL;
}
```

- **O programa principal ficaria:**

```
int main()
{
   Lista *inicio;
   inicio = inicializa();
   inicio = insere(inicio,9);
   inicio = insere(inicio,6);
   inicio = insere(inicio,8);
   inicio = insere(inicio,5);
   inicio = insere(inicio,7);
   printf("\n\n");
   imprime(inicio);
   inicio = libera(inicio);
   if(vazia(inicio))
   {
      printf("\n\n\tLISTA VAZIA\n\n\t");
   }

}
```

Listas Encadeadas

```
int main()
{
   Lista *inicio;
   inicio = inicializa();
   inicio = insere(inicio,9);
   inicio = insere(inicio,6);
   inicio = insere(inicio,8);
   inicio = insere(inicio,5);
   inicio = insere(inicio,7);
   printf("\n\n");
   imprime(inicio);
   inicio = libera(inicio);
   if(vazia(inicio))
   {
      printf("\n\n\tLISTA VAZIA\n\n\t");
   }

}
```

```
Lista* inicializa ( )

{

      return NULL;

}
```

Listas Encadeadas

```
int main()
{
   Lista *inicio;
   inicio = inicializa();
   inicio = insere(inicio,9);
   inicio = insere(inicio,6);
   inicio = insere(inicio,8);
   inicio = insere(inicio,5);
   inicio = insere(inicio,7);
   printf("\n\n");
   imprime(inicio);
   inicio = libera(inicio);
   if(vazia(inicio))
   {
      printf("\n\n\tLISTA VAZIA\r
   }

}
```

```
/* inserção no início: retorna a lista atualizada */
   Lista* insere (Lista* recebida, int valor)
   {
        Lista  *novo ;
        novo= (Lista*) malloc(sizeof(Lista));
        novo->info = valor;
        novo->prox = recebida;
        return novo;
   }
```

Listas Encadeadas

```
int main()
{
   Lista *inicio;
   inicio = inicializa();
   inicio = insere(inicio,9);
   inicio = insere(inicio,6);
   inicio = insere(inicio,8);
   inicio = insere(inicio,5);
   inicio = insere(inicio,7);
   printf("\n\n");
   imprime(inicio);
   libera(inicio);
   if(vazia(inicio))
   {
      printf("\n\n\tLISTA VAZI
   }

}
```

```
void imprime (Lista* recebida)
{
      Lista* p; /* variável auxiliar para percorrer a lista */
      for (p = recebida; p != NULL; p = p->prox)
            printf("info = %d\n", p->info);
}
```

Listas Encadeadas

```
int main()
{
   Lista *inicio;
   inicio = inicializa();
   inicio = insere(inicio,9);
   inicio = insere(inicio,6);
   inicio = insere(inicio,8);
   inicio = insere(inicio,5);
   inicio = insere(inicio,7);
   printf("\n\n");
   imprime(inicio);
   inicio = libera(inicio);
   if(vazia(inicio))
   {
      printf("\n\n\tLISTA VAZIA\n\n\t");
   }

}
```

```
Lista* libera(Lista *receb)
{
   Lista *aux;
   while(receb!= NULL)
   {
      aux=receb;
      receb=receb->prox;
      free(aux);
   }
   return NULL;
}
```

Listas Encadeadas

```
int main()
{
    Lista *inicio;
    inicio = inicializa();
    inicio = insere(inicio,9);
    inicio = insere(inicio,6);
    inicio = insere(inicio,8);
    inicio = insere(inicio,5);
    inicio = insere(inicio,7);
    printf("\n\n");
    imprime(inicio);
    inicio = libera(inicio);
    if(vazia(inicio))
    {
        printf("\n\n\tLISTA VAZIA\n\n\t");
    }

}
```

```
/* função vazia: retorna 1 se vazia ou 0 se não vazia */
int vazia (Lista* recebida)
{
        if (recebida == NULL)
        {
                return 1;
        }
        return 0;
}
```

- **Exemplo de Busca**

```
int main()
{
   Lista *inicio, * local;
   inicio = inicializa();
   inicio = insere(inicio,9);
   inicio = insere(inicio,6);
   inicio = insere(inicio,8);
   local = busca(insere,6);
   if(local !=NULL)
   {
         printf("%d",local->info);
   }
   libera(inicio);

}
```

```
/* função busca: busca um elemento na lista */
Lista* busca (Lista* recebida, int v)
{
      Lista* p;
      for (p= recebida; p!=NULL; p=p->prox)
      {
            if (p->info == v)
                  return p;
      }
      return NULL; /* não achou o elemento */
}
```

- **Faça Você**

1. Crie uma função para apagar o primeiro elemento da lista
2. Crie uma função para inserir um elemento no final da lista
3. Implemente todas as funções vistas até agora