Lista Dinâmica Simplesmente Encadeada

Profª Yorah Bosse

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

Formas de Implementação

- **Seqüencial**: Os elementos estão fisicamente contíguos, um após o outro. Pode ser representada por um vetor na memória principal ou um arquivo sequencial em disco.

- **Encadeada**:
  - Estática: Inicio da lista é definido por uma variável inteiro que estabelece onde começa o encadeamento. Cada elemento possui um campo do tipo int, que estabelece o sucessor do mesmo.
    - Desvantagens: Quantidade máxima de elementos estabelecida (Alocação Estática).
  - Dinâmica: Início da lista é definido por um apontador que armazena o endereço do elemento inicial da lista. Cada elemento da lista possui um campo do tipo apontador que armazena o endereço do próximo elemento da lista. Cada elemento da lista é alocado dinamicamente.

Listas Lineares

- Abaixo uma relação de algumas Listas Lineares :
  - Seqüêncial
  - Estática Simples ou duplamente Encadeada
  - Dinâmica Simples ou duplamente Encadeada
    - LDSE = Lista Dinâmica Simplesmente Encadeada
  - Dinâmica Simples ou duplamente Encadeada com descritor
  - Dinâmica Circular Simples ou duplamente Encadeada
  - Pilhas (stack) estáticas e dinâmicas simples ou duplamente Encadeadas
  - Filas (queue) estáticas e dinâmicas simples ou duplamente Encadeadas

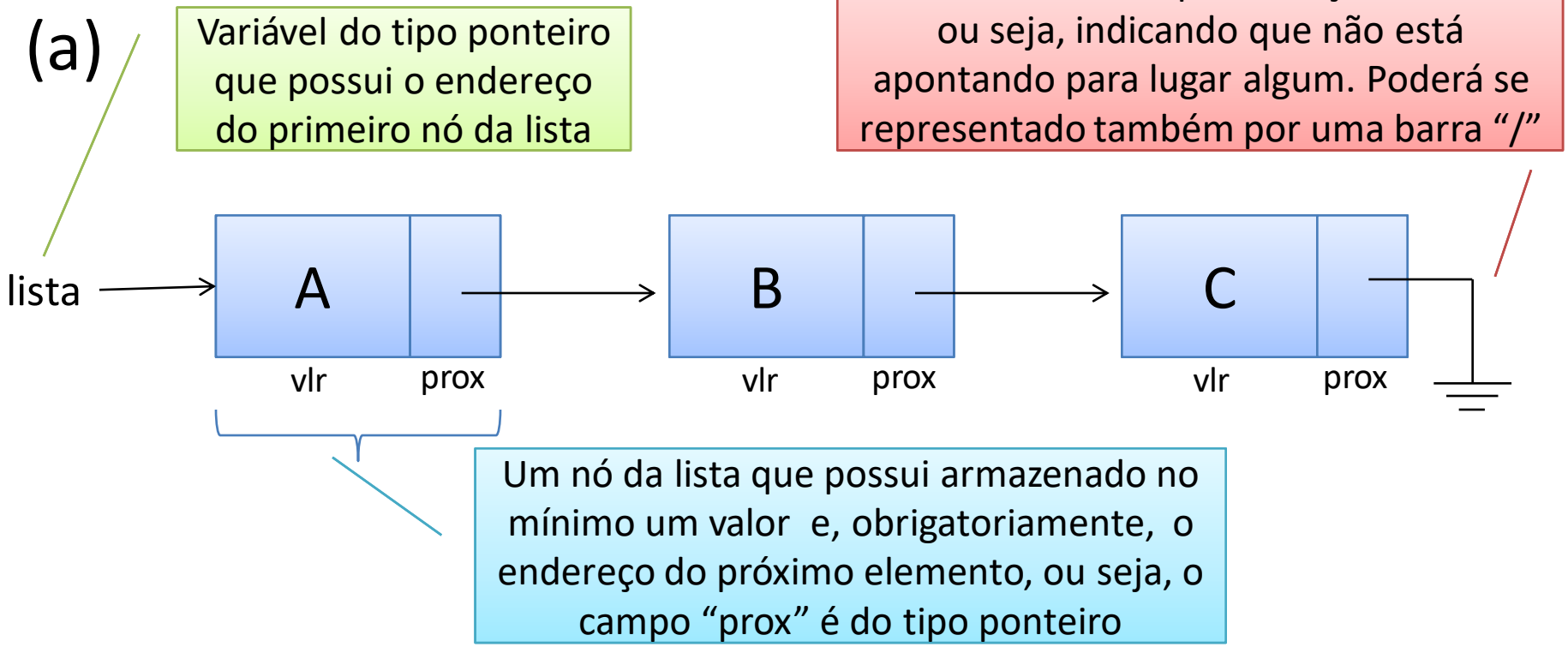
TAD Lista Dinâmica Simplesmente Encadeada

- Operações possíveis
  - Incluir em qualquer lugar da lista
  - Excluir qualquer elemento
  - Mostrar os dados da lista
  - 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

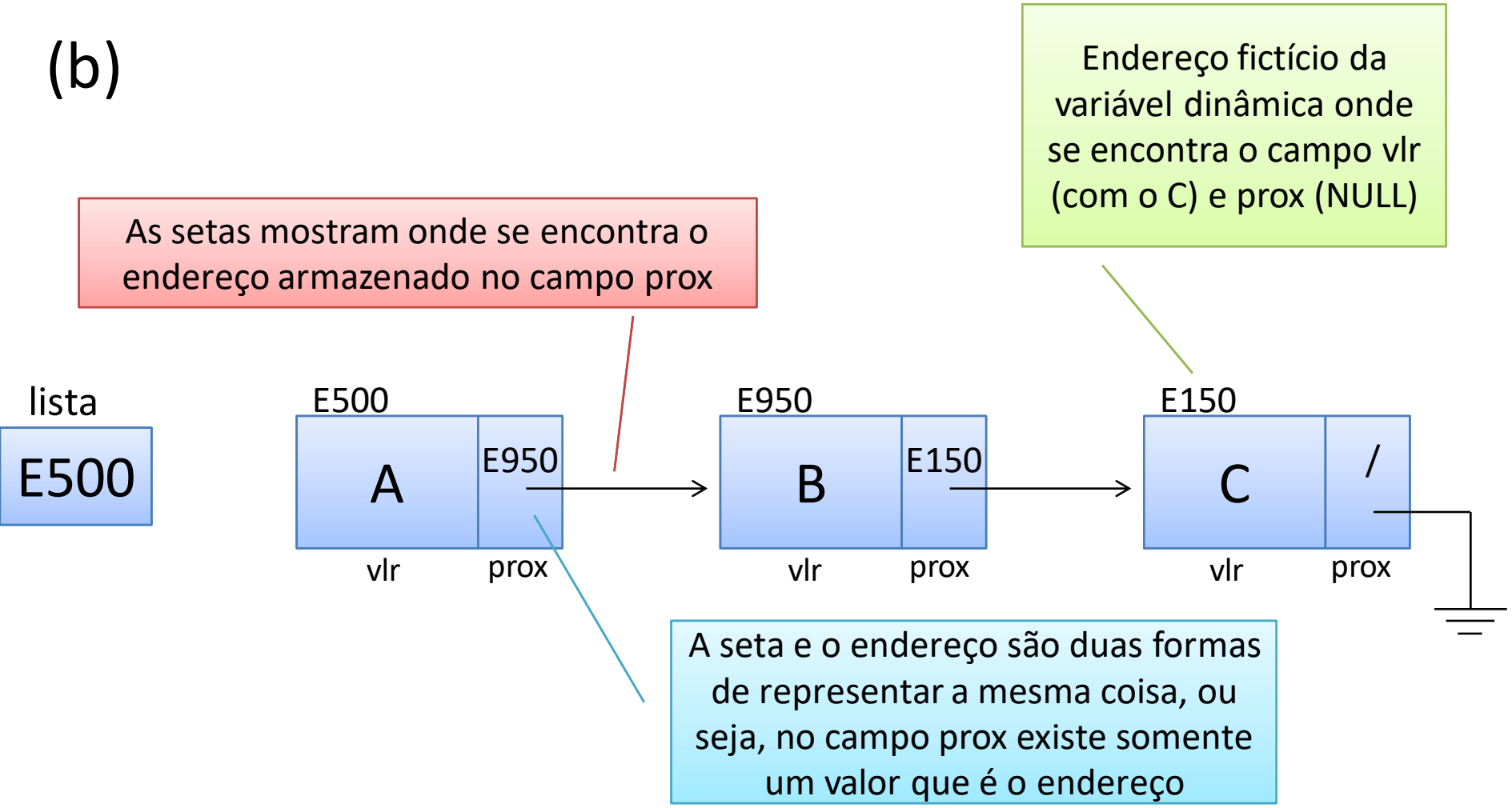
- Um nó é um espaço da lista destinado a armazenar algum valor
- A LDSE é normalmente representada pelo desenho abaixo:

(a)

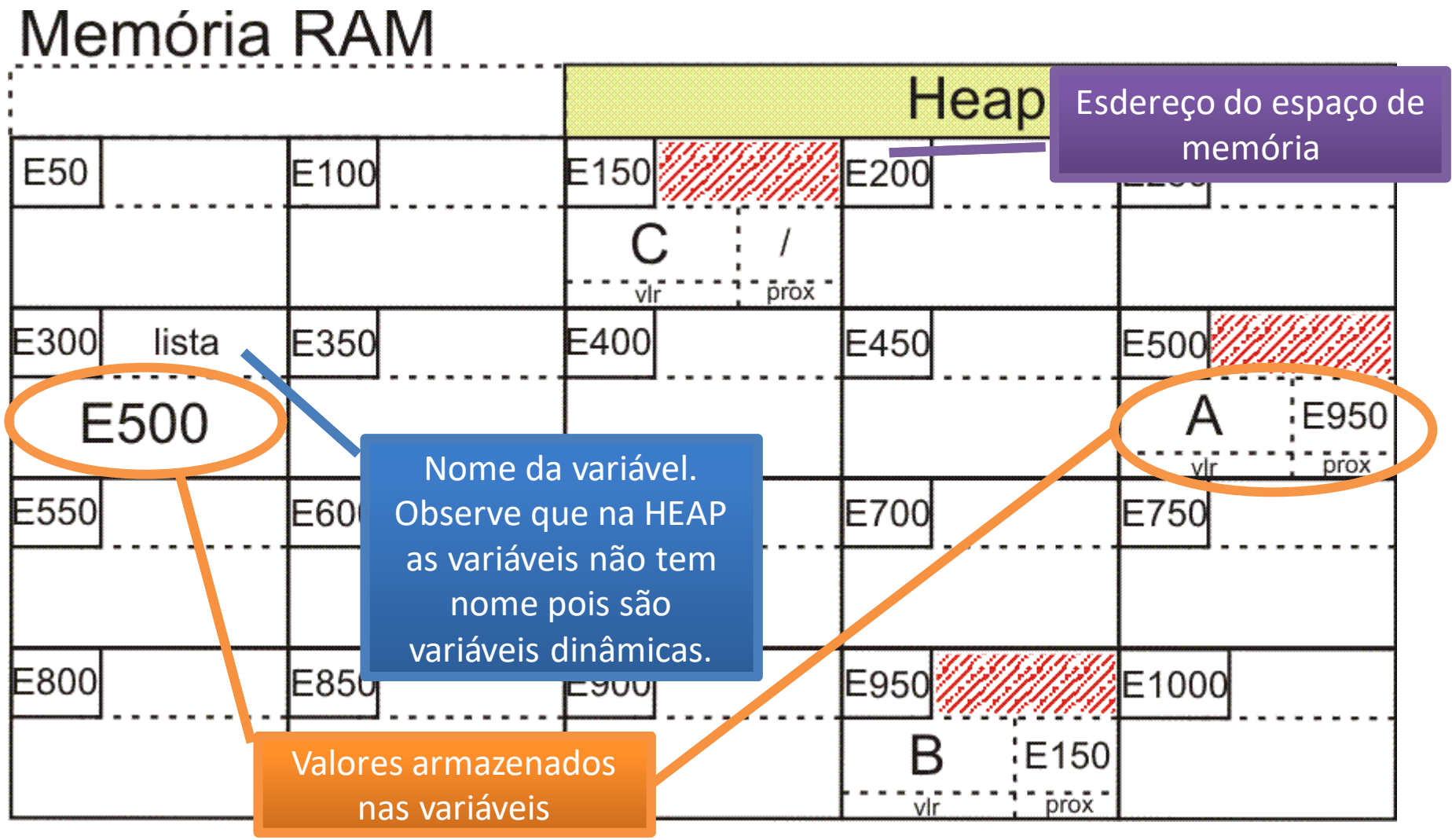


- Variações na representação:

(b)



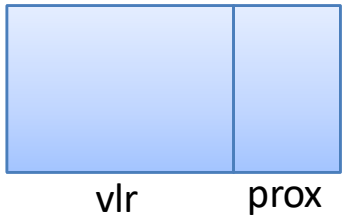
- Variações na representação: (c)



Implementação – LDSE - Criação

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

```
typedef struct dadosLDSE{
        char vlr;
        struct dadosLDSE *prox;
}sLDSE;
```



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

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

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

Implementação – LDSE – Lista Vazia?

Visualização Gráfica

lista

/

L

/

- 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(sLDSE *L){
     return (L == NULL?1:0);
}
```

Implementação – LDSE – Inserir Início da Lista

Visualização Gráfica

lista

/

L

/

dado

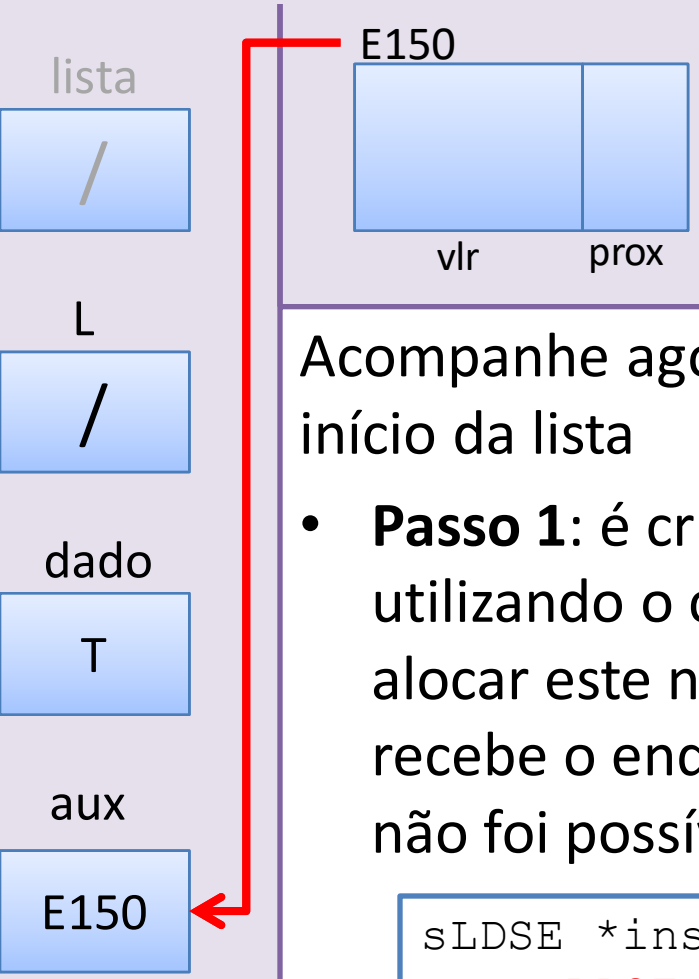
T

- 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

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

Implementação – LDSE – Inserir Início da Lista

Visualização Gráfica



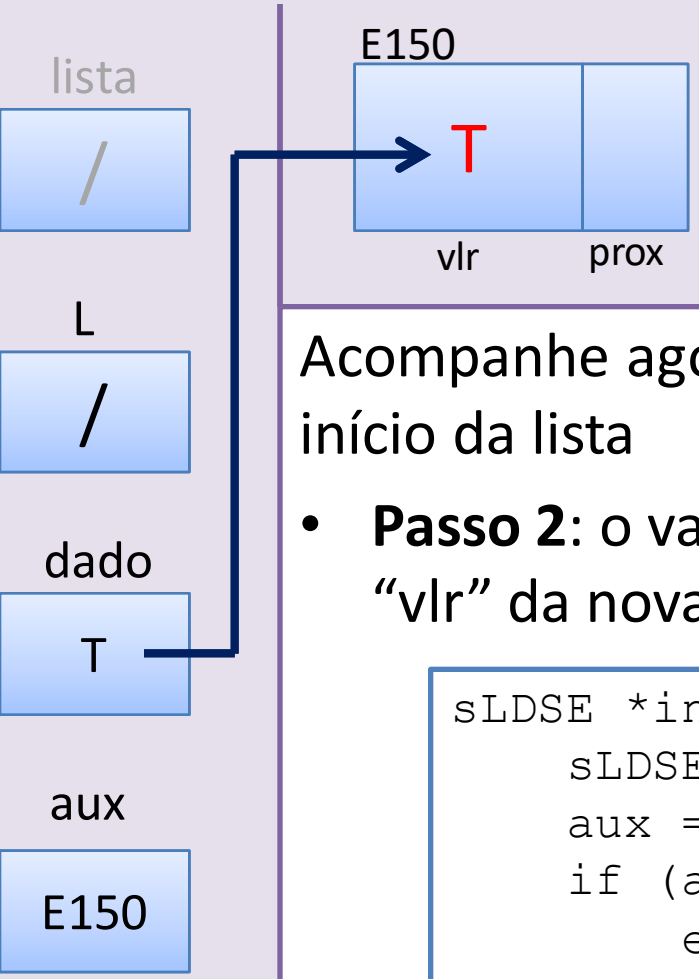
Acompanhe agora os passos para inserir um elemento no início da lista

- **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.

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

Implementação – LDSE – Inserir Início da Lista

Visualização Gráfica



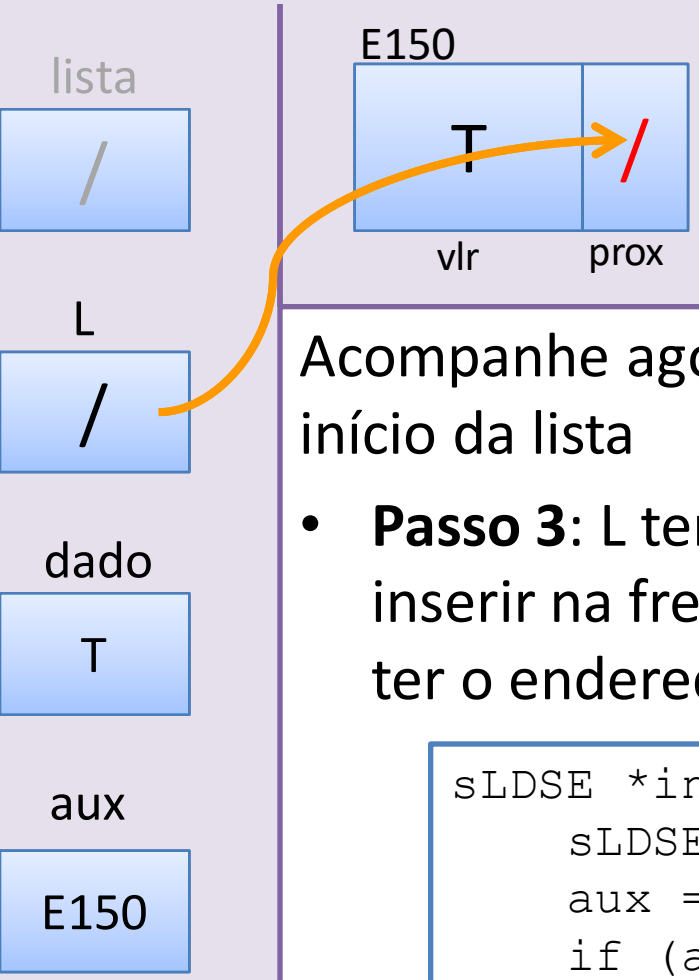
Acompanhe agora os passos para inserir um elemento no início da lista

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

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

Implementação – LDSE – Inserir Início da Lista

Visualização Gráfica



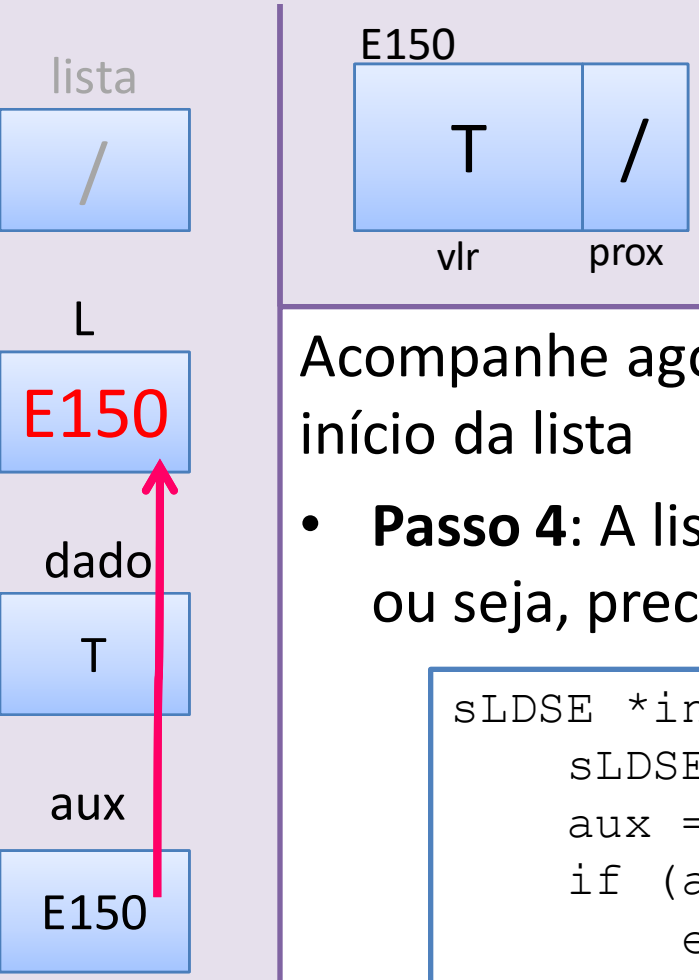
Acompanhe agora os passos para inserir um elemento no início da lista

- **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

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

Implementação – LDSE – Inserir Início da Lista

Visualização Gráfica



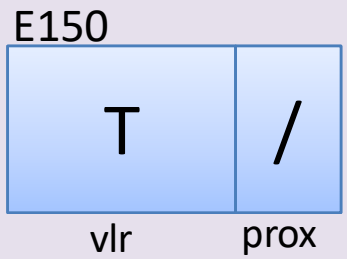
Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 4**: A lista L precisa apontar para este nova elemento, ou seja, precisa receber o endereço dele que está em aux

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

Implementação – LDSE – Inserir Início da Lista

Visualização Gráfica



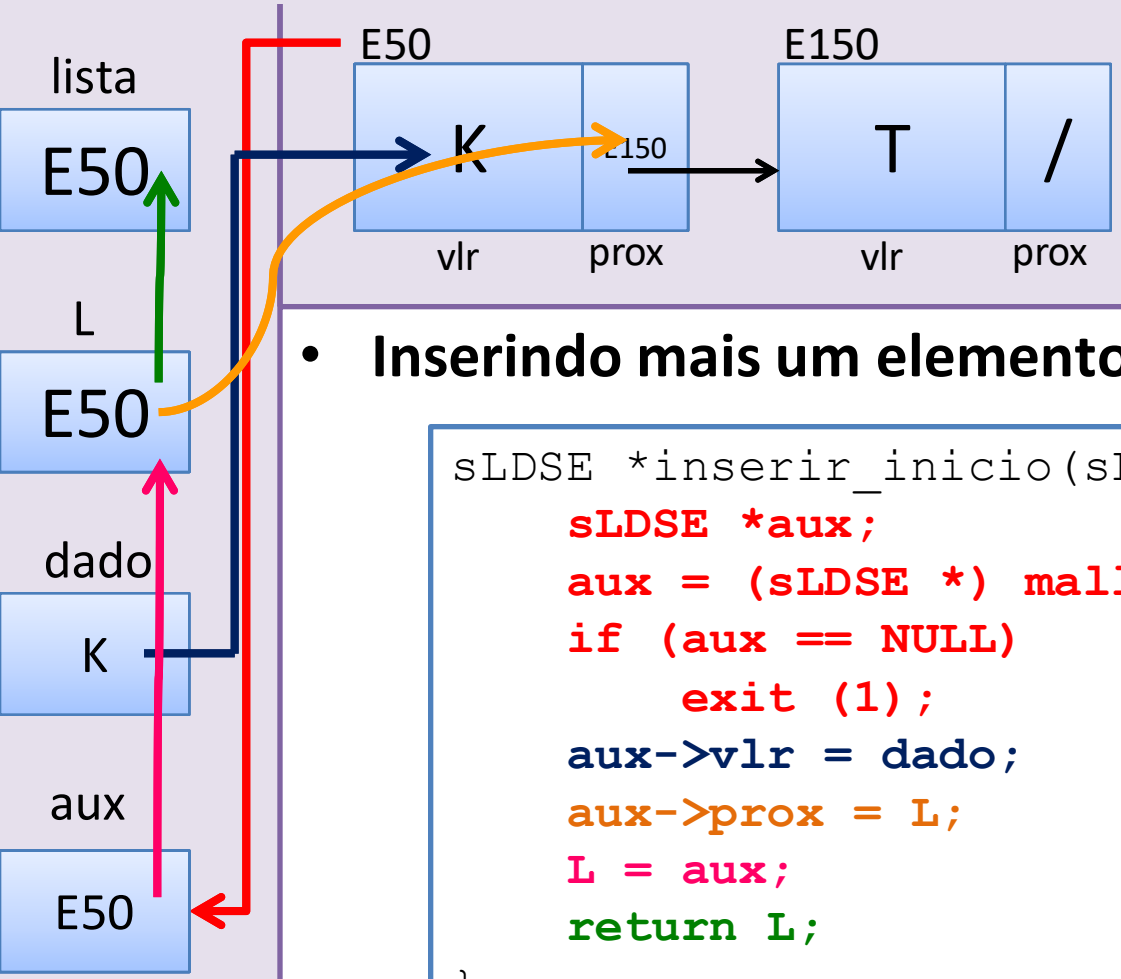
Acompanhe agora os passos para inserir um elemento no início da lista

- **Passo 5**: O endereço da lista com o novo dado é retornado para a variável "lista" e as variáveis locais são eliminadas

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

Implementação – LDSE – Inserir Início da Lista

Visualização Gráfica

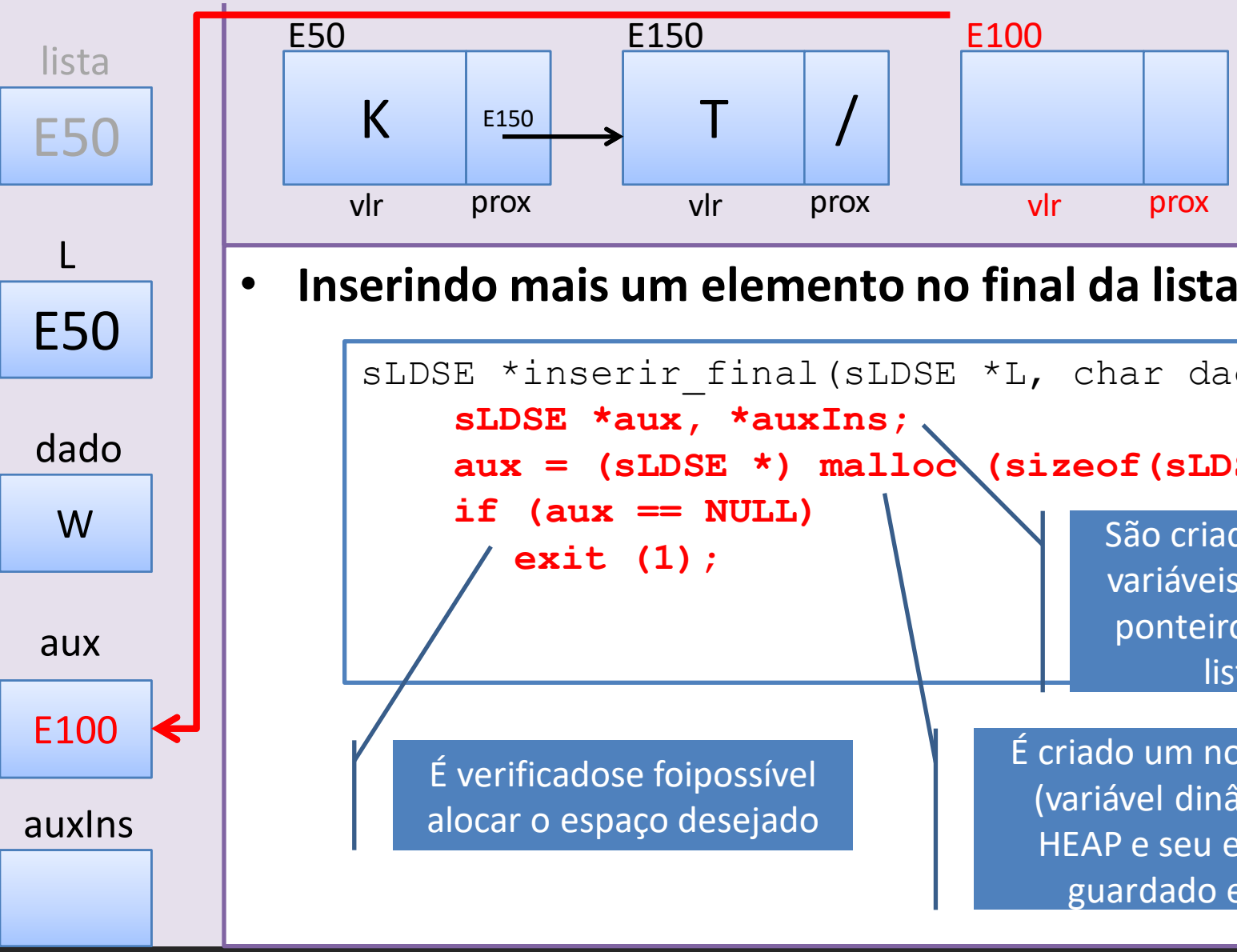


- **Inserindo mais um elemento no início da lista**

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

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica



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

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

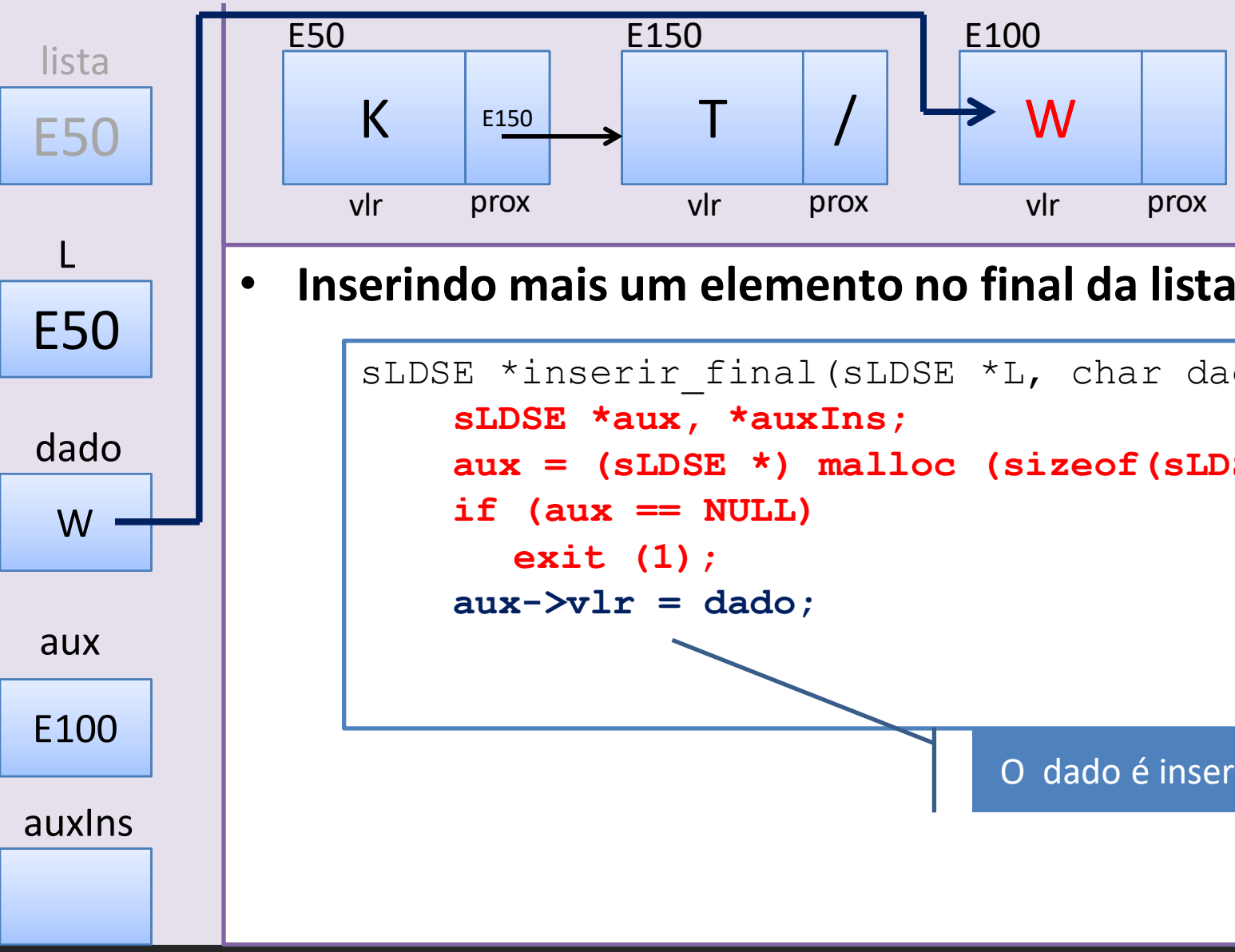
São criadas duas variáveis do tipo ponteiro para a lista

É verificadose foipossível alocar o espaço desejado

É criado um novo espaço (variável dinâmica) na HEAP e seu endereço guardado em aux

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica



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

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

O dado é inserido na lista

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica

lista

E50

L

E50

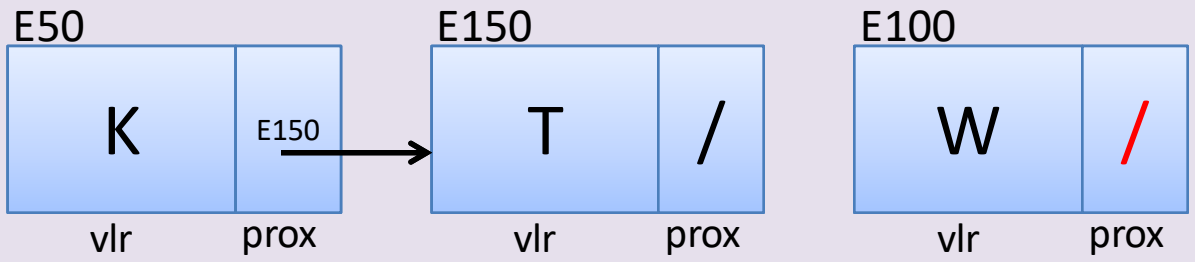
dado

W

aux

E100

auxIns



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

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

O campo prox apontado por aux recebe NULL

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica

lista

E50

L

E50

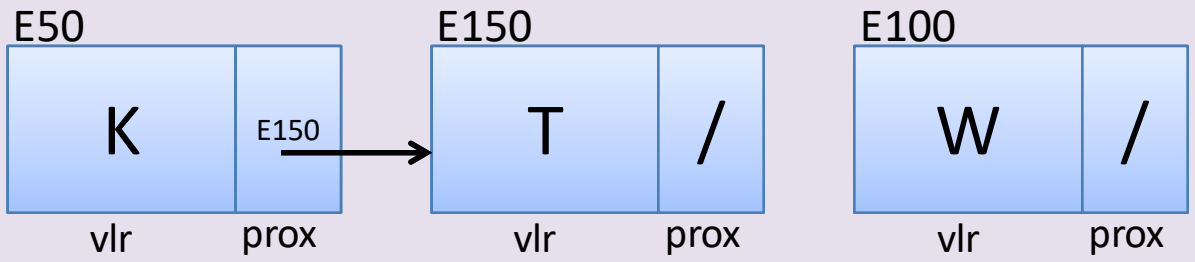
dado

W

aux

E100

auxIns



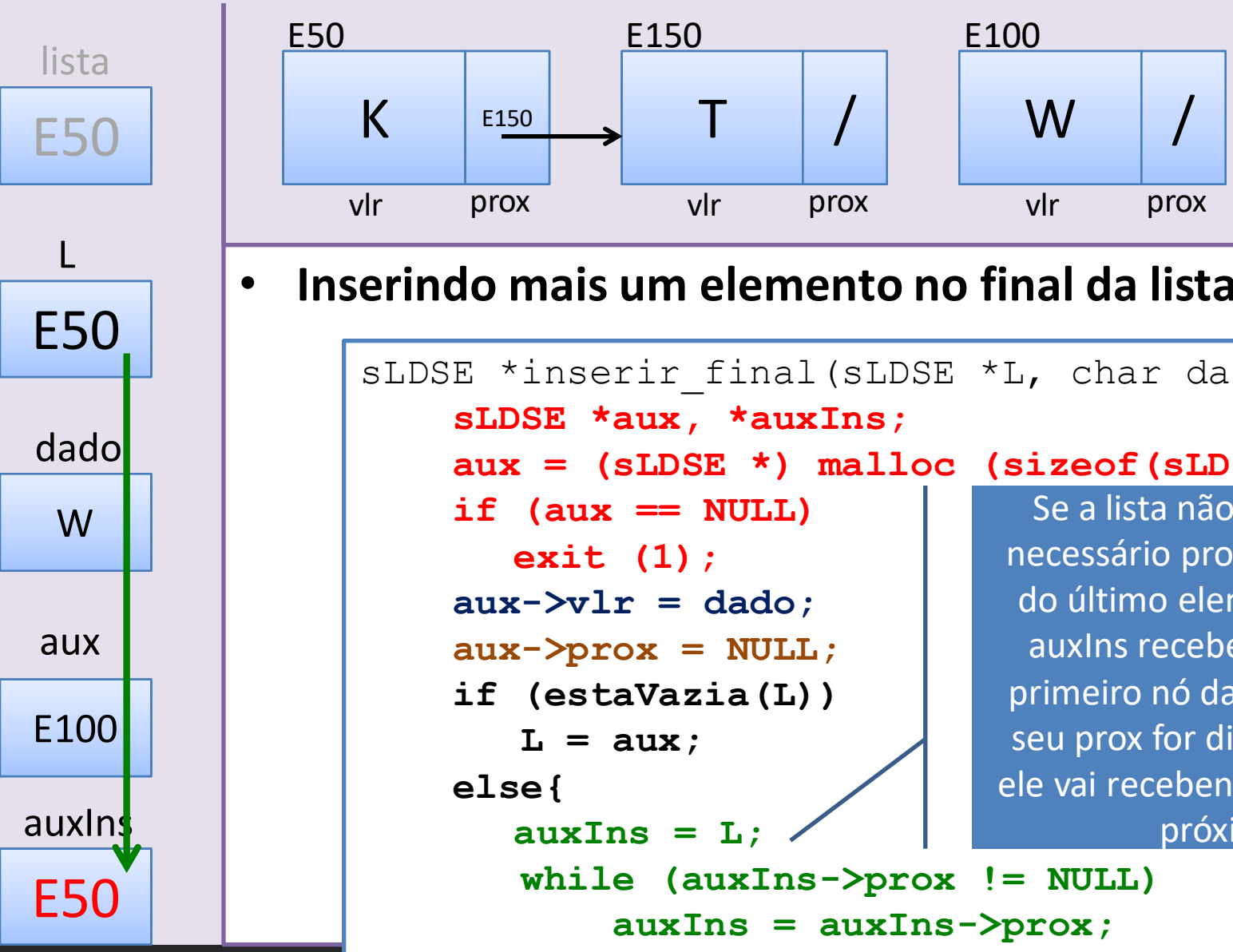
- **Inserindo mais um elemento no final da lista**

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

Se a lista estiver vazia, este novo elemento, apontado por aux, passa a ser o primeiro e único da lista, logo, L recebe aux. Não é nosso caso no momento.

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica



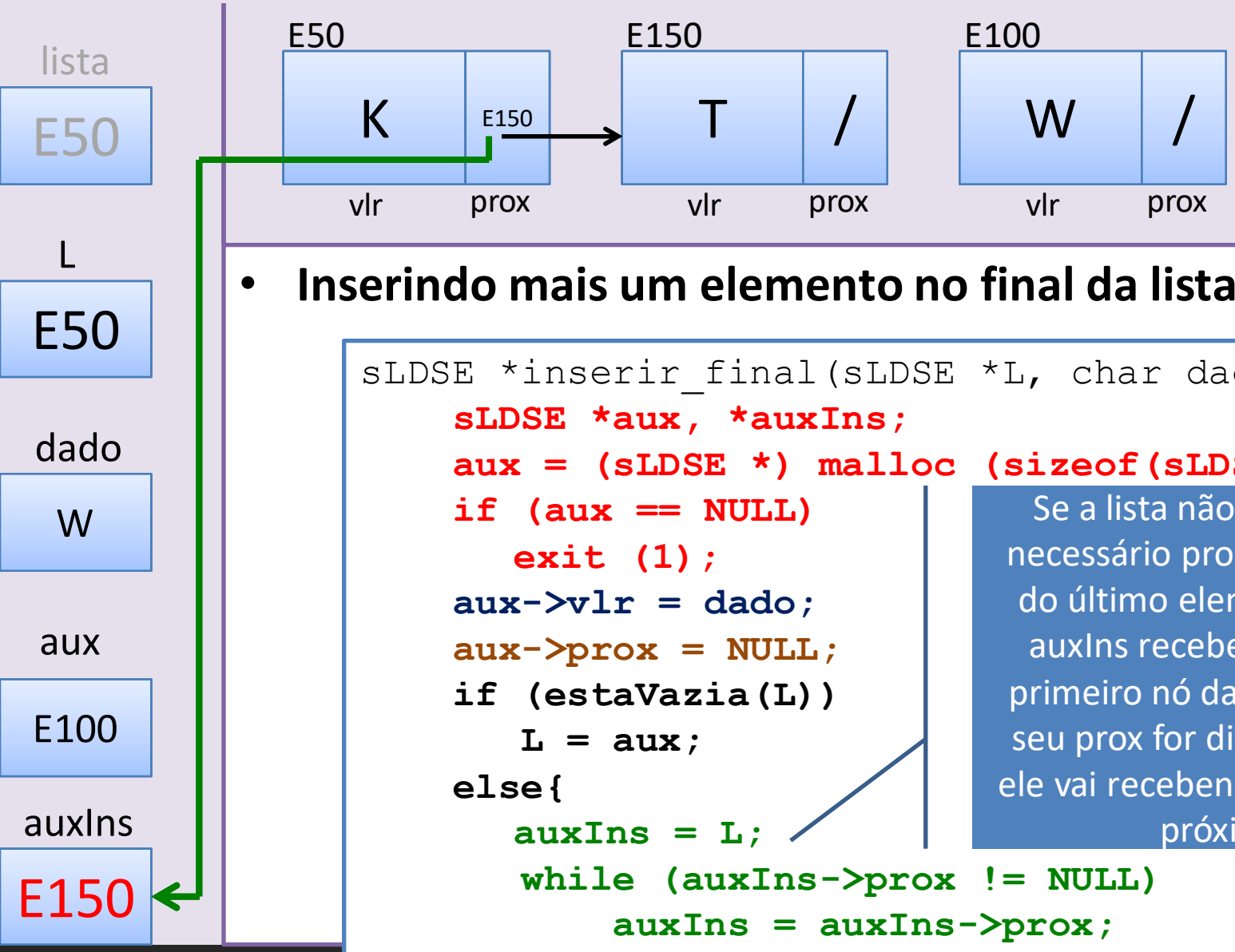
- **Inserindo mais um elemento no final da lista**

```
sLDSE *inserir_final(sLDSE *L, char dado){
     sLDSE *aux, *auxIns;
     aux = (sLDSE *) malloc (sizeof(sLDSE));
     if (aux == NULL)
        exit (1);
     aux->vlr = dado;
     aux->prox = NULL;
     if (estaVazia(L))
        L = aux;
     else{
        auxIns = L;
        while (auxIns->prox != NULL)
            auxIns = auxIns->prox;
```

Se a lista não estiver vazia, é necessário procurar o endereço do último elemento. Para isto, auxIns recebe o endereço do primeiro nó da lista e enquanto seu prox for diferente de NULL, ele vai recebendo o endereço do próximo nó.

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica



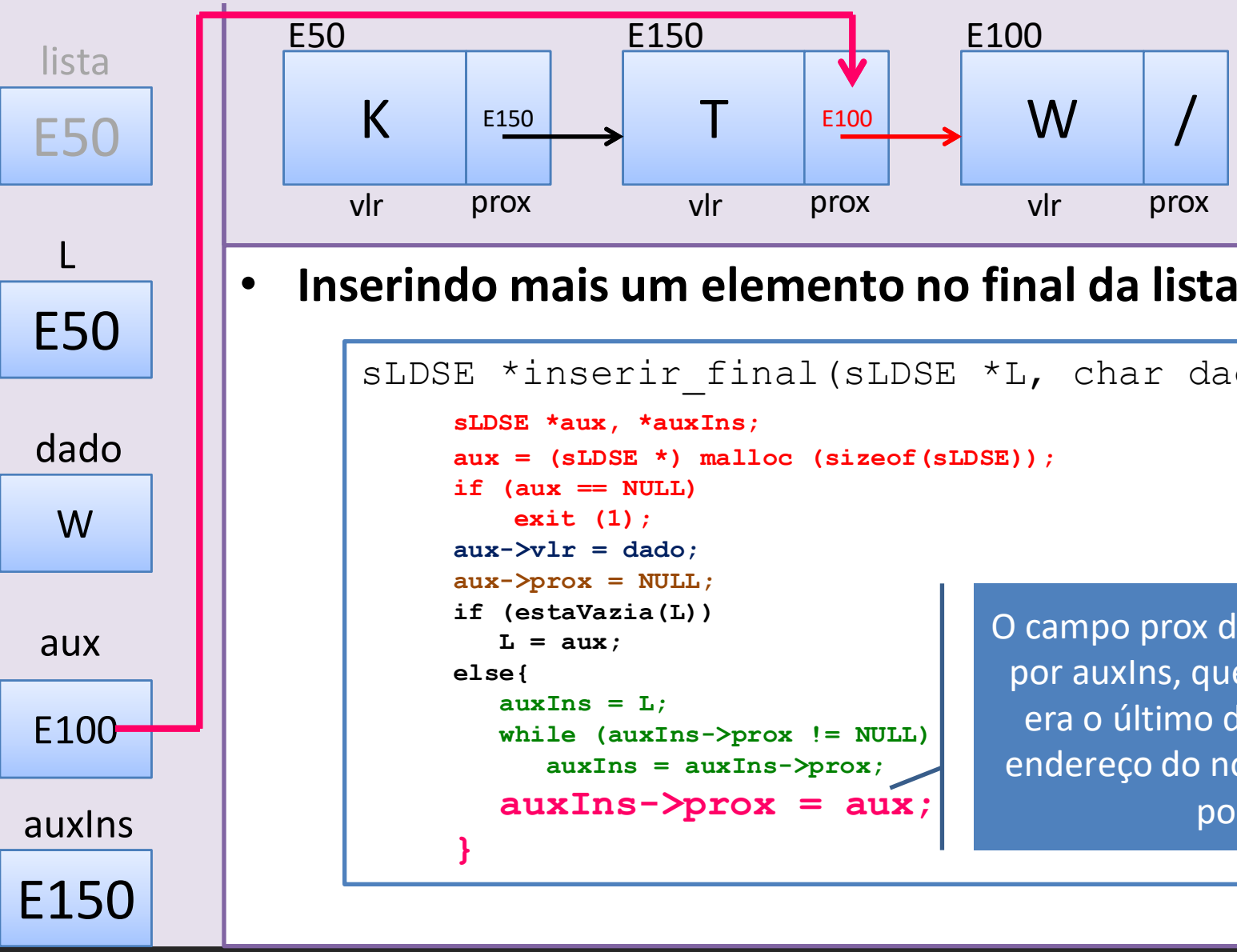
- **Inserindo mais um elemento no final da lista**

```
sLDSE *inserir_final(sLDSE *L, char dado){
     sLDSE *aux, *auxIns;
     aux = (sLDSE *) malloc (sizeof(sLDSE));
     if (aux == NULL)
        exit (1);
     aux->vlr = dado;
     aux->prox = NULL;
     if (estaVazia(L))
        L = aux;
     else{
        auxIns = L;
        while (auxIns->prox != NULL)
            auxIns = auxIns->prox;
```

Se a lista não estiver vazia, é necessário procurar o endereço do último elemento. Para isto, auxIns recebe o endereço do primeiro nó da lista e enquanto seu prox for diferente de NULL, ele vai recebendo o endereço do próximo nó.

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica



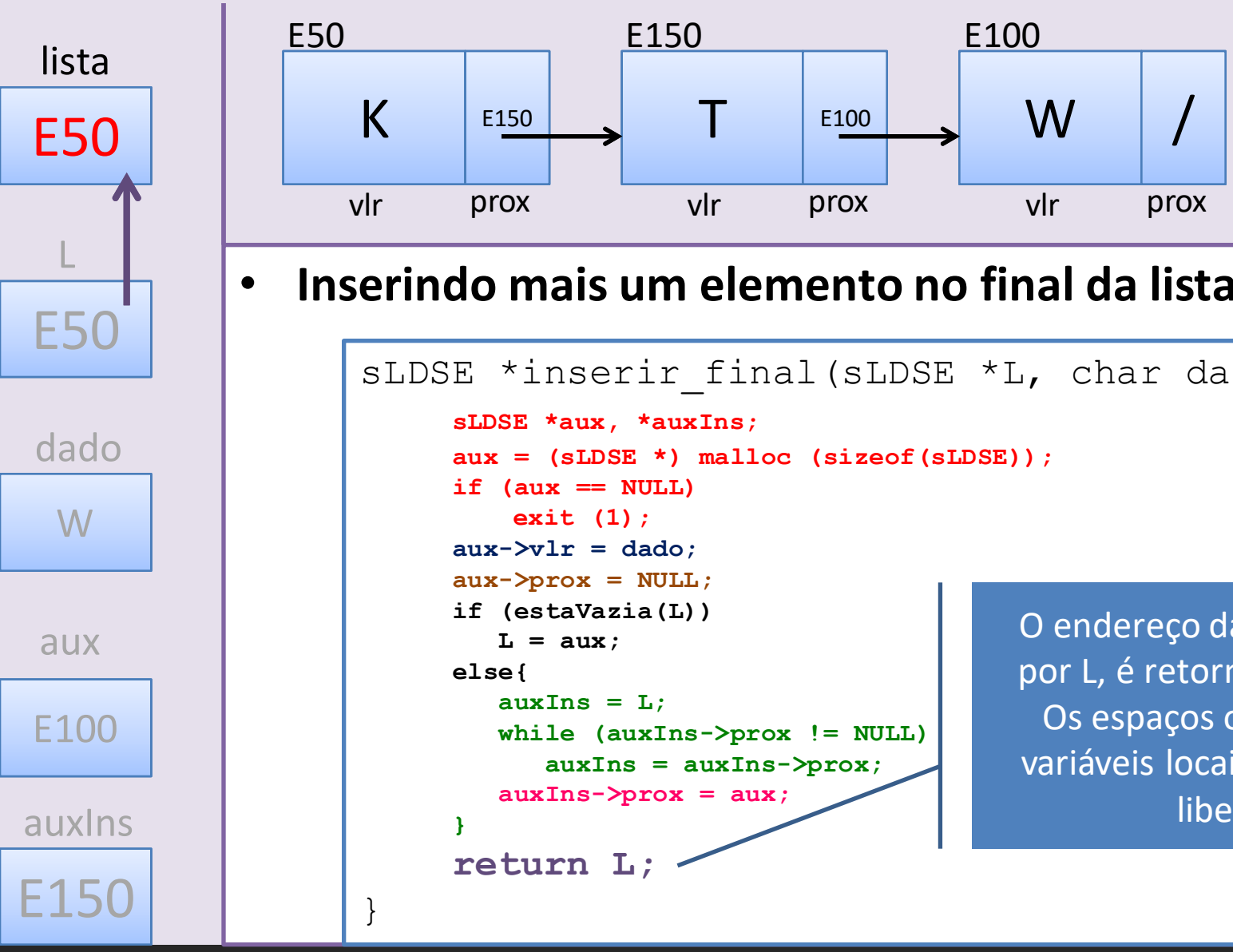
- **Inserindo mais um elemento no final da lista**

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

O campo prox deste nó, apontado por auxIns, que até o momento era o último da lista, recebe o endereço do novo nó, apontado por aux

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica



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

```
sLDSE *inserir_final(sLDSE *L, char dado){
        sLDSE *aux, *auxIns;
        aux = (sLDSE *) malloc (sizeof(sLDSE));
        if (aux == NULL)
            exit (1);
        aux->vlr = dado;
        aux->prox = NULL;
        if (estaVazia(L))
           L = aux;
        else{
           auxIns = L;
           while (auxIns->prox != NULL)
              auxIns = auxIns->prox;
           auxIns->prox = aux;
        }
        return L;
}
```

O endereço da lista, apontado por L, é retornado para "lista". Os espaços ocupados pelas variáveis locais da função são liberados.

Implementação – LDSE – Inserir Final da Lista

Visualização Gráfica

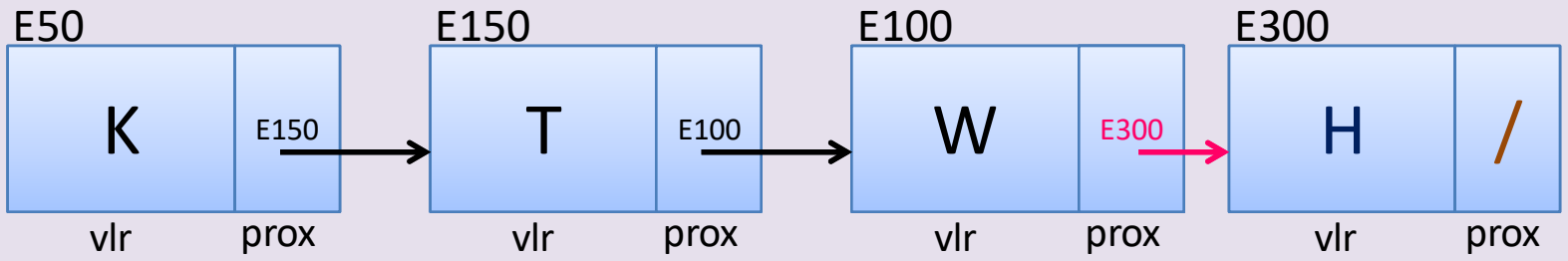
lista
E50

L
E50

dado
H

aux
E300

auxIns
E100



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

```
sLDSE *inserir_final(sLDSE *L, char dado){
      sLDSE *aux, *auxIns;
      aux = (sLDSE *) malloc (sizeof(sLDSE));
      if (aux == NULL)
          exit (1);
      aux->vlr = dado;
      aux->prox = NULL;
      if (estaVazia(L))
         L = aux;
      else{
         auxIns = L;
         while (auxIns->prox != NULL)
            auxIns = auxIns->prox;
         auxIns->prox = aux;
      }
      return L;
}
```

Implementação – LDSE – Listar

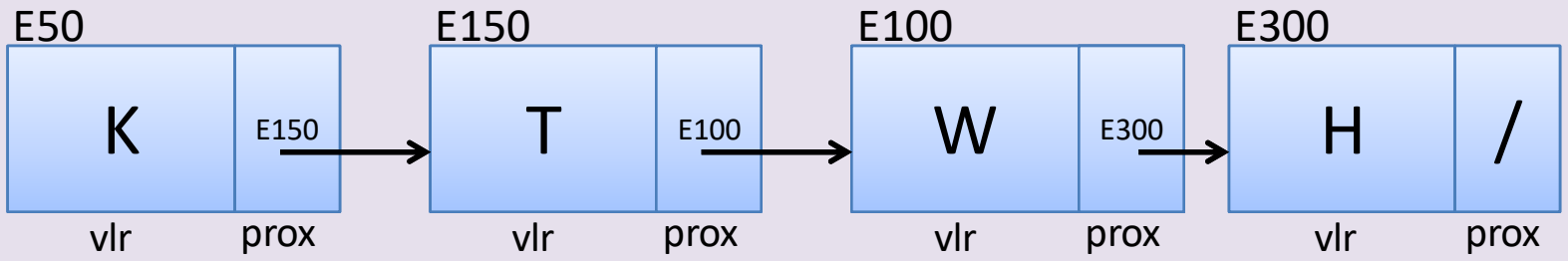
Visualização Gráfica

lista

E50

L

E50



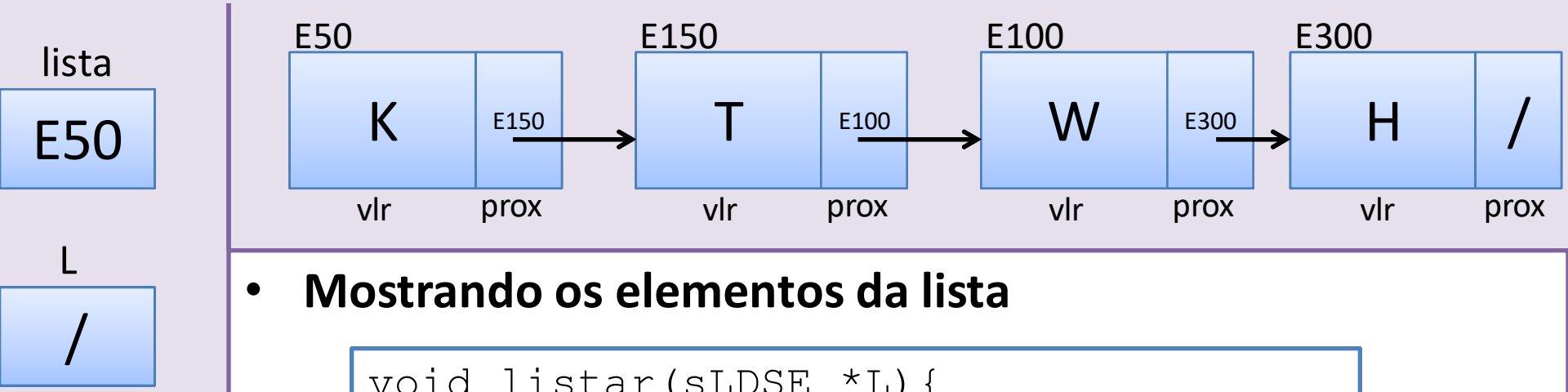
- **Mostrando os elementos da lista**

```
void listar(sLDSE *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 – LDSE – Listar

Visualização Gráfica



- **Mostrando os elementos da lista**

```
void listar(sLDSE *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 – LDSE – Pesquisar

Visualização Gráfica

lista

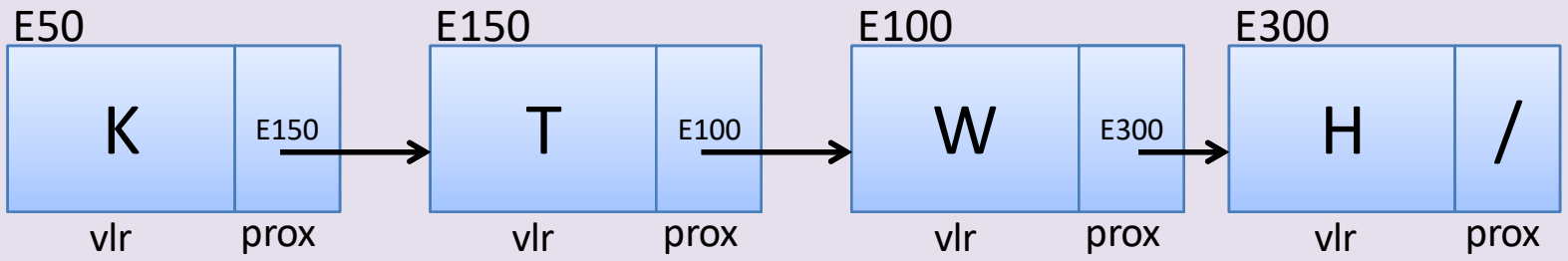
E50

L

E100

dadoPesq

W



- **Procurando um elemento na lista**

```
sLDSE* pesq(sLDSE *L, char dadoPesq, sLDSE **ant){
    *ant = NULL;
    while (L != NULL && L->vlr != dadoPesq){
        *ant = L;
        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 – LDSE – Excluir

Visualização Gráfica

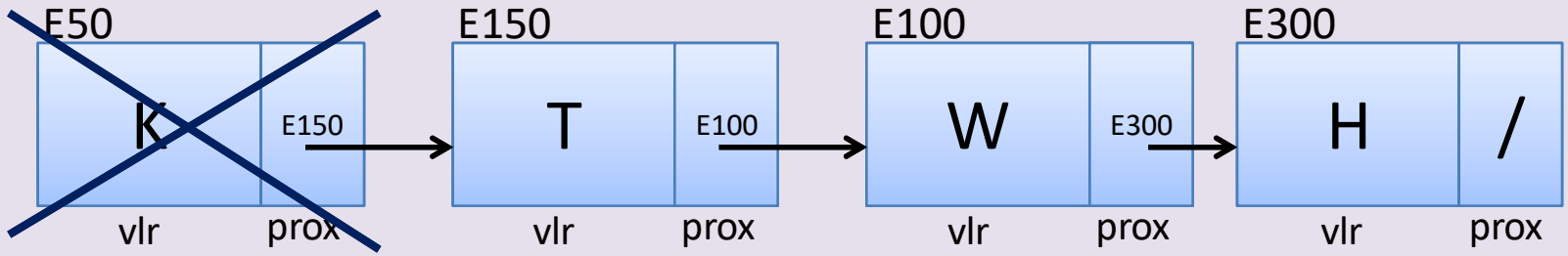
lista
E150

L
E150

dadoExc
K

aux
E50

ant



- **Excluindo o primeiro elemento da lista**

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

Implementação – LDSE – Excluir

Visualização Gráfica

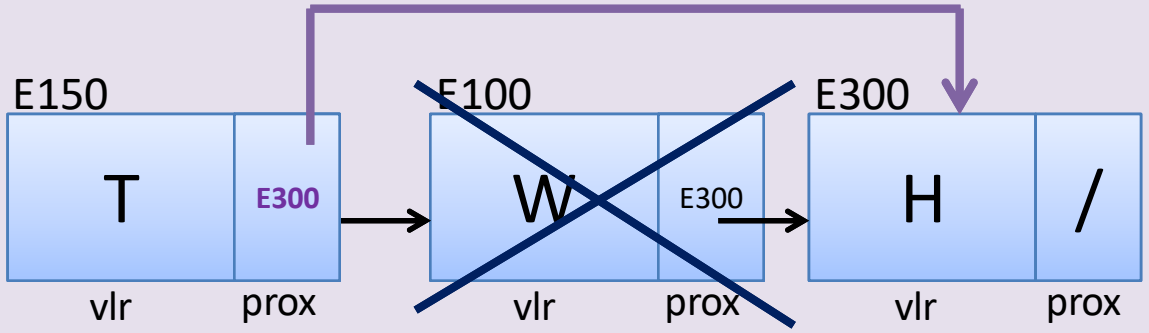
lista
E150

L
E150

dadoExc
W

aux
E100

ant
E150



- **Excluindo um elemento do centro da lista**

```
sLDSE* excluir(sLDSE *L, char dadoExc){
      ...
         aux = pesq(L,dadoExc,&ant);
         ...
               if (aux == L)
                    L = L->prox;
               else
                    ant->prox = aux->prox;
               free(aux);
         ...
      return L;
}
```

Implementação – LDSE – Excluir

- **A função completa ficará:**

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

- **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

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

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

- TENENBAUM A., LANGSAM Y. e AUGENSTEIN M. J. Estrutura de Dados usando C. Editora Makron, 1995.
  - Pág 223, seção 4.2 - até pág. 229
  - Pág 231 (“Operações getnode e freenode”) – até pág. 233
  - Pág 256 (“Listas ligadas usando variáveis dinâmicas) – até pág. 258
  - Pág 260 (“Exemplos de operações de listas em C”) – até pág. 262

Exercício

- Faça uma função que receba uma lista dinâmica simplesmente 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.