ESTRUTURA DE DADOS

TAD LISTA

PROFESSORA SANDRA ROVENA FRIGERI

SUMÁRIO

- CARACTERÍSTICAS DO TAD LISTA
- TIPOS DE LISTAS
- OPERAÇÕES BÁSICAS DO TAD LISTA
- ESTRUTURA DE DADOS TAD LISTA
- IMPLEMENTAÇÃO COM ENCADEAMENTO

CARACTERÍTICAS: LISTA

LISTA: CONJUNTO DE ELEMENTOS

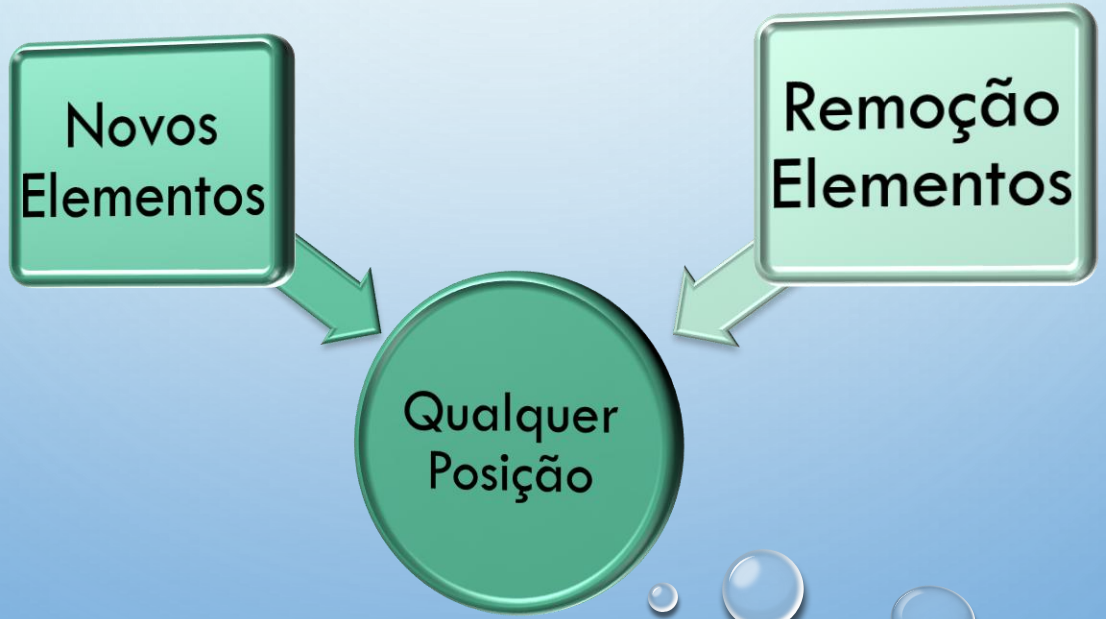
- LISTA DE COMPRAS
- LISTA DE CONVIDADOS
- LISTA DE TIPOS
- LISTA DE VERIFICAÇÃO
- LISTA DE REQUISITOS
- LISTA DE ALUNOS

CARACTERÍSTICAS: LISTA

- CONJUNTO DE ELEMENTOS
- AGRUPA ELEMENTOS QUE POSSUI ALGUMA RELAÇÃO ENTRE ELES
- CRITÉRIO DE ORDENAÇÃO?
  - PODE SER A ORDEM DE INCLUSÃO
  - PODE UTILIZAR UMA CHAVE DE ORDENAÇÃO
- GENERALIZAÇÃO:
  - PILHA
  - FILA

CARACTERÍSTICAS: LISTA

- ONDE SÃO INCLUÍDOS NOVOS ELEMENTOS?
- DE ONDE PODEM SER REMOVIDOS ELEMENTOS?



CARACTERÍSTICAS: TAD LISTA

- ESTRUTURA LINEAR
- ELEMENTOS DO MESMO TIPO
- ELEMENTOS **ORDENADOS**
  - ORDEM DE INCLUSÃO
  - CHAVE DE ORDENAÇÃO
- NOVOS ELEMENTOS: INÍCIO, MEIO, FIM
- REMOVER ELEMENTOS: INÍCIO, MEIO, FIM

TIPOS DE LISTAS

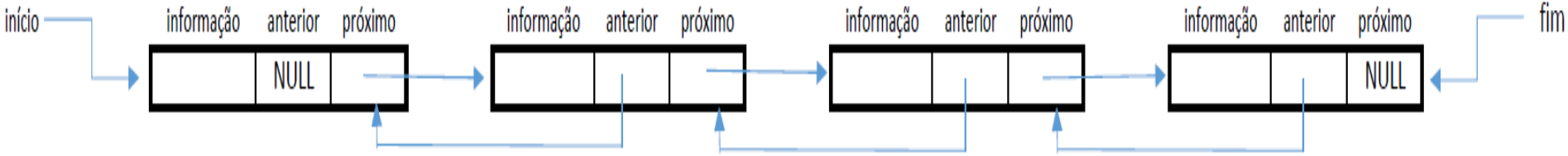
- LISTA SIMPLESMENTE ENCADEADA
- LISTA DUPLAMENTE ENCADEADA
- LISTA CIRCULAR

TIPOS DE LISTAS

LISTA SIMPLESMENTE ENCADEADA

TIPOS DE LISTAS

LISTA DUPLAMENTE ENCADEADA

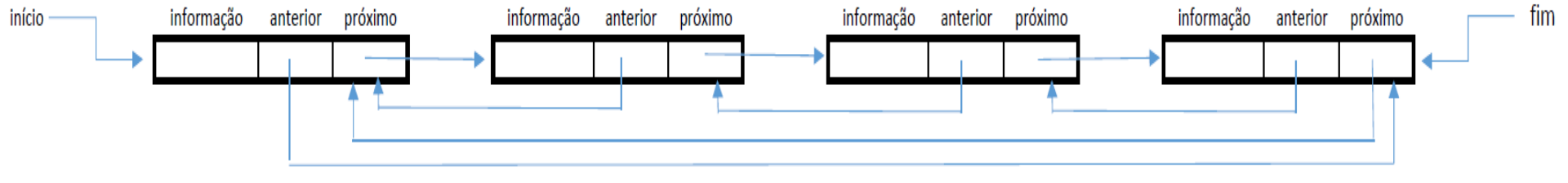


TIPOS DE LISTAS

LISTA CIRCULAR SIMPLESMENTE ENCADEADA

TIPOS DE LISTAS

LISTA CIRCULAR DUPLAMENTE ENCADEADA



OPERAÇÕES TAD

- **OPERAÇÕES PRIMITIVAS:**
  - DEPENDEM DA ESTRUTURA DO TAD
  - OPERAÇÕES BÁSICAS DO TAD

- **OPERAÇÕES NÃO PRIMITIVAS DE UM TAD:**
  - NÃO DEPENDEM DA ESTRUTURA DO TAD
  - CHAMAM AS OPERAÇÕES PRIMITIVAS

OPERAÇÕES PRIMITIVAS TAD LISTA

LISTA* **CRIA** ();

- CRIAR A ESTRUTURA DE DADOS DA LISTA
- INICIALIZAR:
  - TAMANHO DA LISTA
  - REFERÊNCIAS PARA O INÍCIO E FIM DA LISTA

OPERAÇÕES PRIMITIVAS TAD LISTA

INT **VAZIA** (LISTA * L);

- TESTA SE A LISTA ESTÁ VAZIA
  - VERIFICA O TAMANHO DA LISTA OU REFERÊNCIA DO INÍCIO DA LISTA
  - RETORNA 1: LISTA VAZIA
  - RETORNA 0: LISTA NÃO ESTÁ VAZIA

OPERAÇÕES PRIMITIVAS TAD LISTA

VOID **LIBERA** (LISTA * L);

- LIBERA O ESPAÇO DE MEMÓRIA OCUPADO PELA LISTA

OPERAÇÕES PRIMITIVAS TAD LISTA

INT **ELEMENTOS(LISTA** * L);

- RETORNA O TAMANHO DA LISTA
  - CONSULTA O TAMANHO DA LISTA E RETORNA ESSA INFORMAÇÃO

OPERAÇÕES PRIMITIVAS TAD LISTA

VOID **INSERE** (LISTA * L, DADO V);

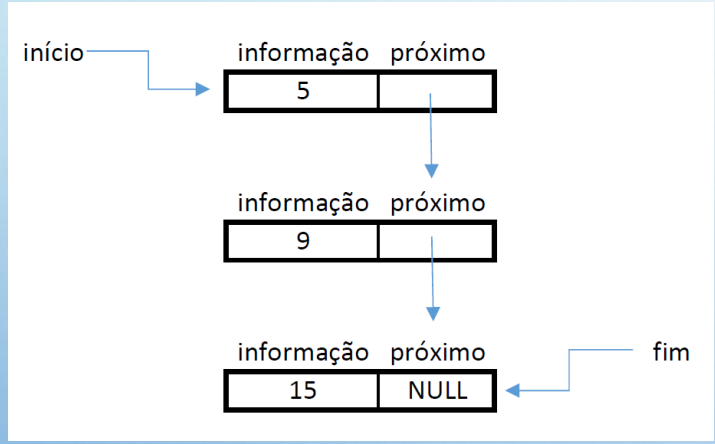
- INCLUI UM NOVO ELEMENTO NA LISTA
  - CONSIDERA O CRITÉRIO DE ORDENAÇÃO E INCLUI NOVO ELEMENTO
  - A INCLUSÃO PRECISA DETERMINAR ONDE O NOVO ELEMENTO PODE SER INCLUÍDO: INÍCIO, MEIO OU FIM DA LISTA

INCLUSÃO DE NOVO ELEMENTO

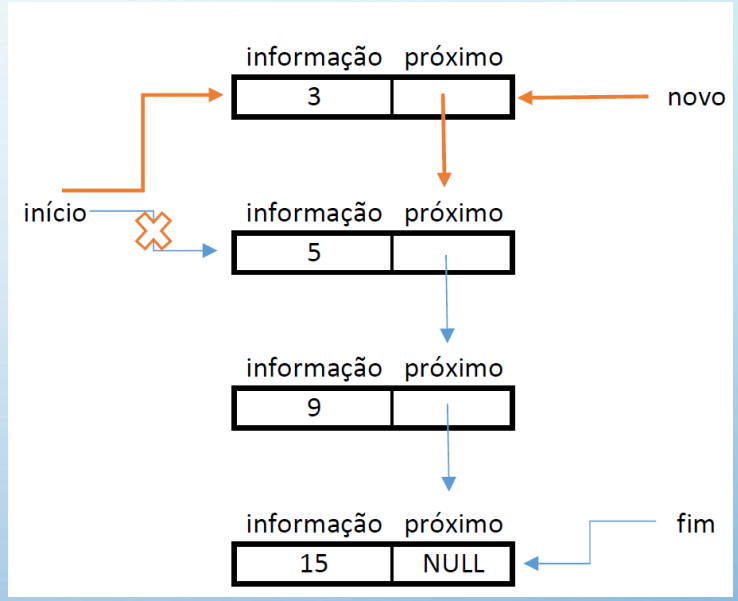
CRITÉRIO DE ORDENAÇÃO:

- **ORDEM DE INCLUSÃO**: LISTA SEMELHANTE A FILA, INCLUI NO FINAL DA LISTA
- **CHAVE DE ORDENAÇÃO**: CAMPO QUE DEVE SER MANTIDO EM ORDEM:
  - NECESSÁRIO VARRER A LISTA E LOCALIZAR POSIÇÃO ONDE O NOVO ELEMENTO SERÁ INSERIDO

INCLUSÃO DE NOVO ELEMENTO

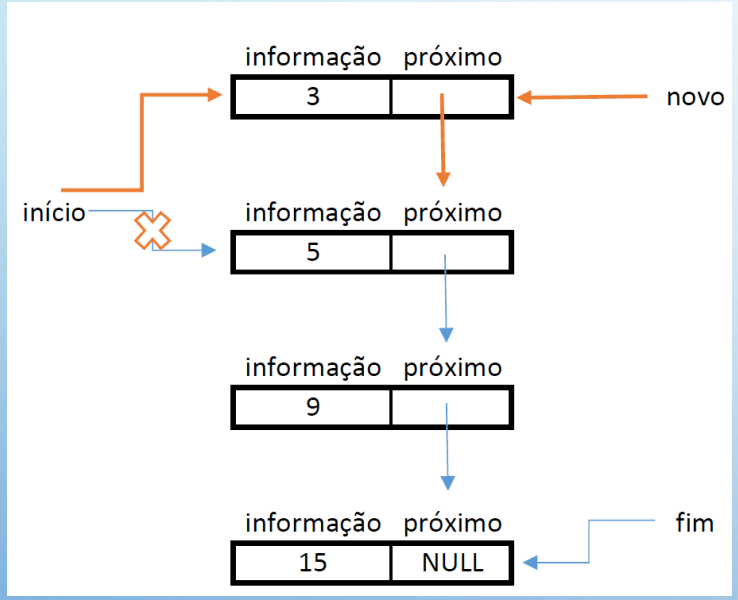


Situação Inicial

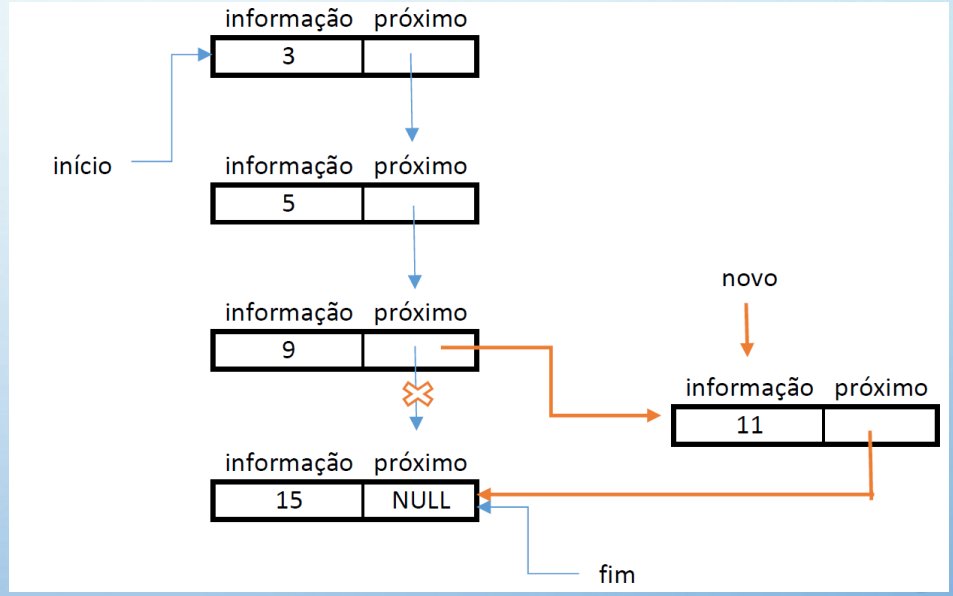


- **Inclusão novo elemento: 3**
- **Início da Lista**

INCLUSÃO DE NOVO ELEMENTO



Situação Inicial

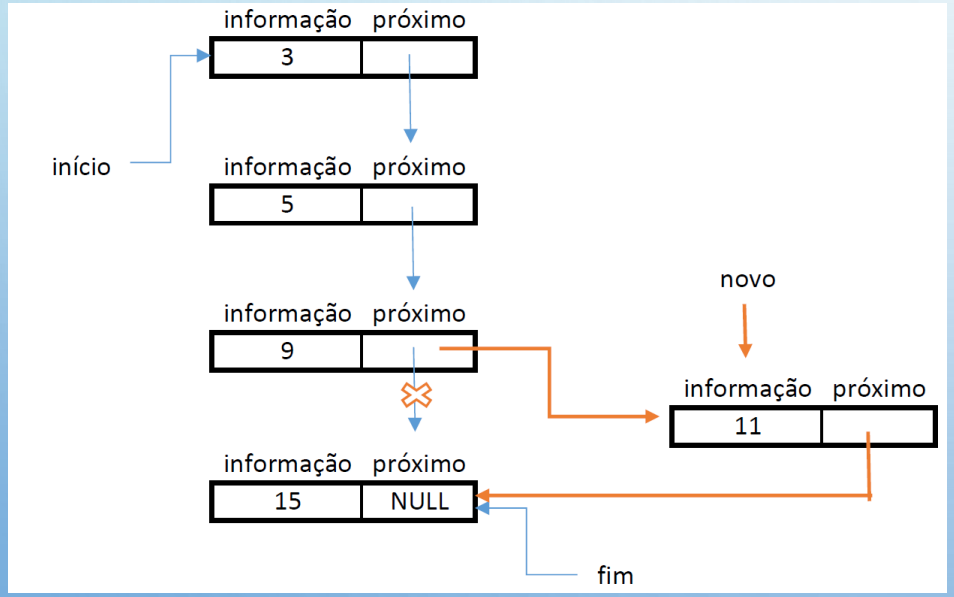


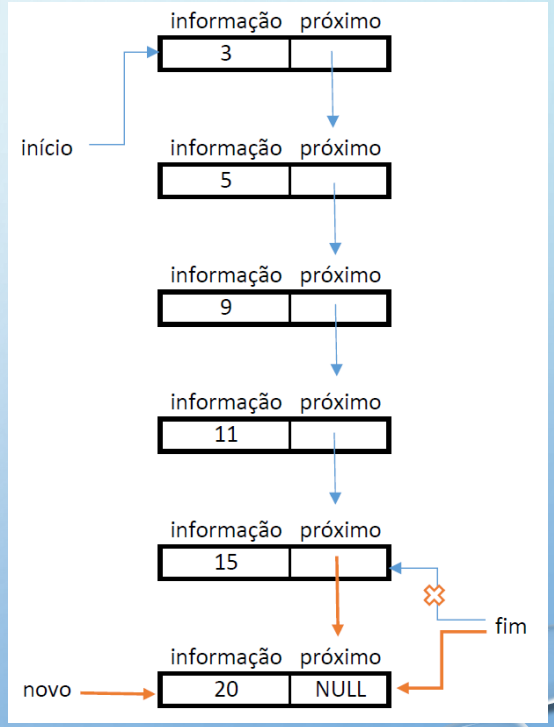
- **Inclusão novo elemento: 11**
- **Meio da Lista**

INCLUSÃO DE NOVO ELEMENTO

Situação Inicial

- **Inclusão novo elemento: 20**
- **Fim da Lista**





OPERAÇÕES PRIMITIVAS TAD LISTA

INT **REMOVE** (LISTA * L, DADO* V);

EXCLUI UM ELEMENTO DA LISTA

- LOCALIZA O ELEMENTO A SER EXCLUÍDO: VARRE A LISTA PARA PROCURÁ-LO
- A EXCLUSÃO PODE OCORRER:
  - **INÍCIO**: DEVE SER ATUALIZADA A REFERÊNCIA DO PRIMEIRO ELEMENTO DA LISTA
  - **MEIO**: DEVEM SER ATUALIZADOS OS PONTEIROS PRÓXIMO (ANTERIOR) DOS NODOS
  - **FIM**: DEVE SER ATUALIZADA A REFERÊNCIA DO ÚLTIMO ELEMENTO DA LISTA

TAD LISTA

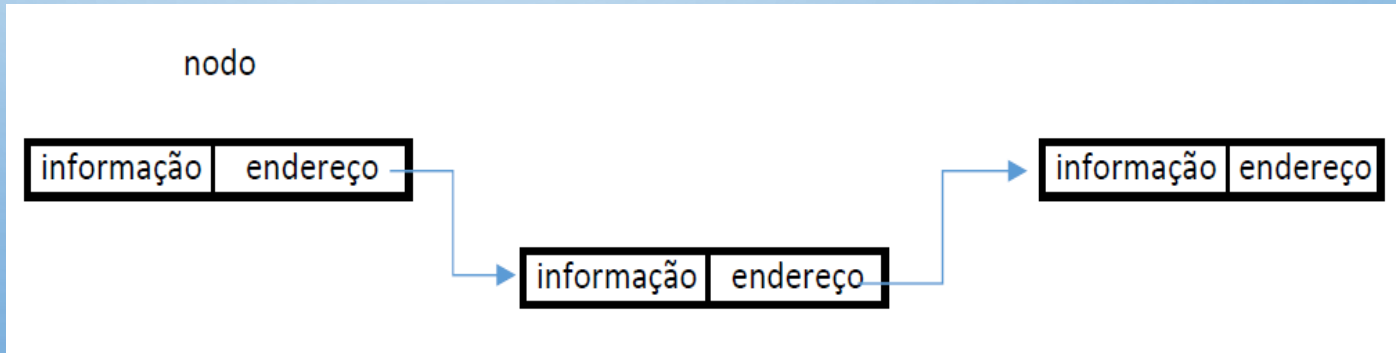
IMPLEMENTAÇÃO COM ENCADEAMENTO

TAD LISTA - ENCADEAMENTO

- ENCADEAMENTO
  - ALOCAÇÃO DINÂMICA DE MEMÓRIA
  - LIGAÇÃO ENTRE OS ELEMENTOS

ENCADEAMENTO

```
TYPEDEF STRUCT NODO{
   FLOAT  INFORMACAO;
   NODO* ENDERECO;
};
```



TAD LISTA - ENCADEAMENTO

```
TYPEDEF STRUCT NODO{
   FLOAT  INFORMAÇÃO;
   NODO* PRÓXIMO;
};
```

LISTA **SIMPLESMENTE** ENCADEADA

Último Nodo aponta para NULL

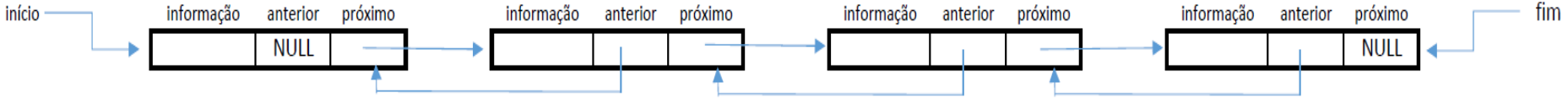
TAD LISTA - ENCADEAMENTO

```
TYPEDEF STRUCT NODO{
   FLOAT  INFORMAÇÃO;
   NODO* PRÓXIMO;
   NODO* ANTERIOR;
};
```

LISTA **DUPLAMENTE** ENCADEADA

Próximo do Último Nodo aponta para NULL

Anterior do Primeiro Nodo aponta para NULL



TAD LISTA - ENCADEAMENTO

```
TYPEDEF STRUCT NODO{
   FLOAT  INFORMAÇÃO;
   NODO* PRÓXIMO;
};
```

LISTA **CIRCULAR** SIMPLESMENTE ENCADEADA

Último Nodo aponta para o primeiro

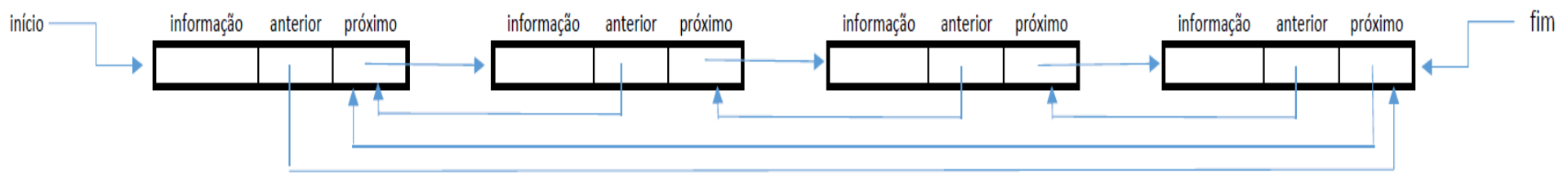
TAD LISTA - ENCADEAMENTO

```
TYPEDEF STRUCT NODO{
   FLOAT  INFORMAÇÃO;
   NODO* PRÓXIMO;
   NODO* ANTERIOR;
};
```

Próximo do Último Nodo aponta para o primeiro

Anterior do Primeiro Nodo aponta para último

LISTA **CIRCULAR** DUPLAMENTE ENCADEADA



ESTRUTURA DE DADOS

```
struct lista {
 int tamanho;
 struct nodo* inicio, fim;
};
struct lista* LISTA;
```

```
TYPEDEF STRUCT NODO
{
   FLOAT  INFORMACAO;
   NODO* PROXIMO;
};
```

OPERAÇÕES PRIMITIVAS

```
LISTA* CRIA (){
    LISTA* L = (LISTA) MALLOC (SIZEOF (LISTA));
    L->TAMANHO = 0;
    L->INICIO = NULL;
    L->FIM = NULL;
    RETURN(L);
}
```

```
struct lista {
 int tamanho;
 struct nodo* inicio, fim;
};
struct lista* LISTA;
```

OPERAÇÕES PRIMITIVAS

```
INT VAZIA (LISTA* L){
    IF (L->INICIO == NULL)
      RETURN(1);
    ELSE
      RETURN(0);
}
```

```
int tamanho (LISTA* l){
    if (l != NULL)
      return(l->tamanho);
    else
      return(0);
}
```

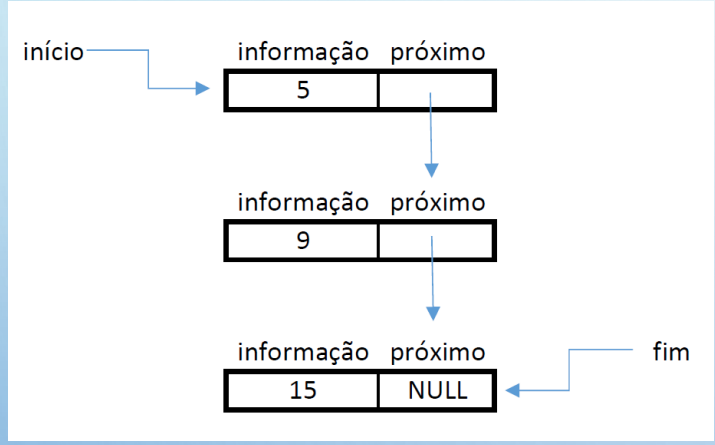
```
VOID LIBERA (LISTA* L){
    IF (L != NULL)
      FREE(L);
}
```

OPERAÇÕES PRIMITIVAS

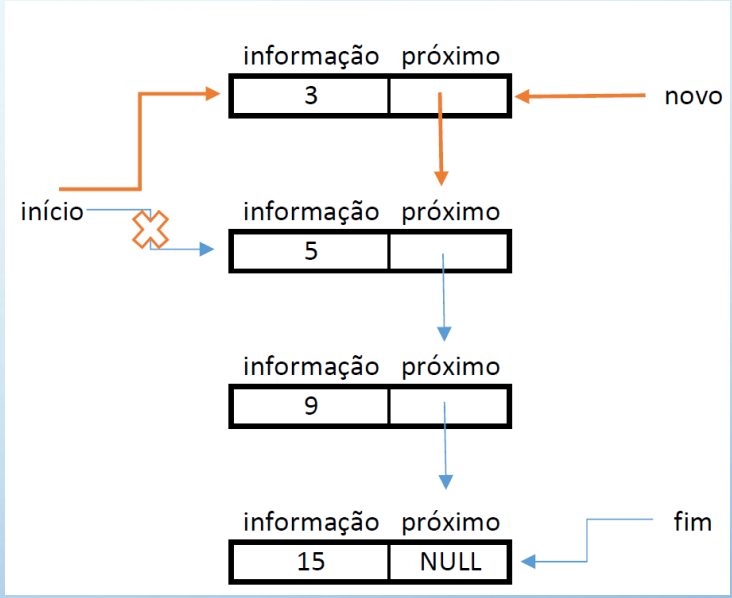
```
VOID INSERE (LISTA * L, DADO V){
 NODO* NOVO;
 NODO* ANTERIOR, CORRENTE;
 NOVO=(NODO)MALLOC(SIZEOF(NODO));
 NODO->CODIGO = V.CODIGO;
 STRCPY(NODO->NOME,V.NOME);
 NODO->IDADE = V.IDADE;
 NODO->PRÓXIMO=NULL;
 IF (VAZIA(L)){
   L->INICIO = NOVO;
   L->FIM = NOVO; }
 ELSE { CORRENTE = L->INICIO;
   IF (NOVO->CODIGO < CORRENTE->CODIGO){
    NOVO->PROXIMO = L->INICIO;
     L->INICIO = NOVO;    }
   ELSE {
WHILE((CORRENTE !=NULL)&&(NOVO->CÓDIGO>=CORRENTE->CODIGO)){
     ANTERIOR = CORRENTE;
     CORRENTE = ANTERIOR->PROXIMO;    }
ANTERIOR->PROXIMO = NOVO;
NOVO->PROXIMO = CORRENTE;
IF (CORRENTE == NULL)
         L->FIM = NOVO;    }  }
  L->TAMANHO = L->TAMANHO + 1;
}
```

INCLUSÃO DE NOVO ELEMENTO

Situação Inicial



- Inclusão novo elemento: 3
- Início da Lista, pois é menor que o primeiro elemento atual



- Novo aponta para quem era o início da lista
- Atualiza o início da lista, que irá apontar para o novo

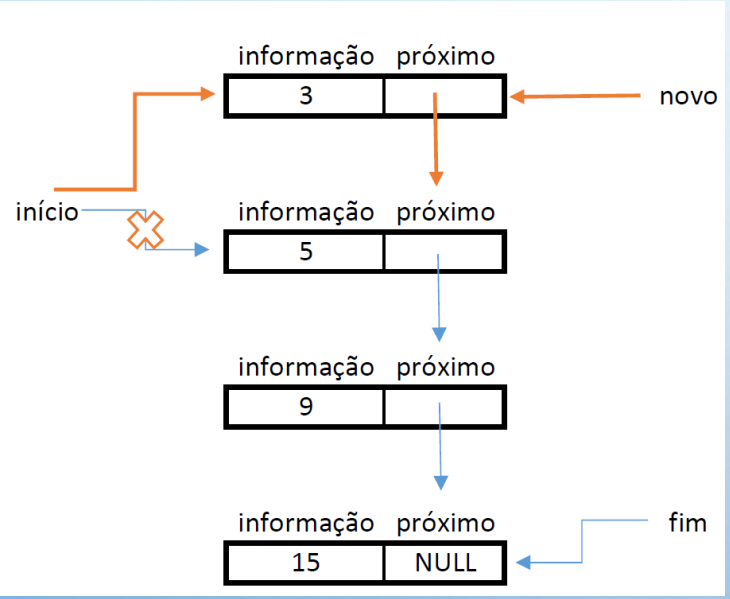
OPERAÇÕES PRIMITIVAS

```
VOID INSERE (LISTA * L, DADO V){
 NODO* NOVO;
 NODO* ANTERIOR, CORRENTE;
 NOVO=(NODO)MALLOC(SIZEOF(NODO));
 NODO->CODIGO = V.CODIGO;
 STRCPY(NODO->NOME,V.NOME);
 NODO->IDADE = V.IDADE;
 NODO->PRÓXIMO=NULL;
 IF (VAZIA(L)){
   L->INICIO = NOVO;
   L->FIM = NOVO; }
```

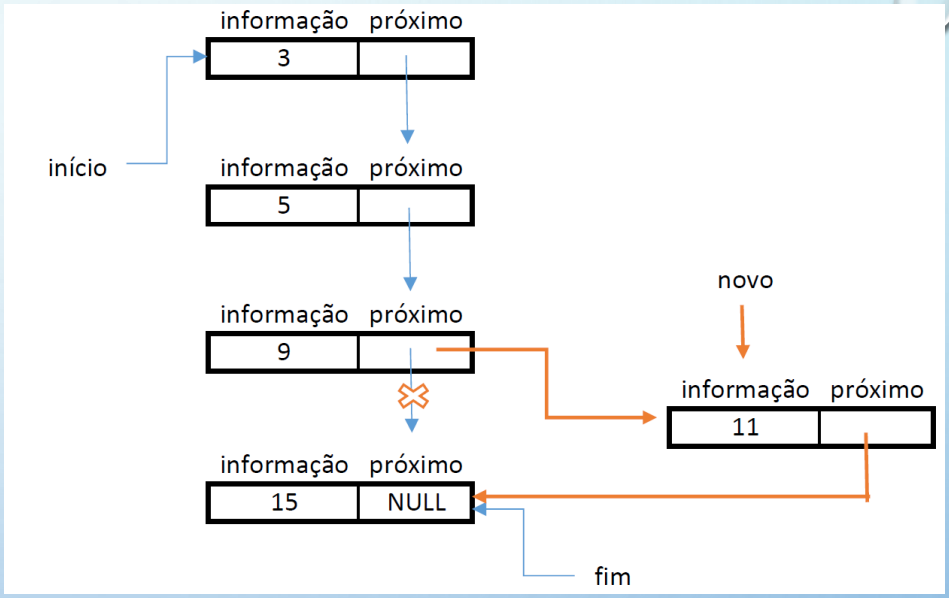
```
 ELSE { CORRENTE = L->INICIO;
   IF (NOVO->CODIGO < CORRENTE->CODIGO){
    NOVO->PROXIMO = L->INICIO;
     L->INICIO = NOVO;    }
   ELSE {
WHILE((CORRENTE !=NULL)&&(NOVO->CÓDIGO>=CORRENTE->CODIGO)){
     ANTERIOR = CORRENTE;
     CORRENTE = ANTERIOR->PROXIMO;    }
ANTERIOR->PROXIMO = NOVO;
NOVO->PROXIMO = CORRENTE;
IF (CORRENTE == NULL)
        L->FIM = NOVO;    }  }
  L->TAMANHO = L->TAMANHO + 1;
}
```

INCLUSÃO DE NOVO ELEMENTO

Situação Atual



- Inclusão novo elemento: 11
- Meio da Lista, entre os elementos 9 e 15



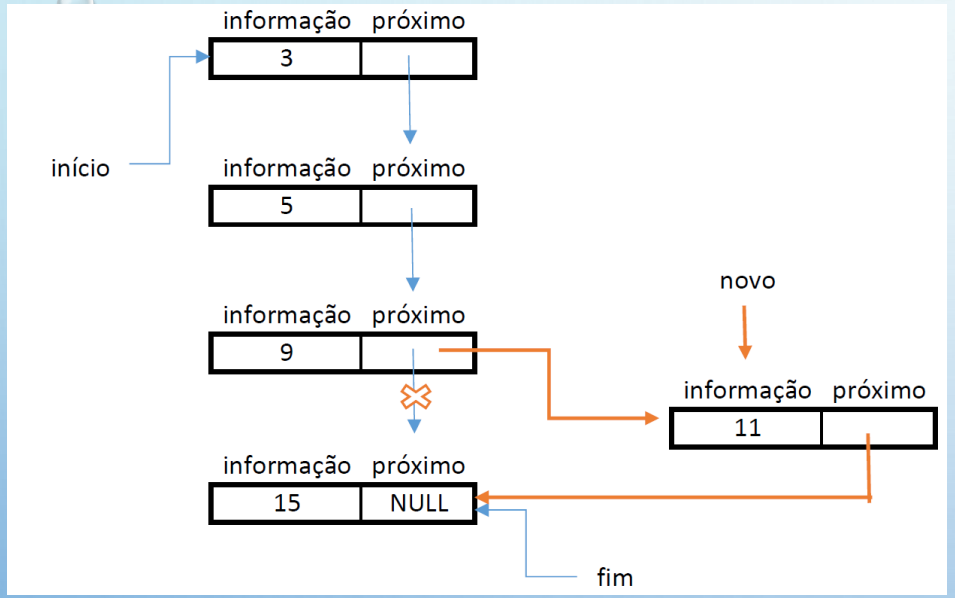
- O novo elemento (11) irá apontar para quem o anterior (9) aponta
- O anterior (9) irá apontar para o novo elemento

OPERAÇÕES PRIMITIVAS

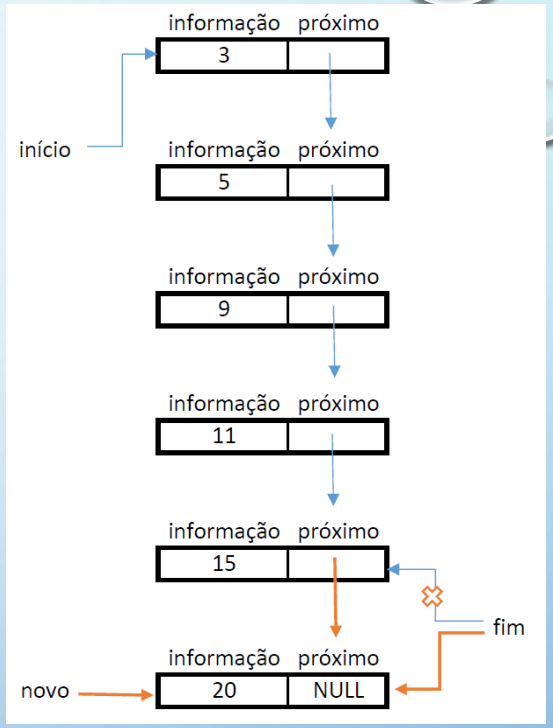
```
VOID INSERE (LISTA * L, DADO V){
 NODO* NOVO;
 NODO* ANTERIOR, CORRENTE;
 NOVO=(NODO)MALLOC(SIZEOF(NODO));
 NODO->CODIGO = V.CODIGO;
 STRCPY(NODO->NOME,V.NOME);
 NODO->IDADE = V.IDADE;
 NODO->PRÓXIMO=NULL;
 IF (VAZIA(L)){
   L->INICIO = NOVO;
   L->FIM = NOVO; }
 ELSE { CORRENTE = L->INICIO;
   IF (NOVO->CODIGO < CORRENTE->CODIGO){
    NOVO->PROXIMO = L->INICIO;
     L->INICIO = NOVO;    }
   ELSE {
WHILE((CORRENTE !=NULL)&&(NOVO->CÓDIGO>=CORRENTE->CODIGO)){
     ANTERIOR = CORRENTE;
     CORRENTE = ANTERIOR->PROXIMO;    }
ANTERIOR->PROXIMO = NOVO;
NOVO->PROXIMO = CORRENTE;
IF (CORRENTE == NULL)
         L->FIM = NOVO;    }  }
  L->TAMANHO = L->TAMANHO + 1;
}
```

INCLUSÃO DE NOVO ELEMENTO

Situação Atual



- Inclusão novo elemento: 20
- Fim da Lista, pois é maior que o último elemento



- O último elemento passa a apontar para o novo
- O novo aponta para NULL
- Fim da Lista aponta para o novo

OPERAÇÕES PRIMITIVAS

```
VOID INSERE (LISTA * L, DADO V){
 NODO* NOVO;
 NODO* ANTERIOR, CORRENTE;
 NOVO=(NODO)MALLOC(SIZEOF(NODO));
 NODO->CODIGO = V.CODIGO;
 STRCPY(NODO->NOME,V.NOME);
 NODO->IDADE = V.IDADE;
 NODO->PRÓXIMO=NULL;
 IF (VAZIA(L)){
   L->INICIO = NOVO;
   L->FIM = NOVO; }
 ELSE { CORRENTE = L->INICIO;
   IF (NOVO->CODIGO < CORRENTE->CODIGO){
    NOVO->PROXIMO = L->INICIO;
     L->INICIO = NOVO;    }
   ELSE {
WHILE((CORRENTE !=NULL)&&(NOVO->CÓDIGO>=CORRENTE->CODIGO)){
     ANTERIOR = CORRENTE;
     CORRENTE = ANTERIOR->PROXIMO;    }
ANTERIOR->PROXIMO = NOVO;
NOVO->PROXIMO = CORRENTE;
IF (CORRENTE == NULL)
        L->FIM = NOVO;    }  }
  L->TAMANHO = L->TAMANHO + 1;
}
```

REMOVER ELEMENTO DA LISTA

```
INT REMOVE (LISTA * L, DADO* V){
 INT CÓDIGO = *V.CODIGO;
 NODO* ANTERIOR, CORRENTE;
 IF (VAZIA(L))  {
     RETURN(0);
 }
 ELSE
 {
  CORRENTE = L->INICIO;
```

```
IF (CODIGO == CORRENTE->CODIGO){
    STRCPY(*V.NOME,CORRENTE->NOME);
     *V.IDADE=CORRENTE.IDADE;
     L->INICIO = CORRENTE->PROXIMO;
     FREE(CORRENTE);
     L->TAMANHO = L->TAMANHO – 1;
      IF (L->INICIO == NULL)
          L->FIM = NULL;
     RETURN(1);
}
```

REMOVER ELEMENTO DA LISTA

```
ELSE{
  WHILE((CORRENTE != NULL) &&
        (CODIGO != CORRENTE->CODIGO)&&
        (CODIGO > CORRENTE->CODIGO)) {
    ANTERIOR = CORRENTE;
    CORRENTE = ANTERIOR->PRÓXIMO;
  }
  IF (CORRENTE == NULL){
     RETURN(0);
   }
```

```
ELSE {
      STRCPY(*V.NOME,CORRENTE->NOME);
      *V.IDADE=CORRENTE.IDADE;
      ANTERIOR->PRÓXIMO = CORRENTE->PRÓXIMO;
      IF (L->FIM == CORRENTE) {
         L->FIM = ANTERIOR;   }
      L->TAMANHO = L->TAMANHO – 1;
      FREE(CORRENTE);
      RETURN(1);
     }  }  } }
```

TAD LISTA:

ATIVIDADES NO MOODLE