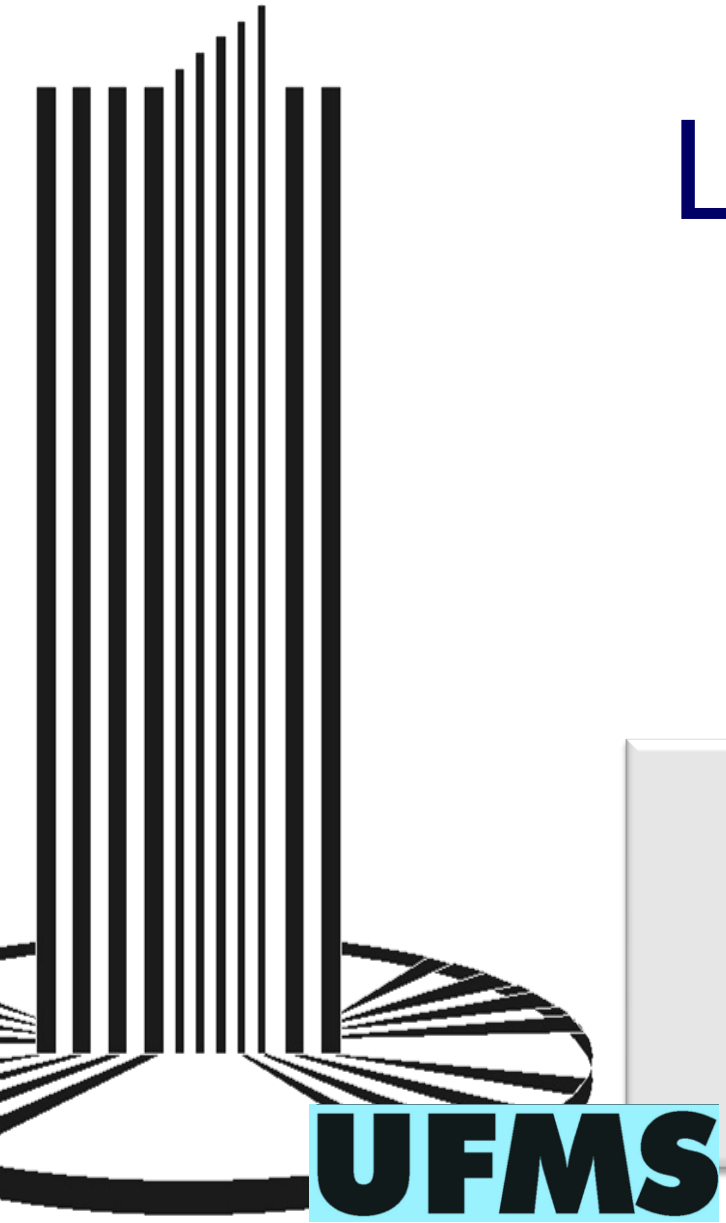


Lista Dinâmica Duplamente Encadeada

Profª Yorah Bosse

yorah.bosse@gmail.com
yorah.bosse@ufms.br

TAD Lista Dinâmica Duplamente Encadeada

- Operações possíveis
  - Incluir em qualquer lugar da lista
  - Excluir qualquer elemento
  - Mostrar os dados da lista em qualquer sentido
  - Verificar se a lista está Cheia
  - Verificar se a lista está Vazia
  - Ordenar os dados em qualquer ordem
  - Pesquisar algum elemento
  - Alterar dados
  - (entre outras)

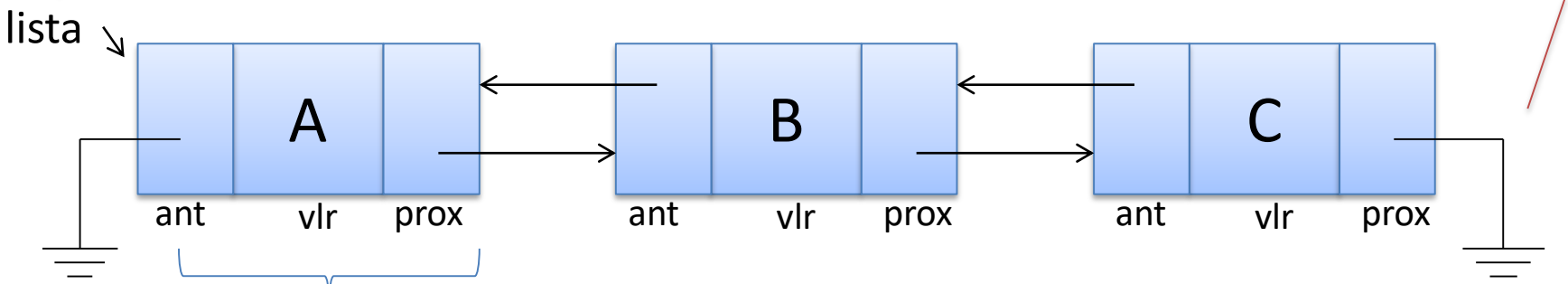
Nó

- Um nó é um espaço da lista destinado a armazenar algum valor
- A LDDE é normalmente representada pelo desenho abaixo:

(a)

Variável do tipo ponteiro que possui o endereço do primeiro nó da lista

Este símbolo é a representação do NULL, ou seja, indicando que não está apontando para lugar algum. Poderá se representado também por uma barra "/"



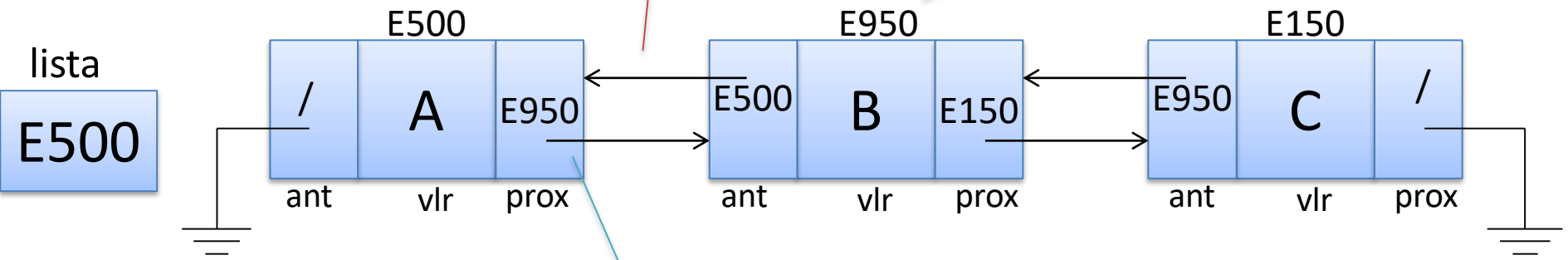
Um nó da lista que possui armazenado no mínimo um valor e, obrigatoriamente, o endereço do próximo elemento, ou seja, o campo "prox" é do tipo ponteiro

- Variações na representação:

(b)

As setas mostram onde se encontra o endereço armazenado no campo ant e prox

Endereço fictício da variável dinâmica onde se encontra o campo vlr (com o B), o campo ant (com E500) e prox (com E150)



A seta e o endereço são duas formas de representar a mesma coisa, ou seja, no campo prox existe somente um valor que é o endereço

Implementação – LDDE - Criação

- A estrutura necessária para a criação do nó da lista é:

```
typedef struct dadosLDDE{
        char vlr;
        struct dadosLDDE *prox,
                         *ant;
}sLDDE;
```

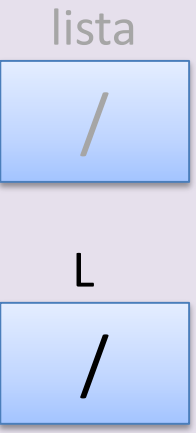
- Para criar a lista é declarada uma variável do tipo ponteiro desta estrutura, que deve ser inicializada com NULL:

```
sLDDE *lista;
lista = inicializa(lista);
```

```
sLDDE *inicializa(sLDDE *L){
      L = NULL;
      return L;
}
```

Implementação – LDDE – Lista Vazia?

Visualização Gráfica

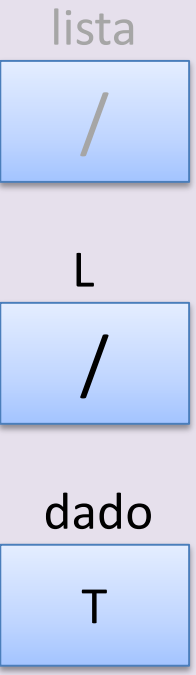


- Quando a lista está vazia, a variável do tipo ponteiro terá NULL armazenado dentro dela. A função para verificar esta condição da lista é:

```
int estaVazia(sLDDE *L){
      return (L == NULL?1:0);
}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica

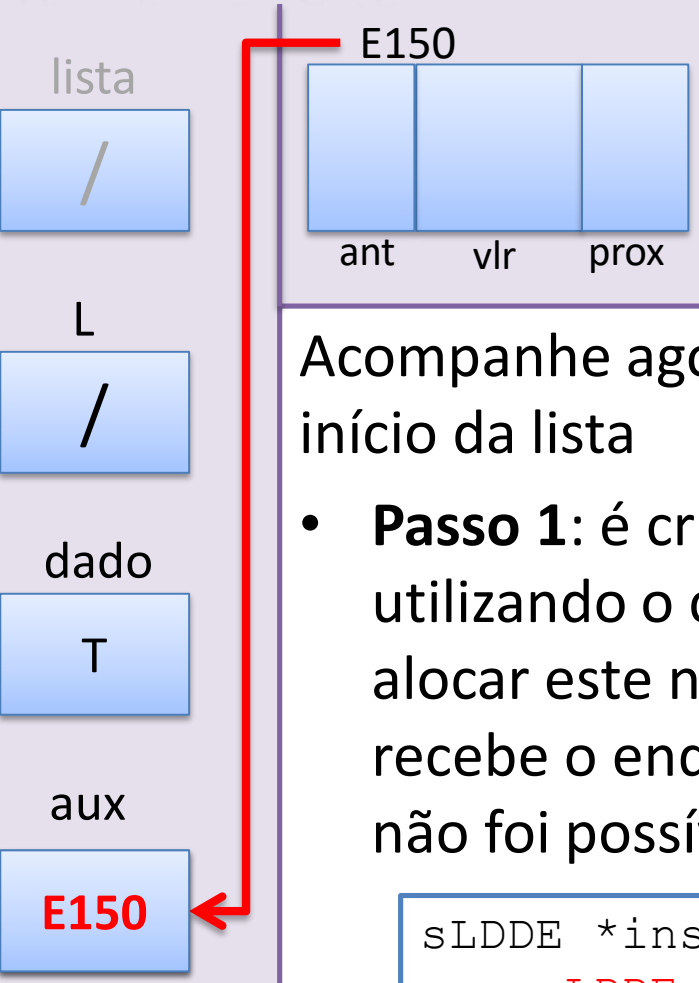


- Para a função que faz somente a inserção de um elemento na lista, são passados dois parâmetros:
  - a lista onde será feita a inserção
  - o dado a ser inserido
- A função retornará a lista onde a inserção foi realizada para dentro da variável “lista”.
- O dado poderá ser inserido em qualquer posição da lista.
- Veremos a seguir a forma como são inseridos os valores no início da lista

```
sLDDE *inserir_inicio(sLDDE *L, char dado){...}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica



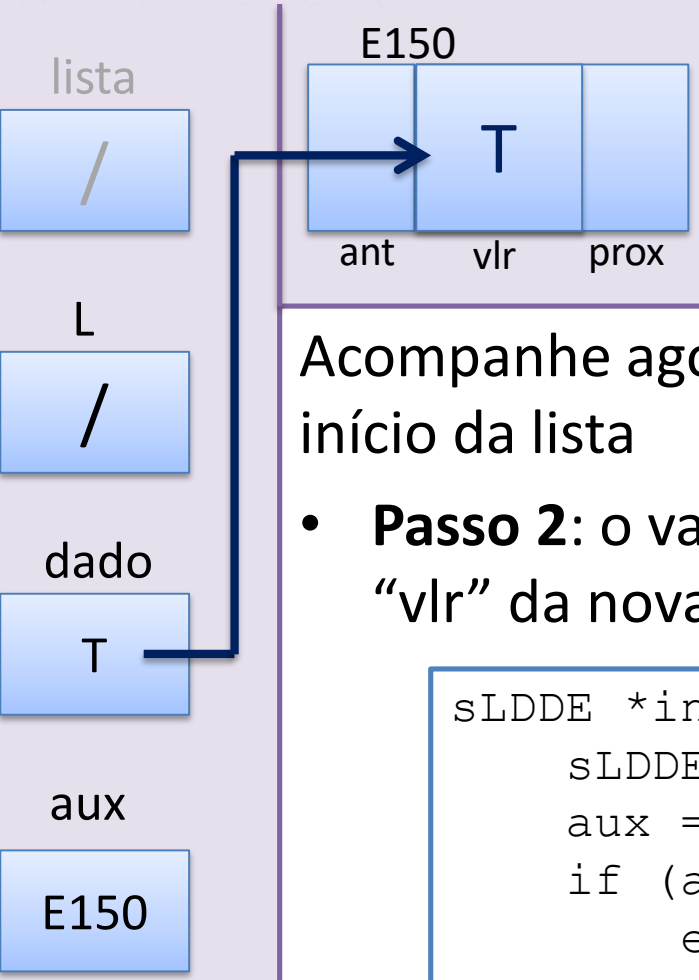
Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 1**: é criado um novo nó (variável dinâmica) , utilizando o comando malloc, e verificado se foi possível alocar este novo espaço na memória, ou seja, se aux, que recebe o endereço do novo espaço, tiver NULL é porque não foi possível.

```
sLDDE *inserir_inicio(sLDDE *L, char dado){
     sLDDE *aux;
     aux = (sLDDE *) malloc (sizeof(sLDDE));
     if (aux == NULL)
          exit (1);
}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica



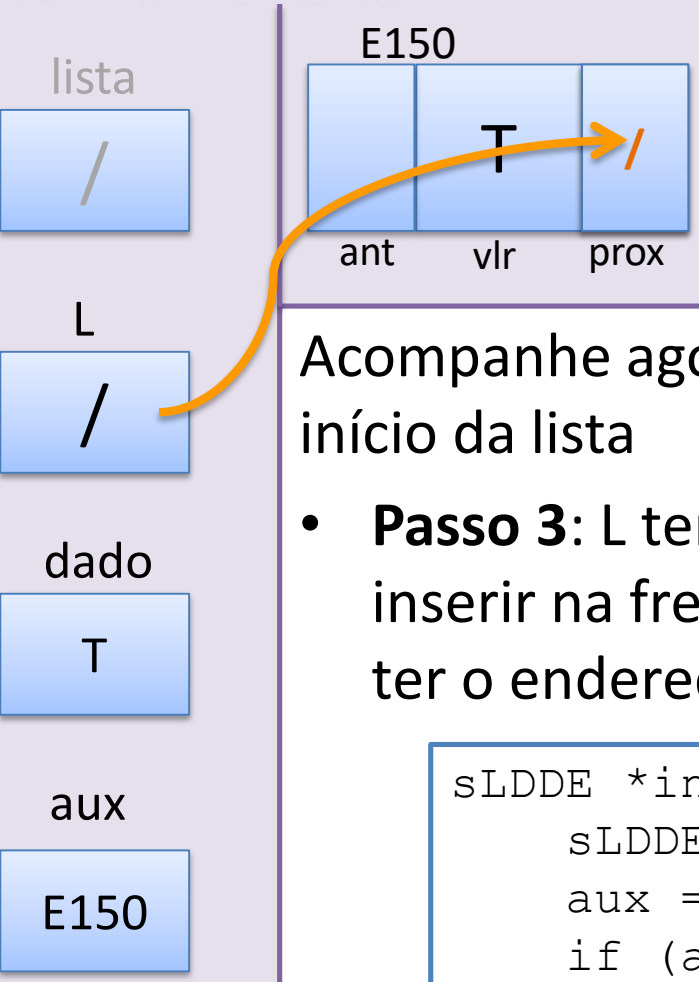
Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 2**: o valor da variável "dado" é inserido no campo "vlr" da nova variável dinâmica.

```
sLDDE *inserir_inicio(sLDDE *L, char dado){
     sLDDE *aux;
     aux = (sLDDE *) malloc (sizeof(sLDDE));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica



Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 3**: L tem o endereço do primeiro nó da lista. Ao inserir na frente deste, o campo prox do novo nó deverá ter o endereço contido em L

```
sLDDE *inserir_inicio(sLDDE *L, char dado){
     sLDDE *aux;
     aux = (sLDDE *) malloc (sizeof(sLDDE));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = L;
}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica

lista

/

L

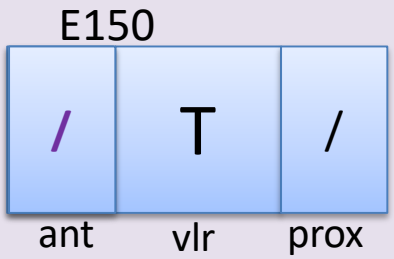
/

dado

T

aux

E150



Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 4**: como este será o primeiro elemento da lista, não haverá nenhum antes dele, logo, o campo "ant" receberá NULL

```
sLDDE *inserir_inicio(sLDDE *L, char dado){
     sLDDE *aux;
     aux = (sLDDE *) malloc (sizeof(sLDDE));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = L;
     aux->ant = NULL;
}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica

lista

/

L

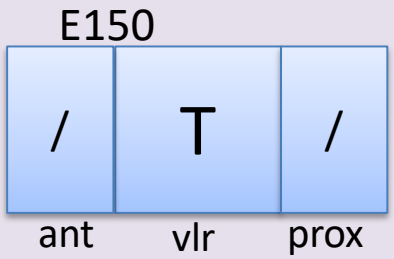
/

dado

T

aux

E150



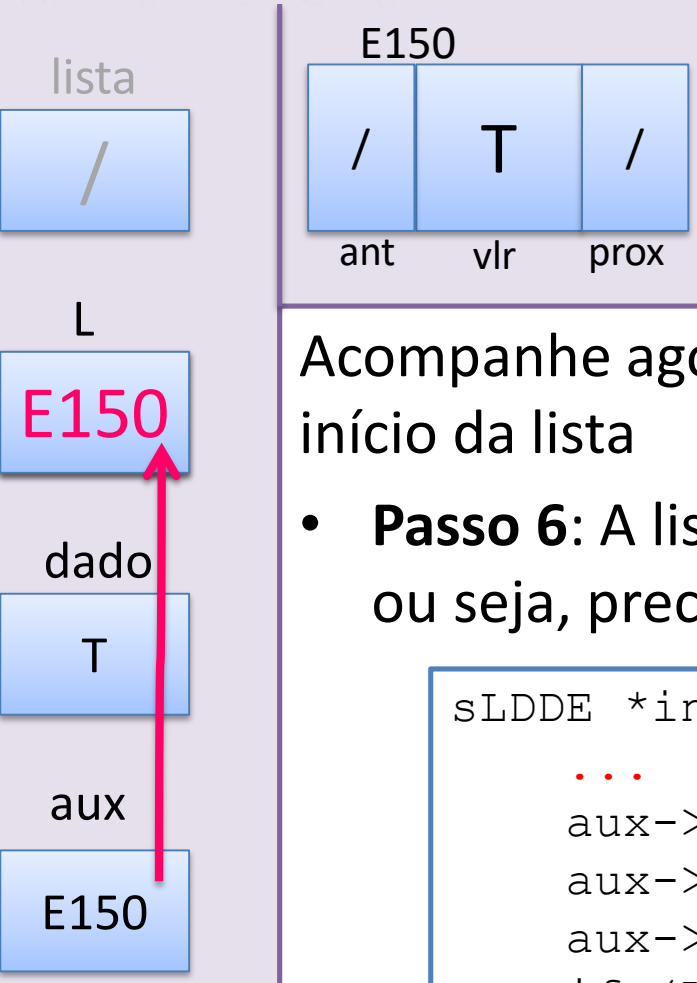
Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 5**: Se já existia pelo menos um elemento na lista, o primeiro precisa estar ligado ao novo elemento pelo campo “ant”

```
sLDDE *inserir_inicio(sLDDE *L, char dado){
      ...
      aux->vlr = dado;
      aux->prox = L;
      aux->ant = NULL;
      if (L != NULL)
          L->ant = aux;
}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica



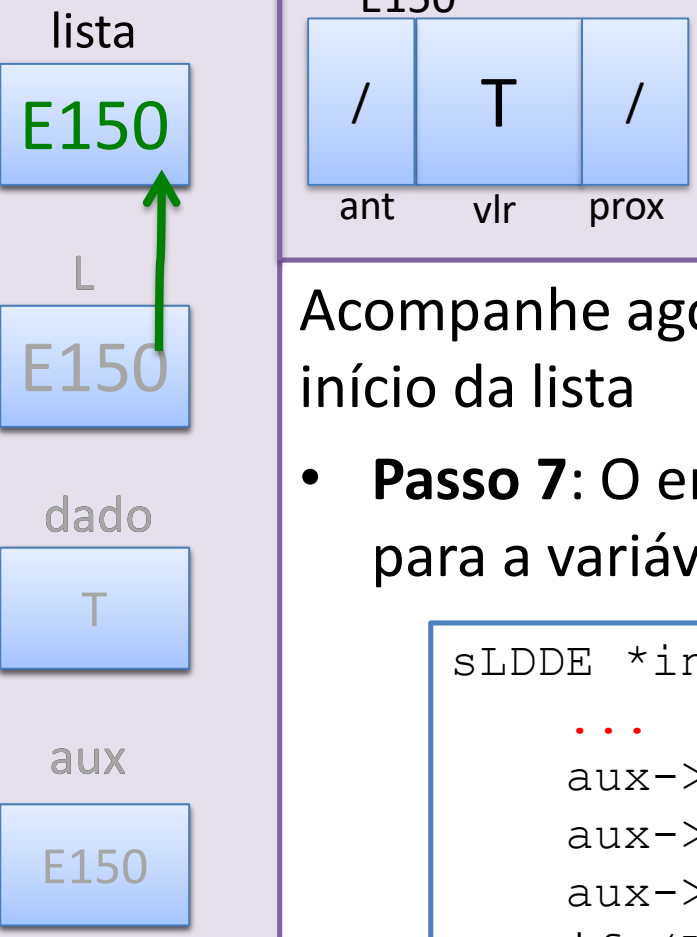
Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 6**: A lista L precisa apontar para este nova elemento, ou seja, precisa receber o endereço dele que está em aux

```
sLDDE *inserir_inicio(sLDDE *L, char dado){
      ...
      aux->vlr = dado;
      aux->prox = L;
      aux->ant = NULL;
      if (L != NULL)
          L->ant = aux;
      L = aux;
}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica



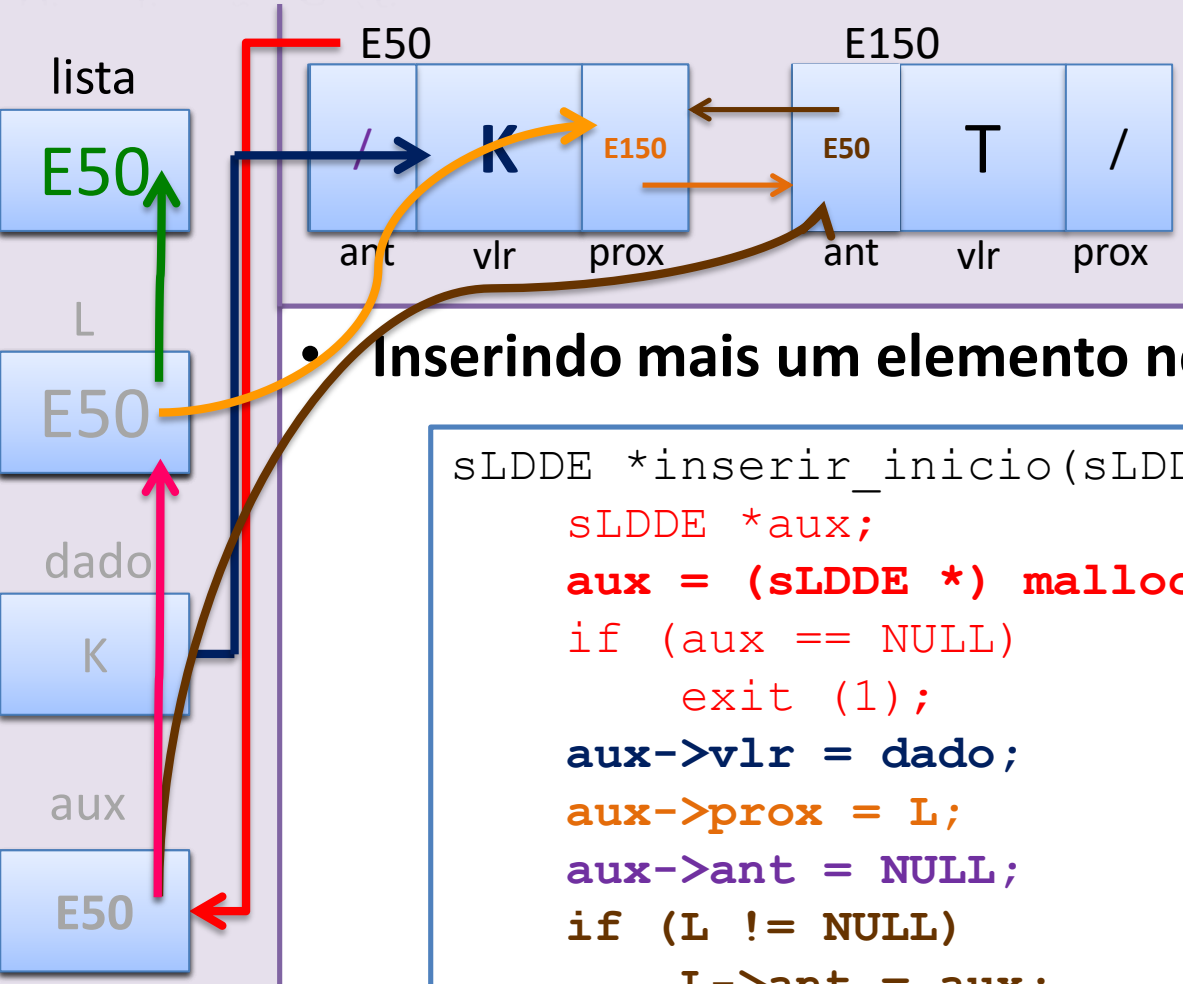
Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 7**: O endereço da lista com o novo dado é retornado para a variável "lista" e as variáveis locais são eliminadas

```
sLDDE *inserir_inicio(sLDDE *L, char dado){
     ...
     aux->vlr = dado;
     aux->prox = L;
     aux->ant = NULL;
     if (L != NULL)
          L->ant = aux;
     L = aux;
     return L;
}
```

Implementação – LDDE – Inserir Início da Lista

Visualização Gráfica

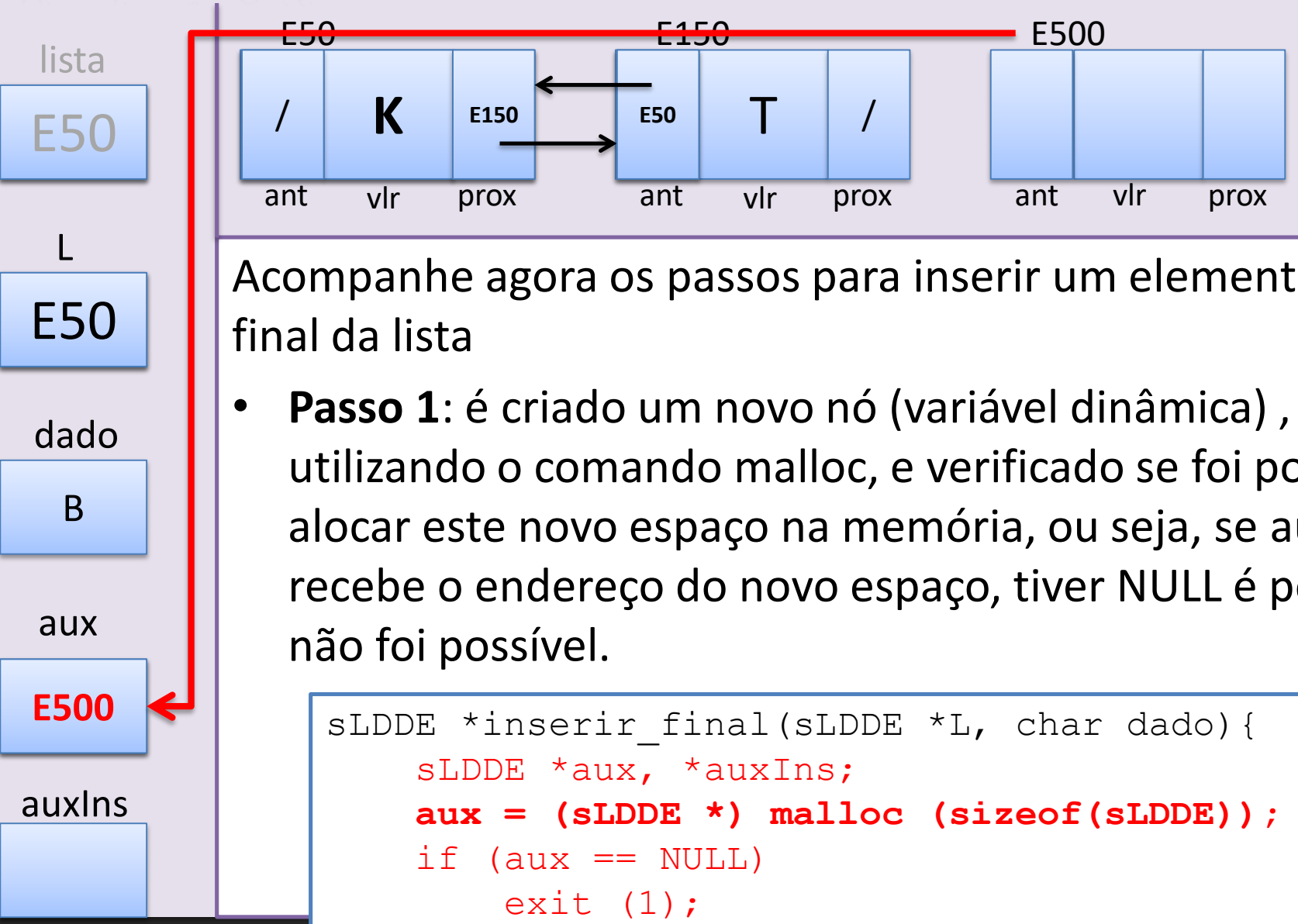


- **Inserindo mais um elemento no início da lista**

```
sLDDE *inserir_inicio(sLDDE *L, char dado){
     sLDDE *aux;
     aux = (sLDDE *) malloc (sizeof(sLDDE));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = L;
     aux->ant = NULL;
     if (L != NULL)
          L->ant = aux;
     L = aux;
     return L;
}
```

Implementação – LDDE – Inserir Final da Lista

Visualização Gráfica



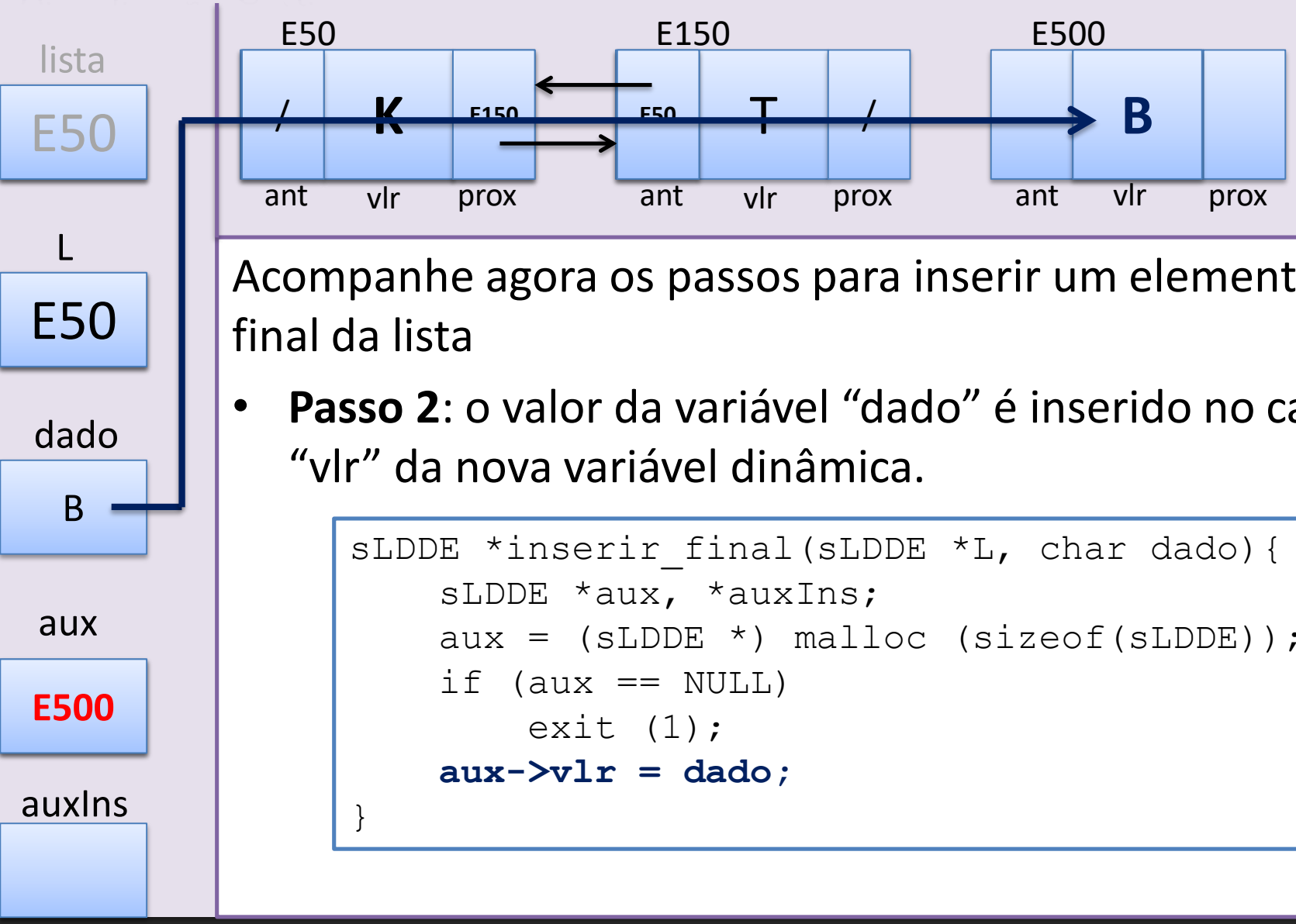
Acompanhe agora os passos para inserir um elemento no final da lista

- **Passo 1**: é criado um novo nó (variável dinâmica) , utilizando o comando malloc, e verificado se foi possível alocar este novo espaço na memória, ou seja, se aux, que recebe o endereço do novo espaço, tiver NULL é porque não foi possível.

```
sLDDE *inserir_final(sLDDE *L, char dado){
     sLDDE *aux, *auxIns;
     aux = (sLDDE *) malloc (sizeof(sLDDE));
     if (aux == NULL)
          exit (1);
}
```

Implementação – LDDE – Inserir Final da Lista

Visualização Gráfica



Acompanhe agora os passos para inserir um elemento no final da lista

- **Passo 2**: o valor da variável "dado" é inserido no campo "vlr" da nova variável dinâmica.

```
sLDDE *inserir_final(sLDDE *L, char dado){
     sLDDE *aux, *auxIns;
     aux = (sLDDE *) malloc (sizeof(sLDDE));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
}
```

Implementação – LDDE – Inserir Final da Lista

Visualização Gráfica

lista

E50

L

E50

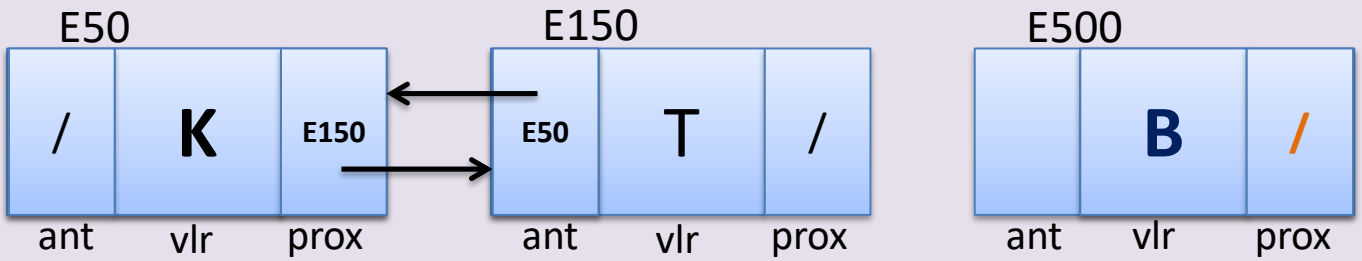
dado

B

aux

E500

auxIns



Acompanhe agora os passos para inserir um elemento no final da lista

- **Passo 3**: tornando-se este o último elemento da lista, seu campo "prox" receberá NULL

```
sLDDE *inserir_final(sLDDE *L, char dado){
     sLDDE *aux, *auxIns;
     aux = (sLDDE *) malloc (sizeof(sLDDE));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = NULL;
}
```

Implementação – LDDE – Inserir Final da Lista

Visualização Gráfica

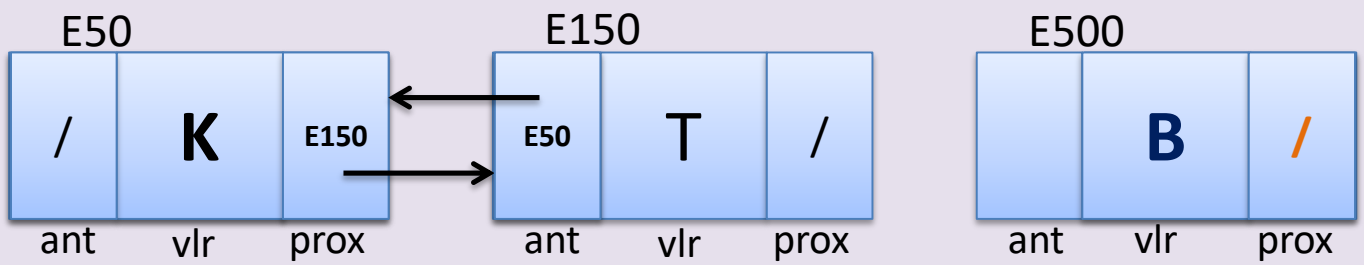
lista
E50

L
E50

dado
B

aux
E500

auxIns



Acompanhe agora os passos para inserir um elemento no final da lista

- **Passo 4**: se este for o primeiro elemento da lista, o campo "ant" será NULL e a lista "L" receberá este endereço.

```
sLDDE *inserir_final(sLDDE *L, char dado){
      ...
      aux->vlr = dado;
      aux->prox = NULL;
      if (estaVazia(L)){
            aux->ant = NULL;
            L = aux;
}
```

Implementação – LDDE – Inserir Final da Lista

Visualização Gráfica

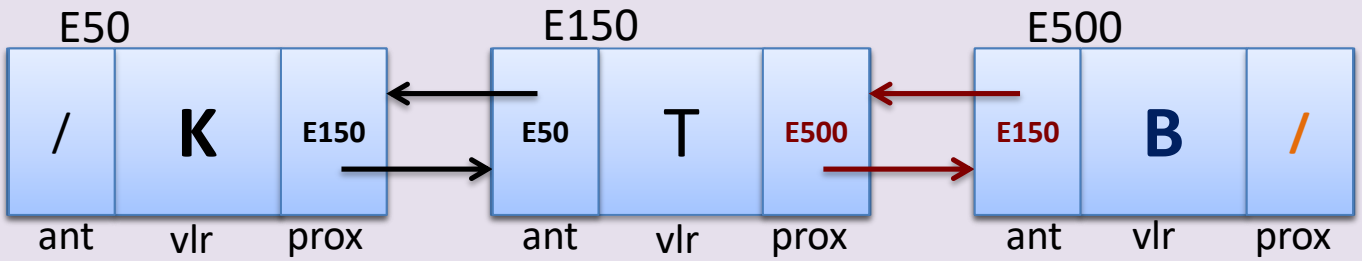
lista
E50

L
E50

dado
B

aux
E500

auxIns
E150



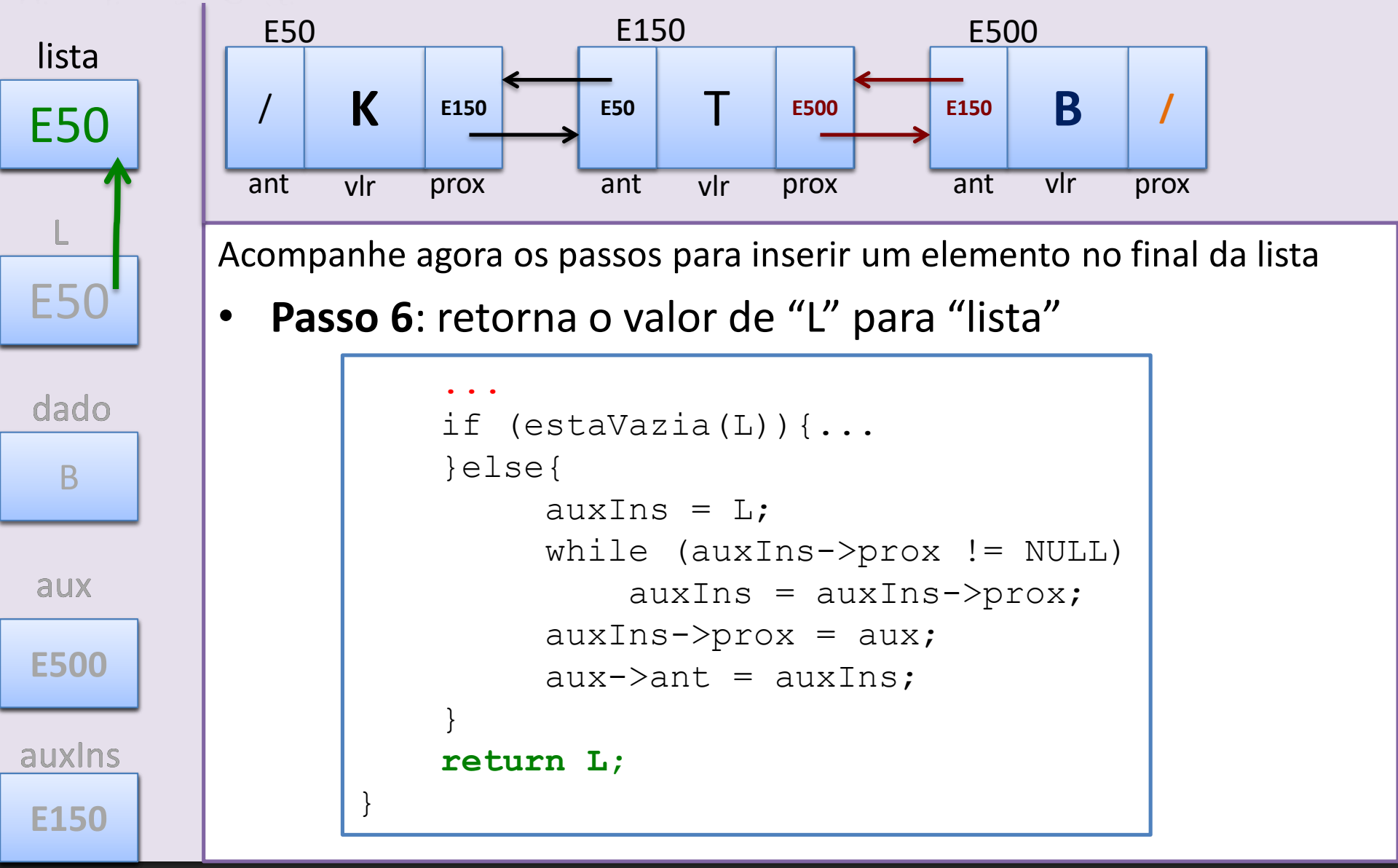
Acompanhe agora os passos para inserir um elemento no final da lista

- **Passo 5**: caso a lista não esteja vazia, deve-se procurar o último elemento, mudar o "prox" dele e fazer o"ant do novo dado apontar para ele.

```
    ...
    if (estaVazia(L)){...
    }else{
          auxIns = L;
          while (auxIns->prox != NULL)
               auxIns = auxIns->prox;
          auxIns->prox = aux;
          aux->ant = auxIns;
    }
}
```

Implementação – LDDE – Inserir Final da Lista

Visualização Gráfica



Acompanhe agora os passos para inserir um elemento no final da lista

- **Passo 6**: retorna o valor de "L" para "lista"

```
    ...
    if (estaVazia(L)){...
    }else{
          auxIns = L;
          while (auxIns->prox != NULL)
              auxIns = auxIns->prox;
          auxIns->prox = aux;
          aux->ant = auxIns;
    }
    return L;
}
```

Implementação – LDDE – Listar

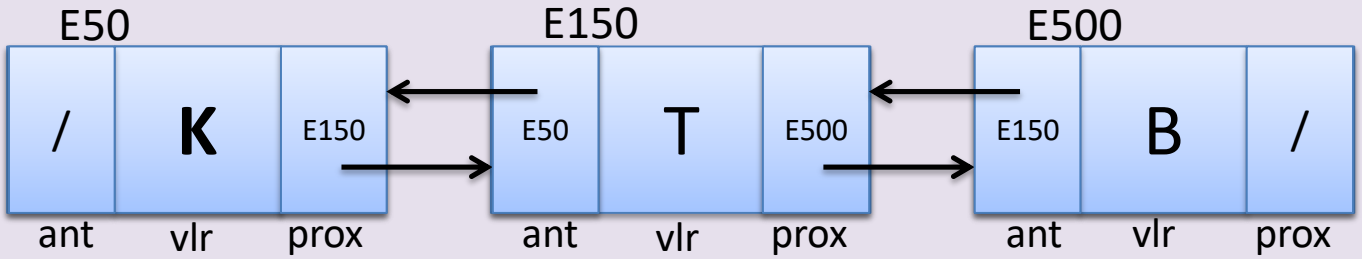
Visualização Gráfica

lista

E50

L

E50



- **Mostrando os elementos da lista**

```
void listar(sLDDE *L){
     if (estaVazia(L))
        printf("\nLista Vazia!\n\n");
     else{
        while (L != NULL){
              printf("%c    ",L->vlr);
              L = L->prox;
        }
     }
}
```

Ao iniciar a função “L” é criada e recebe o endereço de “lista”. Se a lista não tiver vazia é mostrado o valor e “L” recebe o valor do campo “prox” . Isto ocorre até que “L” seja igual a NULL.

Implementação – LDDE – Listar

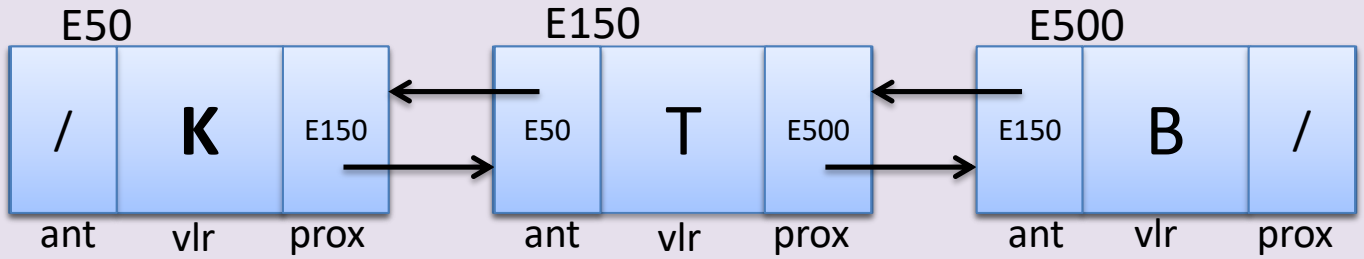
Visualização Gráfica

lista

E50

L

/



- **Mostrando os elementos da lista**

```
void listar(sLDDE *L){
     if (estaVazia(L))
        printf("\nLista Vazia!\n\n");
     else{
        while (L != NULL){
              printf("%c   ",L->vlr);
              L = L->prox;
        }
     }
}
```

Implementação – LDDE – Pesquisar

Visualização Gráfica

lista

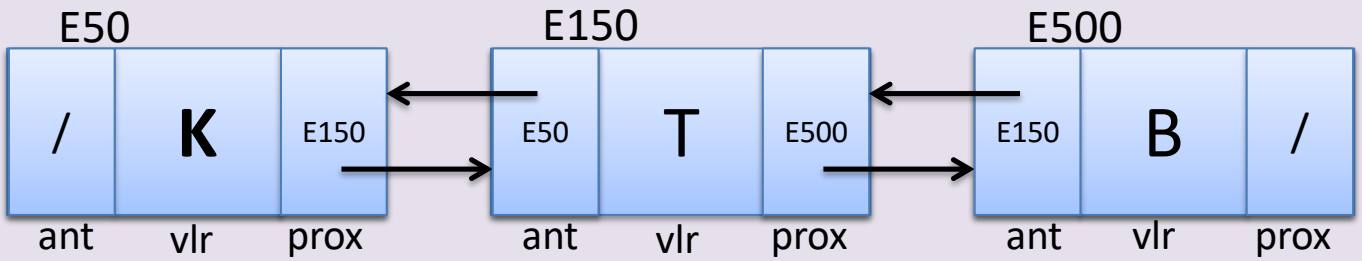
E50

L

E500

dadoPesq

B



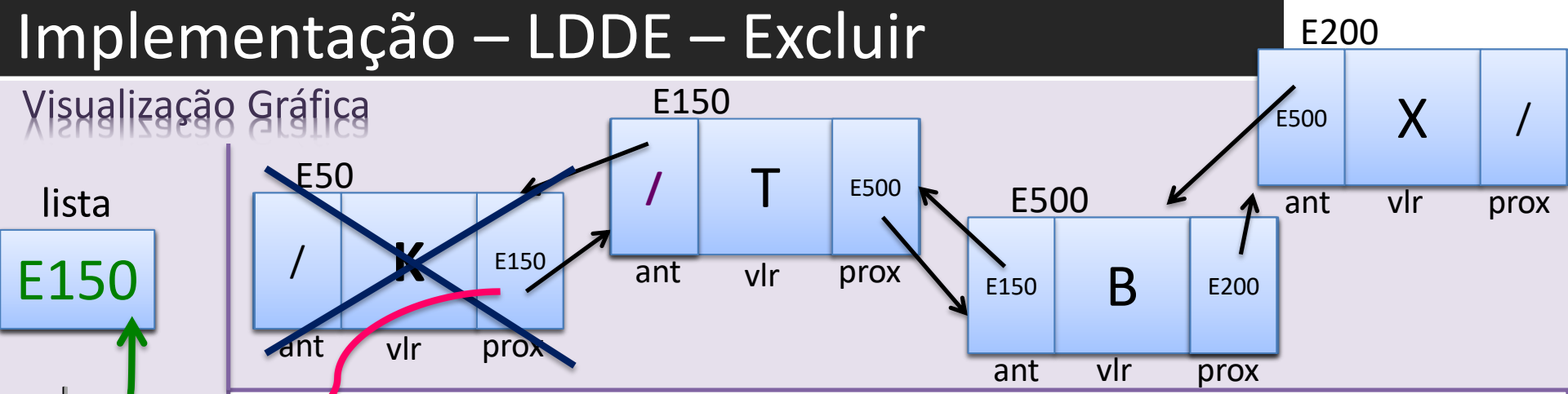
- **Procurando um determinado elementos na lista**

```
sLDDE* pesq(sLDDE *L, char dadoPesq){
    while (L != NULL && L->vlr != dadoPesq)
        L = L->prox;
    return L;
}
```

A função de pesquisa recebe o endereço da lista a ser realizada a pesquisa e o valor a ser procurado. Ela retorna o endereço onde se encontra o valor procurado ou NULL, caso ele não exista. Este retorno não é colocado na variável "lista"

Implementação – LDDE – Excluir

Visualização Gráfica

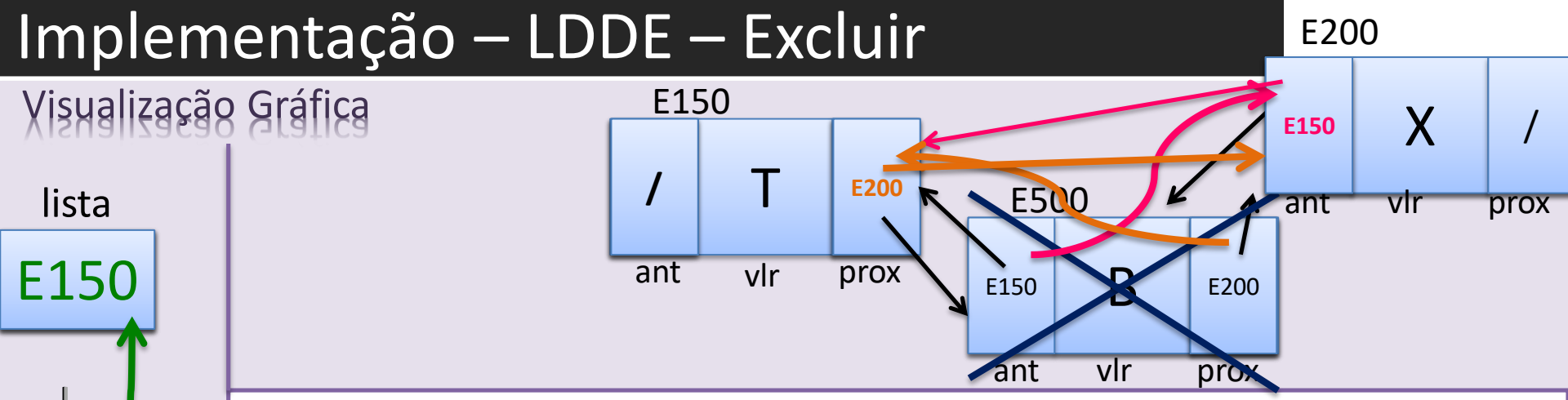


- **Excluindo o primeiro elemento da lista**

```
sLDDE* excluir(sLDDE *L, char dadoExc){
      sLDSE *aux;
      if (estaVazia(L))
         printf("\n\nLista VAZIA!\n\n");
      else{
         aux = pesq(L,dadoExc);
         if (aux == L){
             if (aux->prox != NULL)
                aux->prox->ant = NULL;
             L = L->prox;
         }else{
            ...
         }
         free(aux);
      }
      return L;
}
```

Implementação – LDDE – Excluir

Visualização Gráfica



- **Excluindo o primeiro elemento da lista**

```
sLDDE* excluir(sLDDE *L, char dadoExc){
      sLDSE *aux;
      if (estaVazia(L))
         printf("\n\nLista VAZIA!\n\n");
      else{
         aux = pesq(L,dadoExc);
         if (aux == L){
              ...
         }else{
              if (aux->prox != NULL)
                   aux->prox->ant = aux->ant;
              aux->ant->prox = aux->prox;
         }
         free(aux);
      }
      return L;
}
```

Implementação – LDDE – Excluir

- **A função completa ficará:**

```
sLDDE* excluir(sLDDE *L, char dadoExc){
      sLDSE *aux;
      if (estaVazia(L))
         printf("\n\nLista VAZIA!\n\n");
      else{
         aux = pesq(L,dadoExc);
         if (aux == L){
             if (aux->prox != NULL)
                aux->prox->ant = NULL;
             L = L->prox;
         }else{
             if (aux->prox != NULL)
                 aux->prox->ant = aux->ant;
             aux->ant->prox = aux->prox;
         }
         free(aux);
      }
      return L;
}
```

Vantagens

- **Vantagens:**
  - Facilidade de inserir ou remover elementos do meio da lista:
    - Como os elementos não precisam estar armazenados em posições consecutivas de memória, nenhum dado precisa ser movimentado,
    - bastando atualizar o campo prox que precede aquele inserido ou removido
  - Útil em aplicações em que o tamanho máximo da lista não precisa ser definido a priori
  - Facilidade de caminhar nos dois sentidos da lista

Desvantagens

- **Desvantagens:**
  - Surge quando desejamos acessar uma posição especifica dentro da lista
  - Neste caso, devemos partir do primeiro elemento e ir seguindo os campos de ligação, um a um, até atingir a posição desejada
  - Obviamente, para listas extensas, esta operação pode ter um alto custo em relação a tempo.

Fonte: http://www.lcmat.uenf.br/page_attachments/0000/0098/Aulas_2Marzo2010-EDI.pdf

Leitura Recomendada

DROSDEK, A. Estrutura de dados e algoritmos em C++.

Cengage: 2002.

- Página 80 – Listas Duplamente Ligadas

Exercício

- Faça uma função que receba uma lista dinâmica duplamente encadeada, um valor “X” a ser inserido (letra) e um valor de referência “R” (letra). Insira “X” após “R”. Caso “R” não exista, insira no final da lista.