Estrutura de Dados

Aula 9

Listas Simplesmente Encadeadas

Alocação Dinâmica de Memória

Projeto **ListaSE**

ESTRUTURAS DE DADOS

Implementação:

- Representação por contiguidade física.

  - Os nodos são armazenados em endereços contíguos, ou igualmente distanciados um do outro.

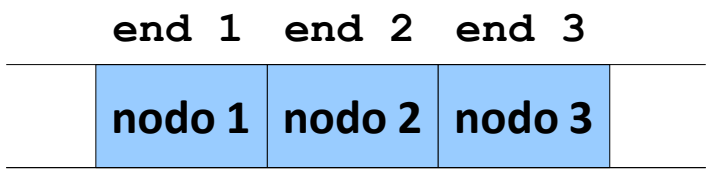
- Representação por encadeamento.

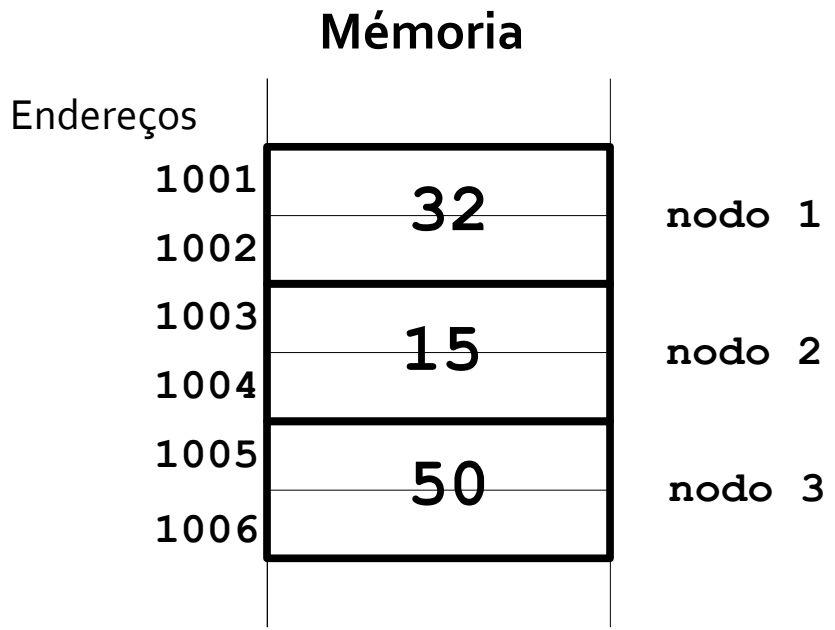
  - Os nodos são armazenados em endereços que não mantém qualquer relação entre si. Os relacionamentos entre os nodos são representados por meio de ligações físicas explícitas.

Lista Linear

Representação por contiguidade física

- Os nodos são armazenados em endereços contíguos, ou igualmente distanciados um do outro.
- Os relacionamentos são representados pela disposição física dos componentes na memória.
- A posição na estrutura lógica determina a posição na estrutura física.

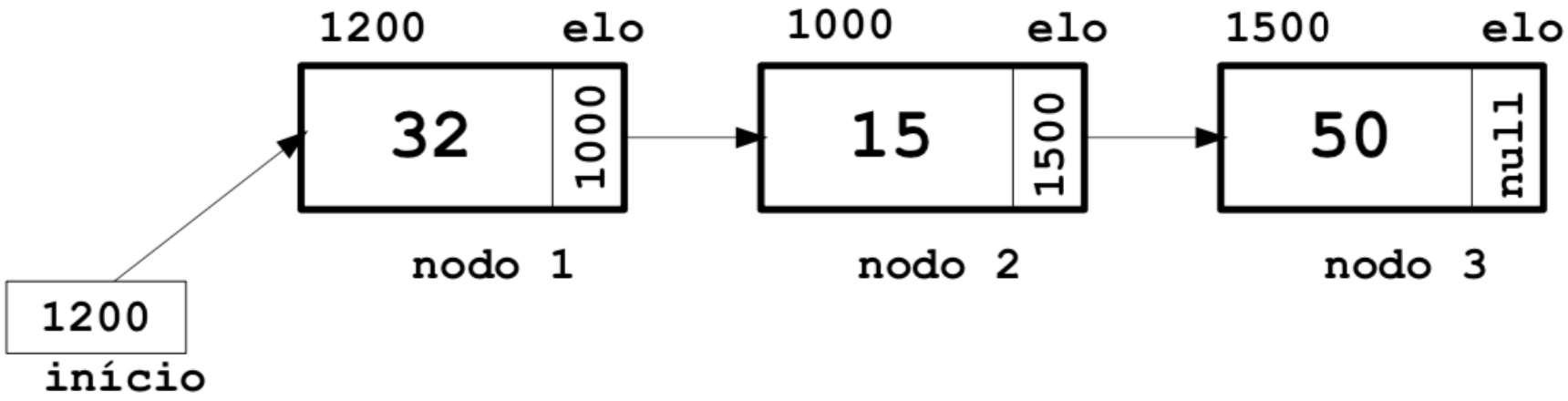




Lista Linear

Representação por encadeamento

- A disposição física dos nodos independe de sua posição na estrutura lógica.
- Os relacionamentos são representados por elos que são ligações físicas explícitas.
- O valor contido em um campo de elo é o endereço de outro nodo.



Alocação dinâmica

`malloc` – Reserva um espaço de memória com uma quantidade de bytes passada como argumento. Retorna **NULL** caso o bloco não possa ser alocado.

`free` – Desaloca o bloco de memória cujo o endereço é passado como argumento.

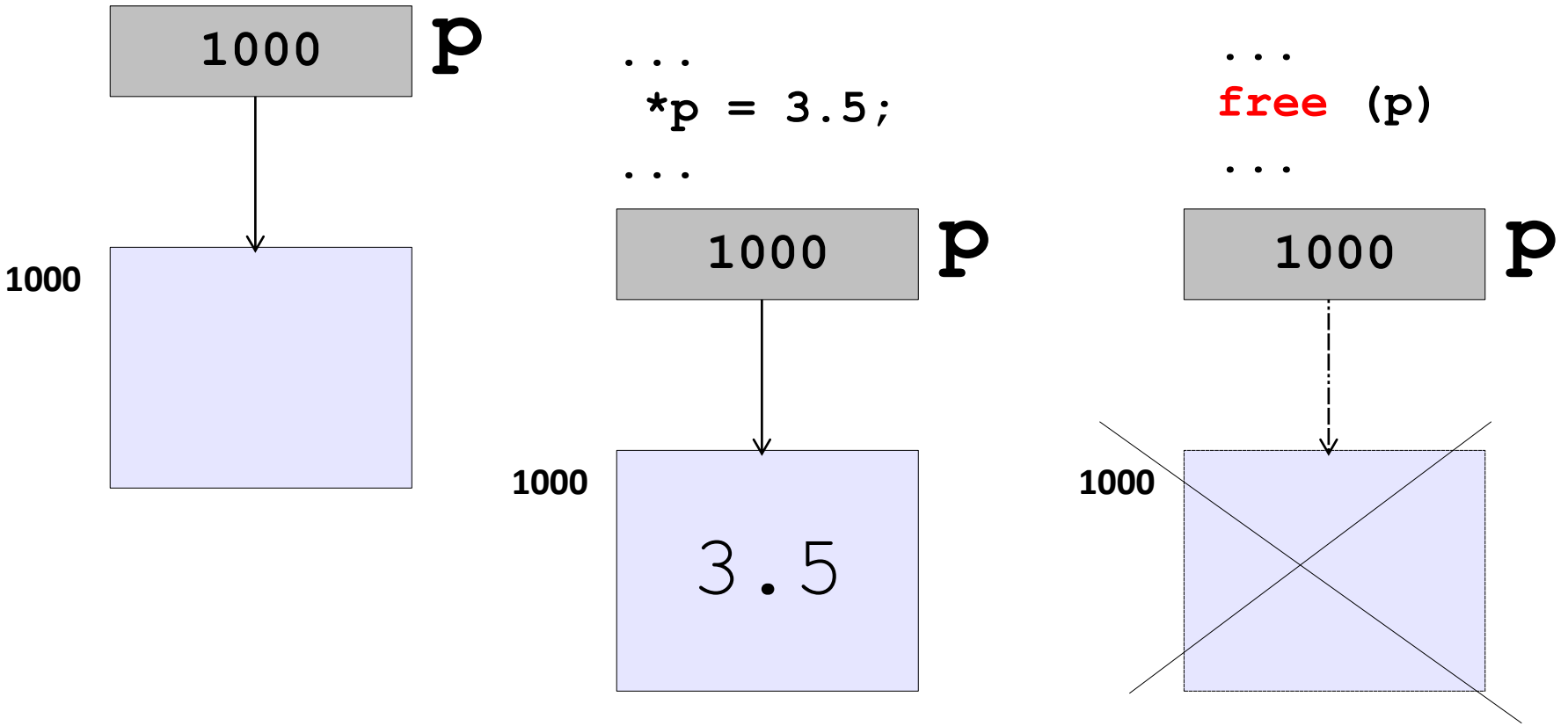
Alocação dinâmica

Permite criar uma variável durante a execução do programa.

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

int main() {
  float *p;

  p = (float *) malloc( sizeof(float));
  if (p==NULL)
     printf("Memória insuficiente\n");
  else {
     *p = 3.5;
     printf("Valor: %f\n",*p);
     free(p);
  }

  return 0;
}
```

Alocação dinâmica

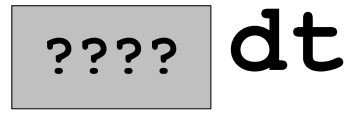
```
...
p =(float *) malloc(sizeof(float));
...
```

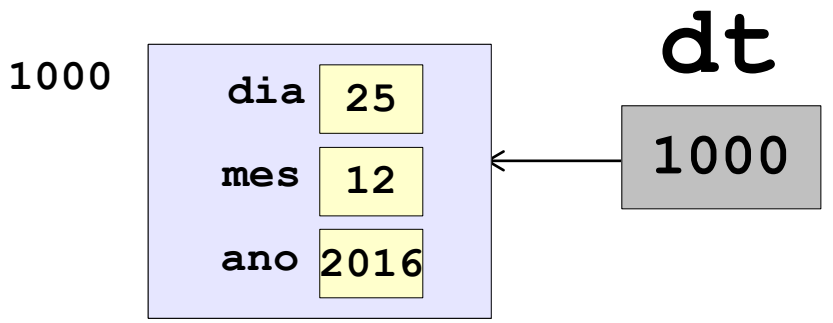


A área de memória é alocada com a função **malloc** e desalocada pela função **free**.

Alocando uma struct dinamicamente

```
//...

int main() {
  Data *dt;

  dt = (Data *) malloc ( sizeof(Data));
  if (dt == NULL)
     printf("Memória insuficiente\n");
  else {
     dt->dia = 25;
     dt->mes = 12;
     dt->ano = 2016;
     printf("%d/%d/%d\n",dt->dia, dt->mes, dt->ano);
     free (dt);
  }

  return 0;
}
```

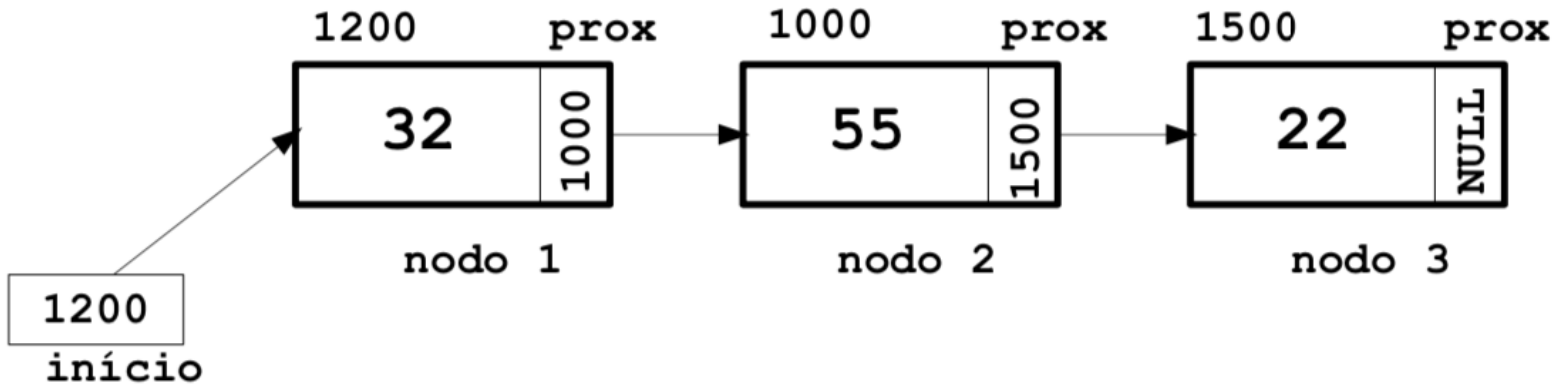




Lista Simplesmente Encadeada

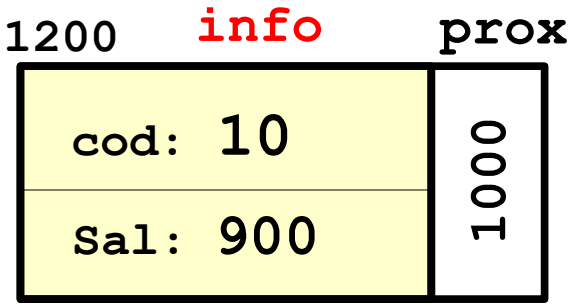
Representação por encadeamento

- A disposição física dos nodos independe de sua posição na estrutura lógica.
- Os relacionamentos são representados por elos que são ligações físicas explícitas.
- O valor contido em um campo **prox** é o endereço do próximo nodo.
- O valor contido em **inicio** representa o endereço do primeiro nodo da lista (NULL se a lista está vazia).



Lista Simplesmente Encadeada

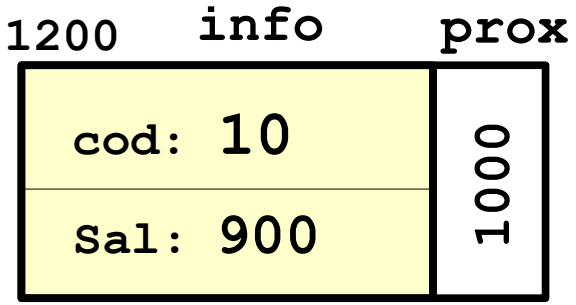
Campos de um **Nodo**



`info` : contém a informação armazenada.
(Ex: dados de um funcionário)

`prox`: contém o **endereço** do **próximo** nodo da lista.
(**`NULL`** se não existe próximo)

Lista Simplesmente Encadeada




```
typedef struct {
  int cod;
  float sal;
} Dado;

typedef struct nodo Nodo;

struct nodo {
  Dado info;     /* Informação armazenada */
  Nodo *prox;    /* Endereço do próximo   */
};
```

Lista Simplesmente Encadeada

```
typedef struct {
   Nodo *inicio;
} ListaSE;
```

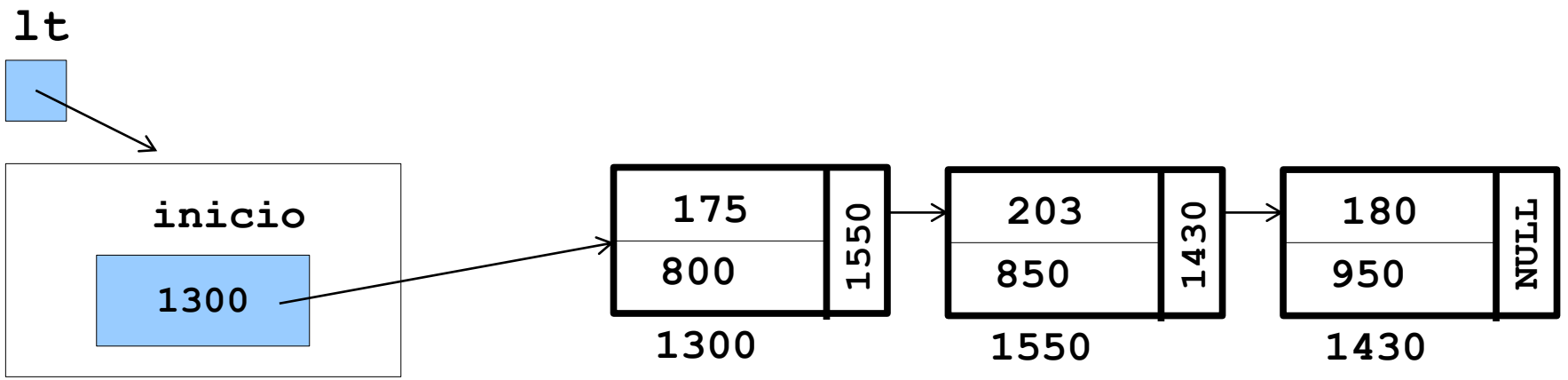
```
   ...
int main() {
   ListaSE lista;

   criaLista(&lista);
   ...
}
```

```
void criaLista(ListaSE *lt) {
   lt->inicio = NULL;
}
```

Lista vazia

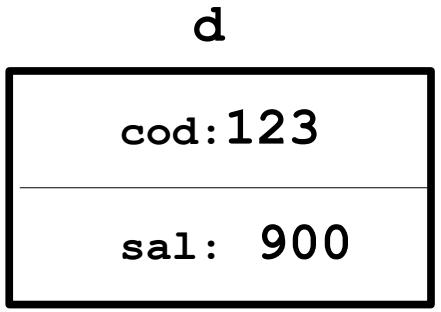
ListaSE - incluiNoInicio

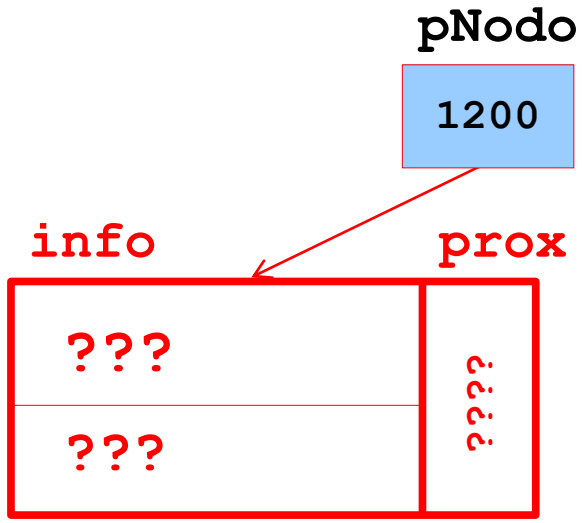
```
int  incluiNoInicio(ListaSE *lt, Dado d) {
   Nodo *pNodo;

   ...
}
```

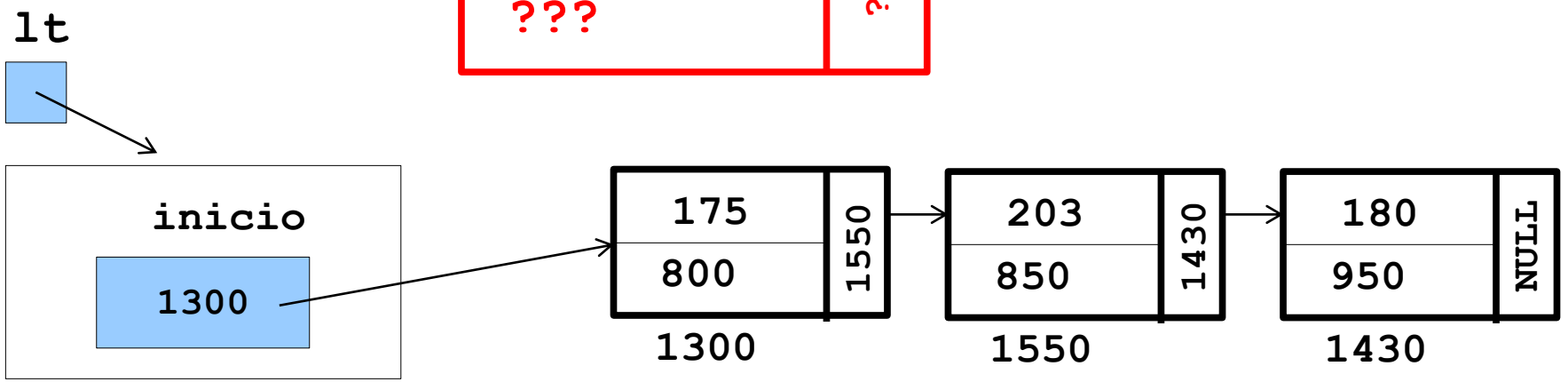


ListaSE - incluiNoInicio

```
int  incluiNoInicio(ListaSE *lt, Dado d) {
  Nodo *pNodo;

  pNodo = (Nodo *) malloc (sizeof (Nodo));

  ...
}
```

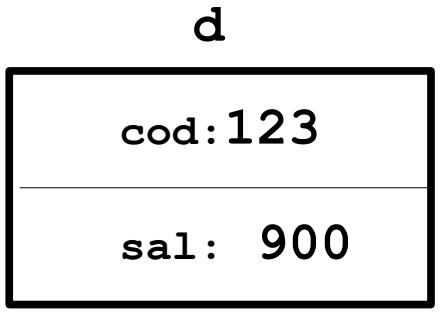


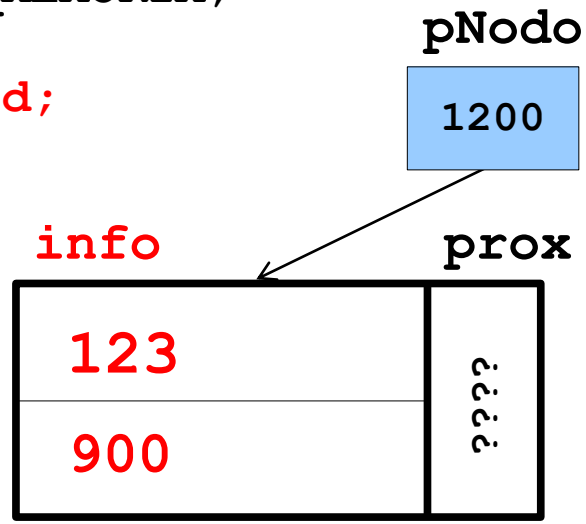


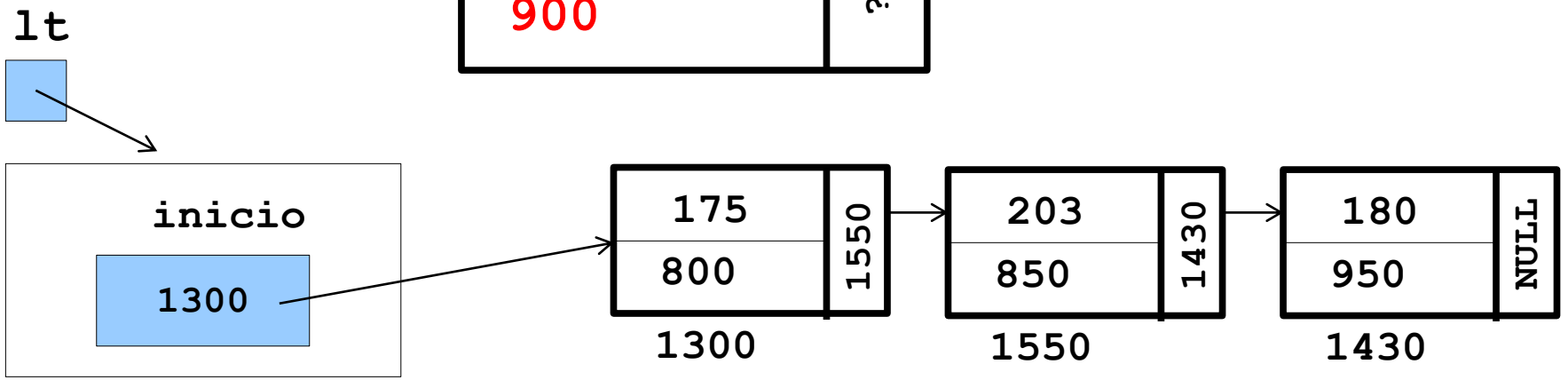


ListaSE - incluiNoInicio

```
int  incluiNoInicio(ListaSE *lt, Dado d) {
  Nodo *pNodo;

  pNodo = (Nodo *) malloc (sizeof (Nodo));
  if (pNodo == NULL)
     return FALTOU_MEMORIA;
  else {
     pNodo->info = d;
     ...

  }
}
```

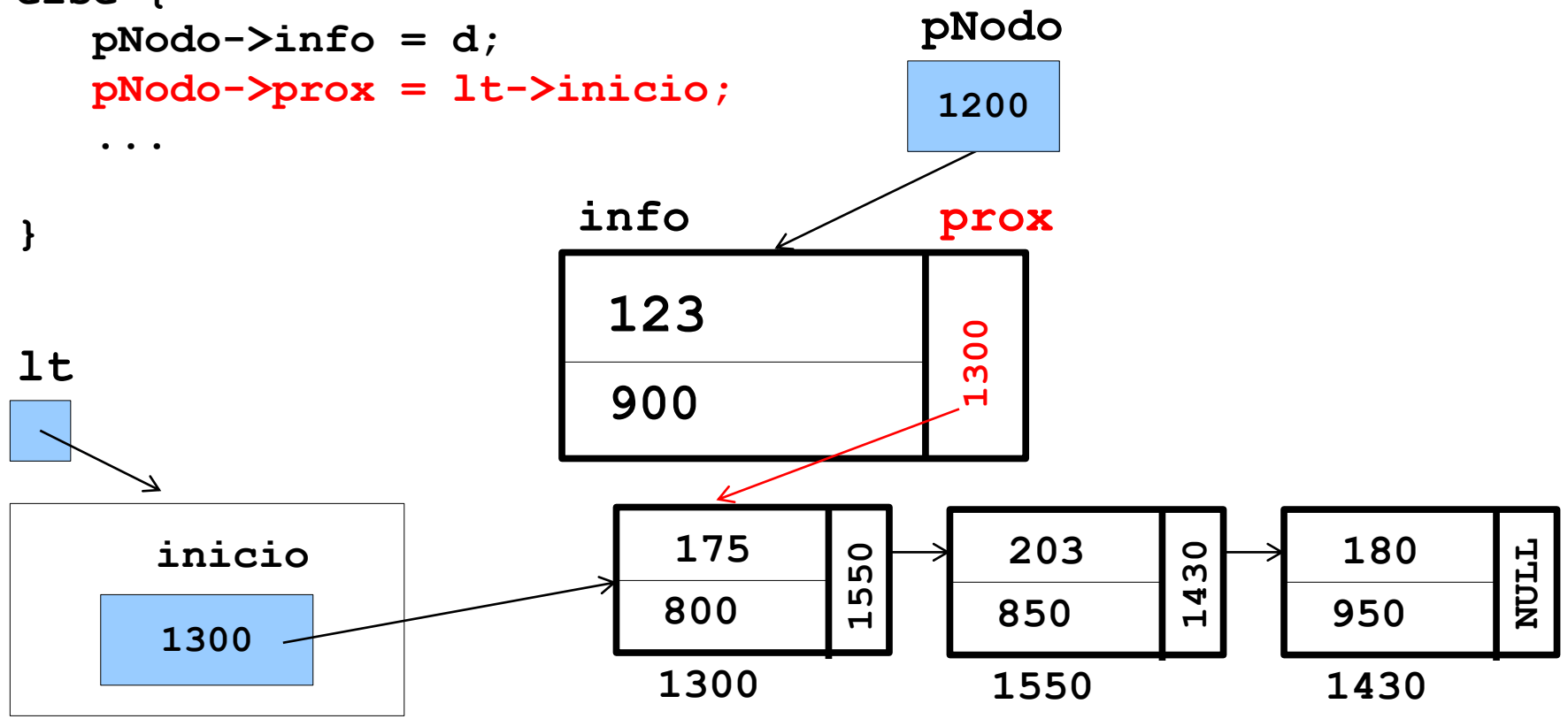






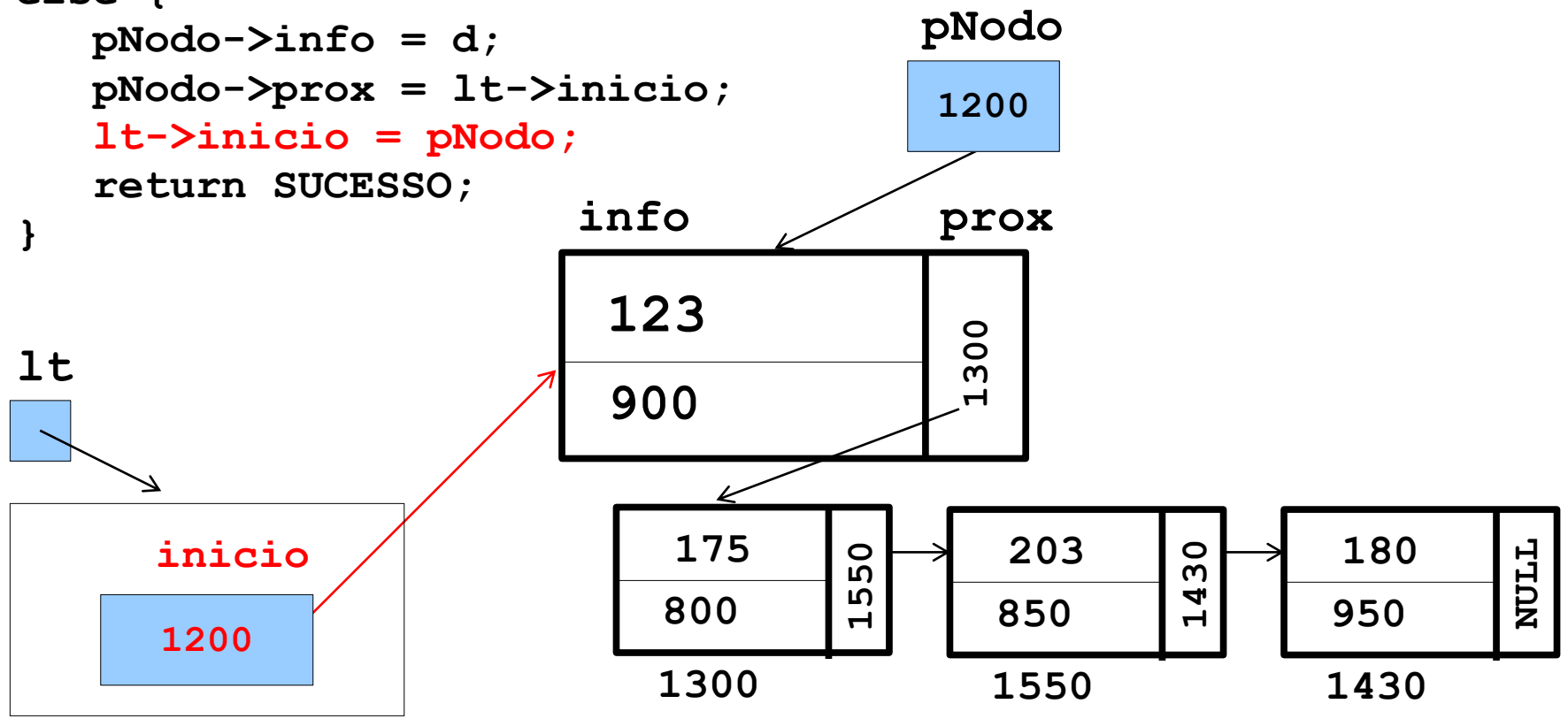
ListaSE - incluiNoInicio

```
int  incluiNoInicio(ListaSE *lt, Dado d) {
  Nodo *pNodo;

  pNodo = (Nodo *) malloc (sizeof (Nodo));
  if (pNodo == NULL)
     return FALTOU_MEMORIA;
  else {
     pNodo->info = d;
     pNodo->prox = lt->inicio;
     ...

  }
}
```



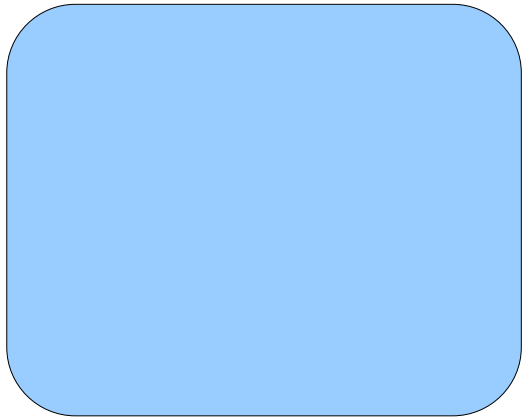
ListaSE - incluiNoInicio

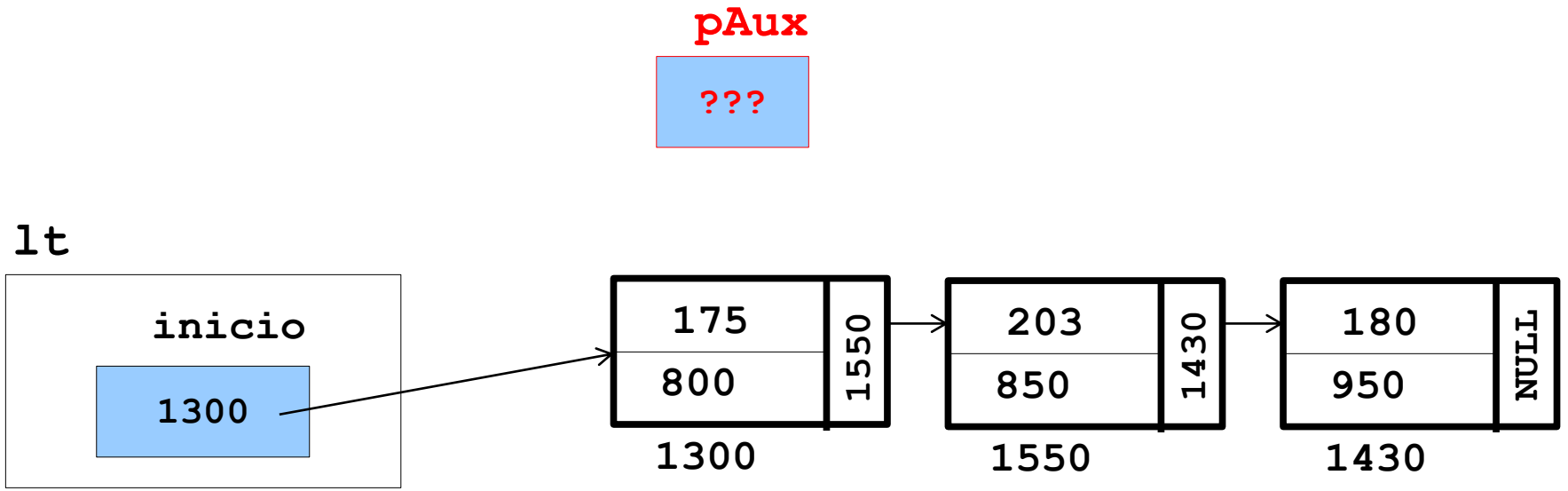
```
int  incluiNoInicio(ListaSE *lt, Dado d) {
  Nodo *pNodo;

  pNodo = (Nodo *) malloc (sizeof (Nodo));
  if (pNodo == NULL)
     return FALTOU_MEMORIA;
  else {
     pNodo->info = d;
     pNodo->prox = lt->inicio;
     lt->inicio = pNodo;
     return SUCESSO;
  }
}
```



ListaSE - exibe

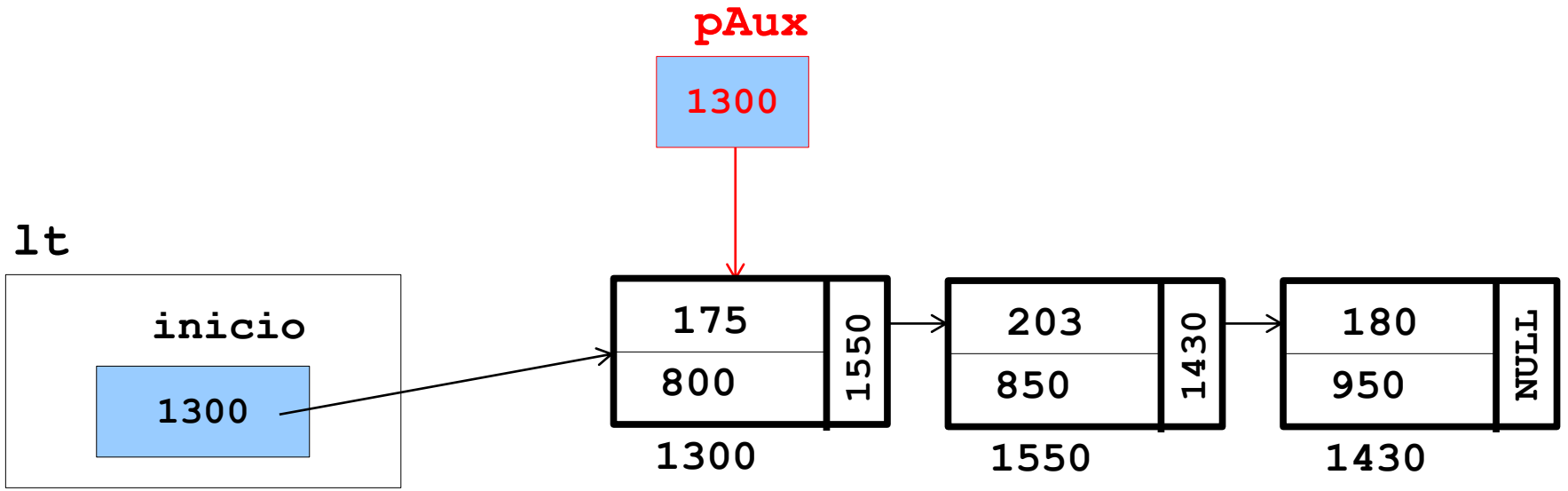
```
void  exibe(ListaSE lt) {
   Nodo *pAux;

   ...
}
```





ListaSE - exibe

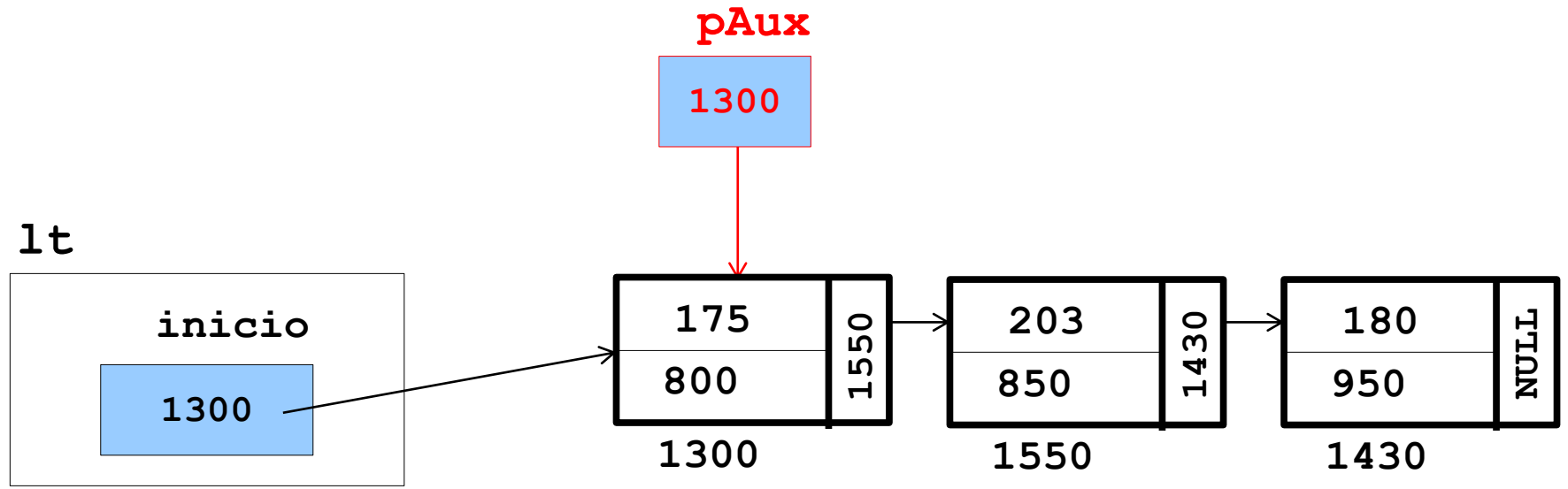
```
void  exibe(ListaSE lt) {
   Nodo *pAux;

   pAux = lt.inicio;
   ...
}
```



ListaSE - exibe

```
void  exibe(ListaSE lt) {
  Nodo *pAux;

  pAux = lt.inicio;
  while (pAux != NULL) {
    printf("%d %f\n",pAux->info.cod,
                     pAux->info.sal);
  ...
  }

}
```

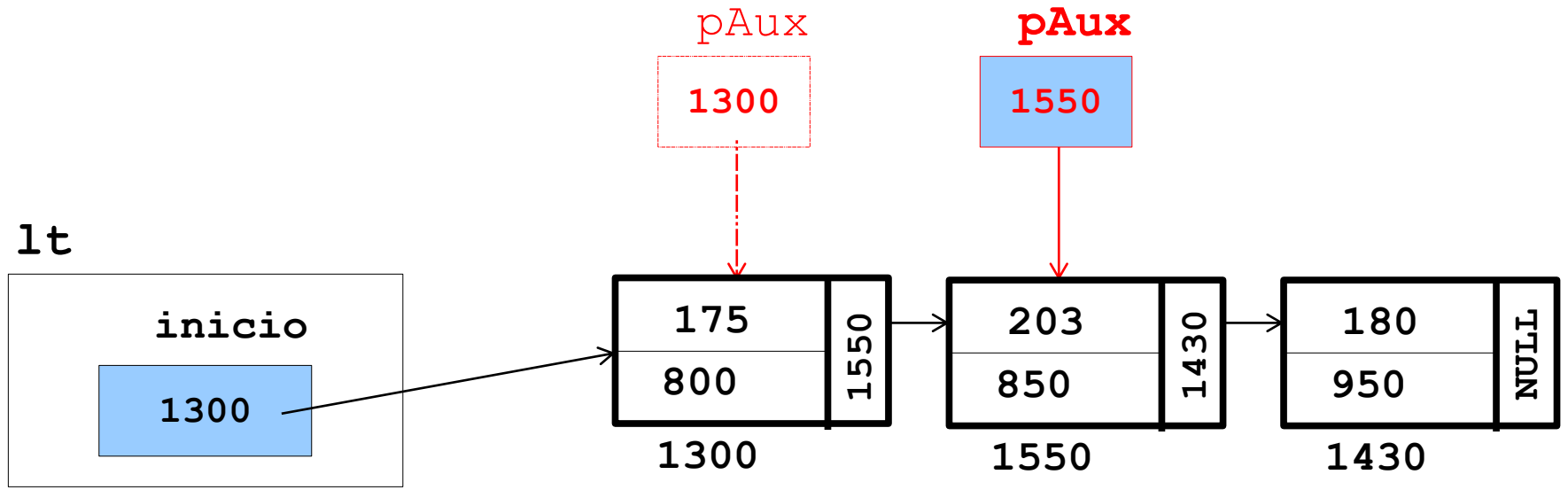
```
175  800
```



ListaSE - exibe

```
void  exibe(ListaSE lt) {
  Nodo *pAux;

  pAux = lt.inicio;
  while (pAux != NULL) {
    printf("%d %f\n",pAux->info.cod,
                     pAux->info.sal);
    pAux = pAux->prox;
  }

}
```

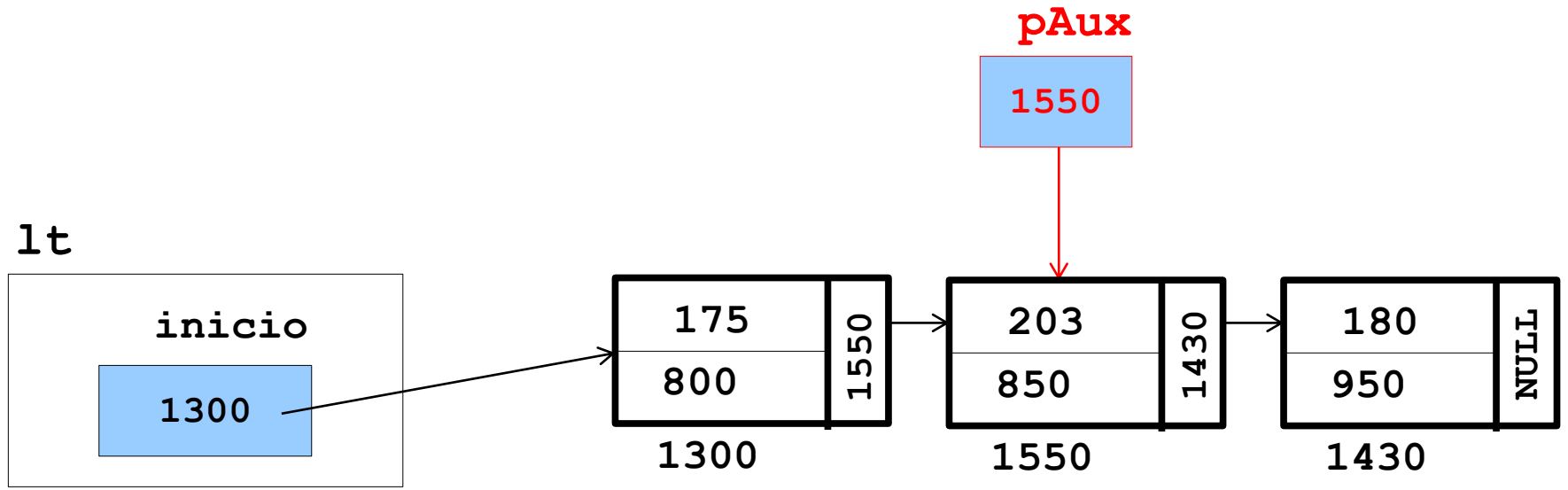
```
175  800
```



ListaSE - exibe

```
void  exibe(ListaSE lt) {
  Nodo *pAux;

  pAux = lt.inicio;
  while (pAux != NULL) {
    printf("%d %f\n",pAux->info.cod,
                     pAux->info.sal);
    pAux = pAux->prox;
  }

}
```

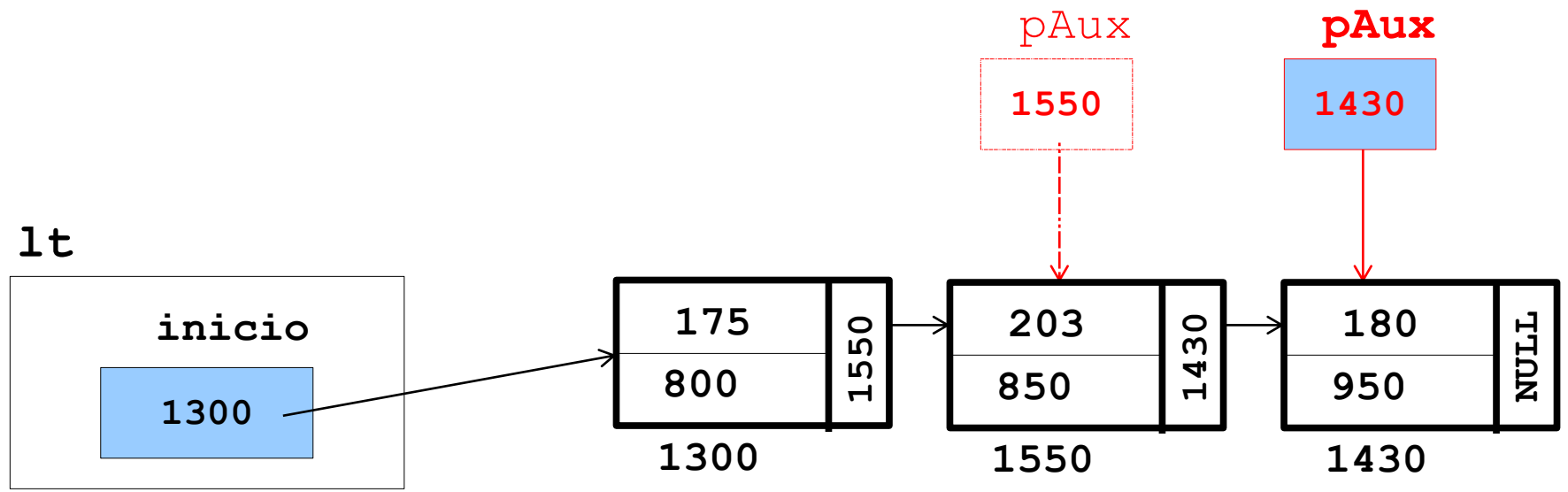
```
175   800
203   850
```



ListaSE - exibe

```
void  exibe(ListaSE lt) {
  Nodo *pAux;

  pAux = lt.inicio;
  while (pAux != NULL) {
    printf("%d %f\n",pAux->info.cod,
                     pAux->info.sal);
    pAux = pAux->prox;
  }

}
```

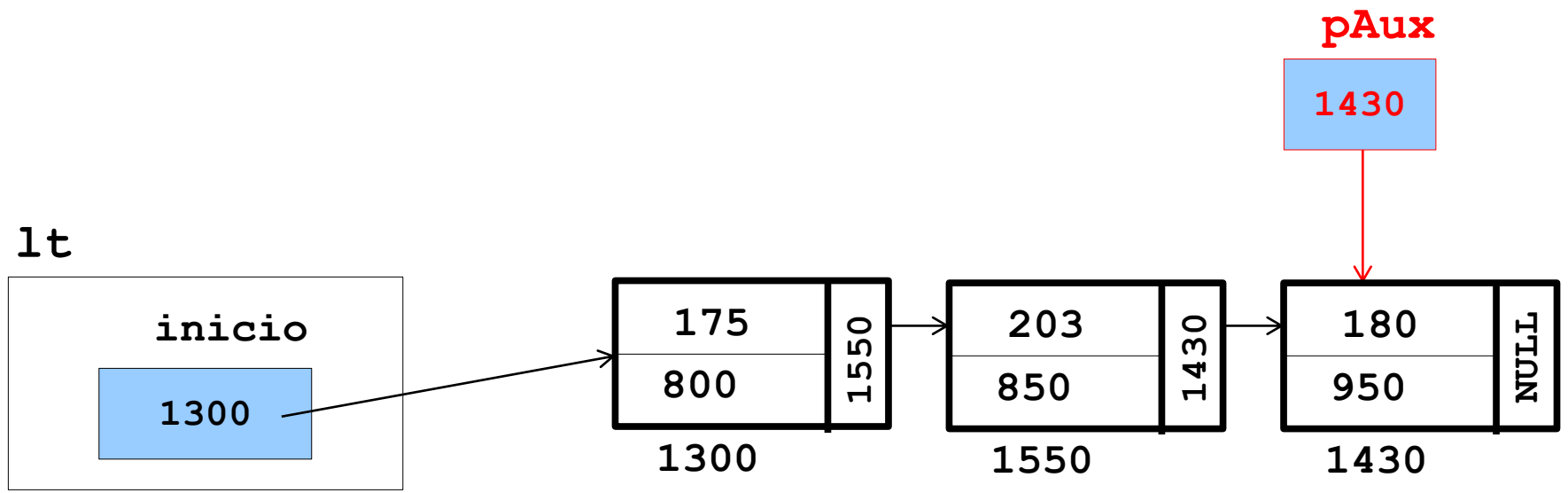
```
175   800
203   850
```



ListaSE - exibe

```
void  exibe(ListaSE lt) {
  Nodo *pAux;

  pAux = lt.inicio;
  while (pAux != NULL) {
    printf("%d %f\n",pAux->info.cod,
                     pAux->info.sal);
    pAux = pAux->prox;
  }

}
```

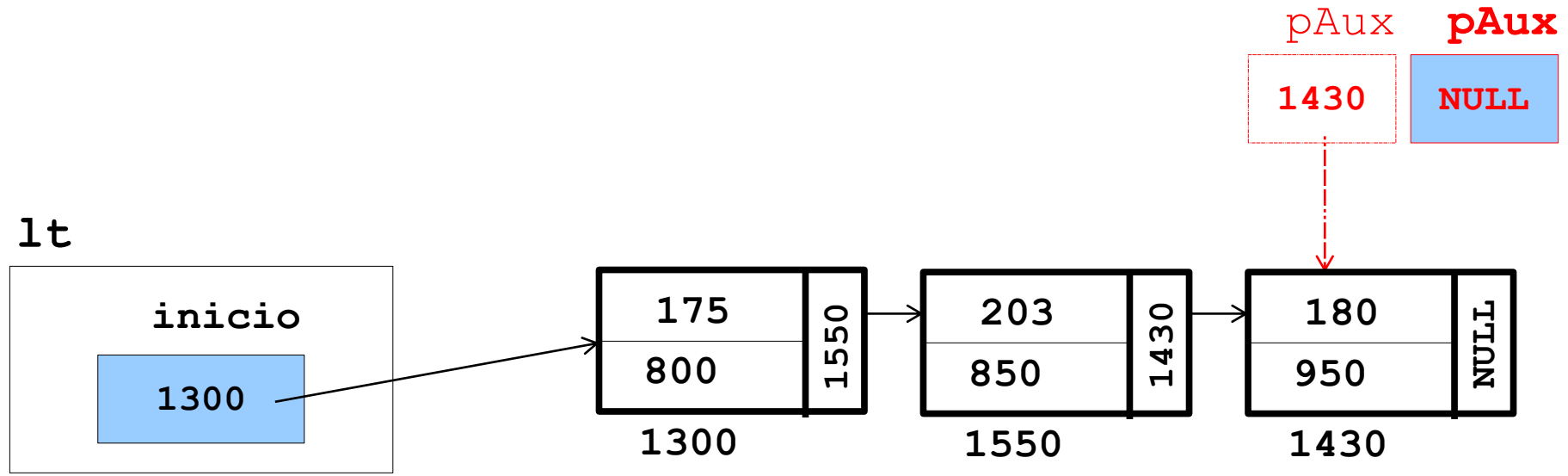
```
175   800
203   850
180   950
```



ListaSE - exibe

```
void  exibe(ListaSE lt) {
  Nodo *pAux;

  pAux = lt.inicio;
  while (pAux != NULL) {
    printf("%d %f\n",pAux->info.cod,
                     pAux->info.sal);
    pAux = pAux->prox;
  }

}
```

```
175   800
203   850
180   950
```



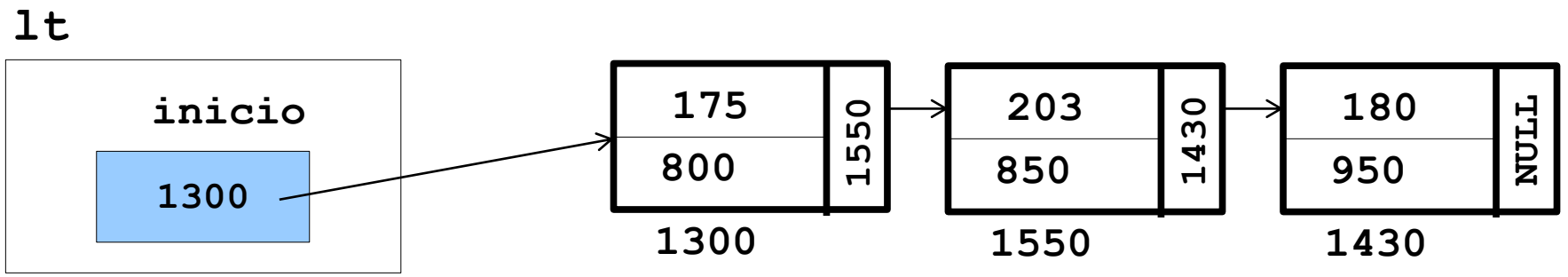
ListaSE - exibe

```
void  exibe(ListaSE lt) {
  Nodo *pAux;

  pAux = lt.inicio;
  while (pAux != NULL) {
    printf("%d %f\n",pAux->info.cod,
                     pAux->info.sal);
    pAux = pAux->prox;
  }

}
```

```
175   800
203   850
180   950
```

pAux

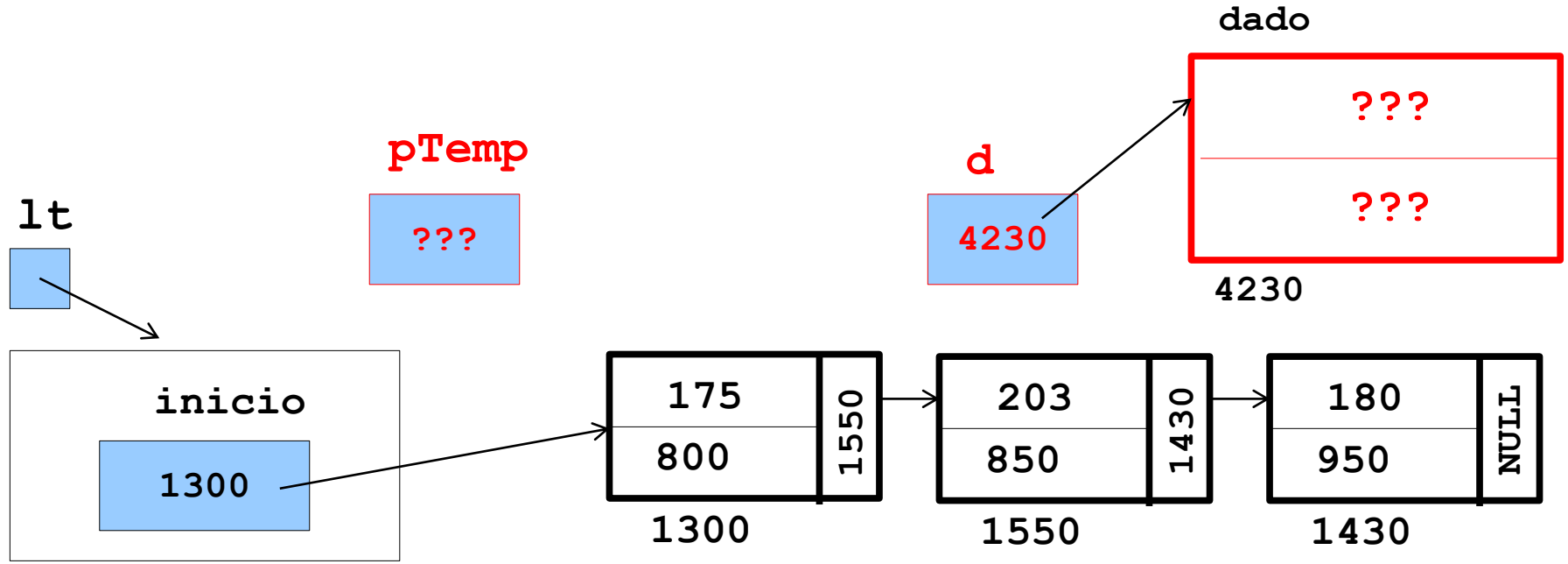
NULL



ListaSE - excluiDoInicio

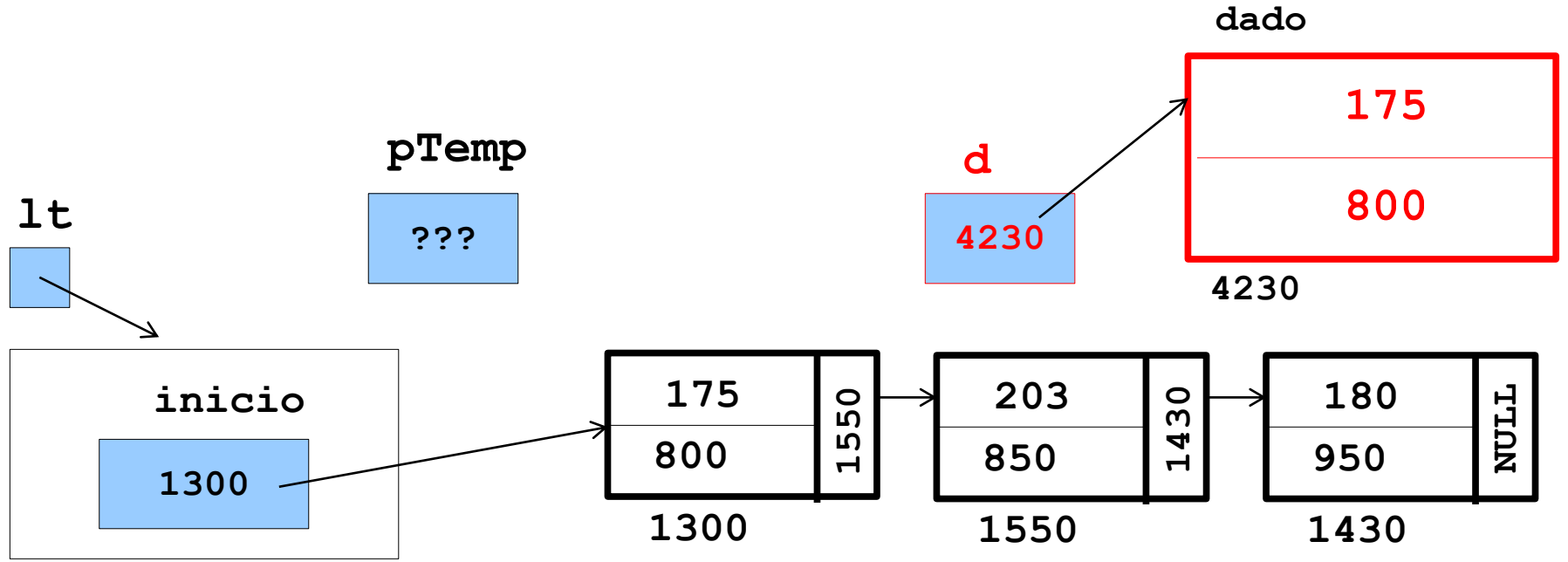
```
int  excluiDoInicio(ListaSE *lt, Dado *d) {
  Nodo *pTemp;
  if (lt->inicio==NULL)
     return LISTA_VAZIA;
  else
     ...

}
```

```
  ...
int main() {
  ListaSE lista;
  Dado dado;
  ...
  excluiDoInicio(&lista,&dado);
  ...
}
```



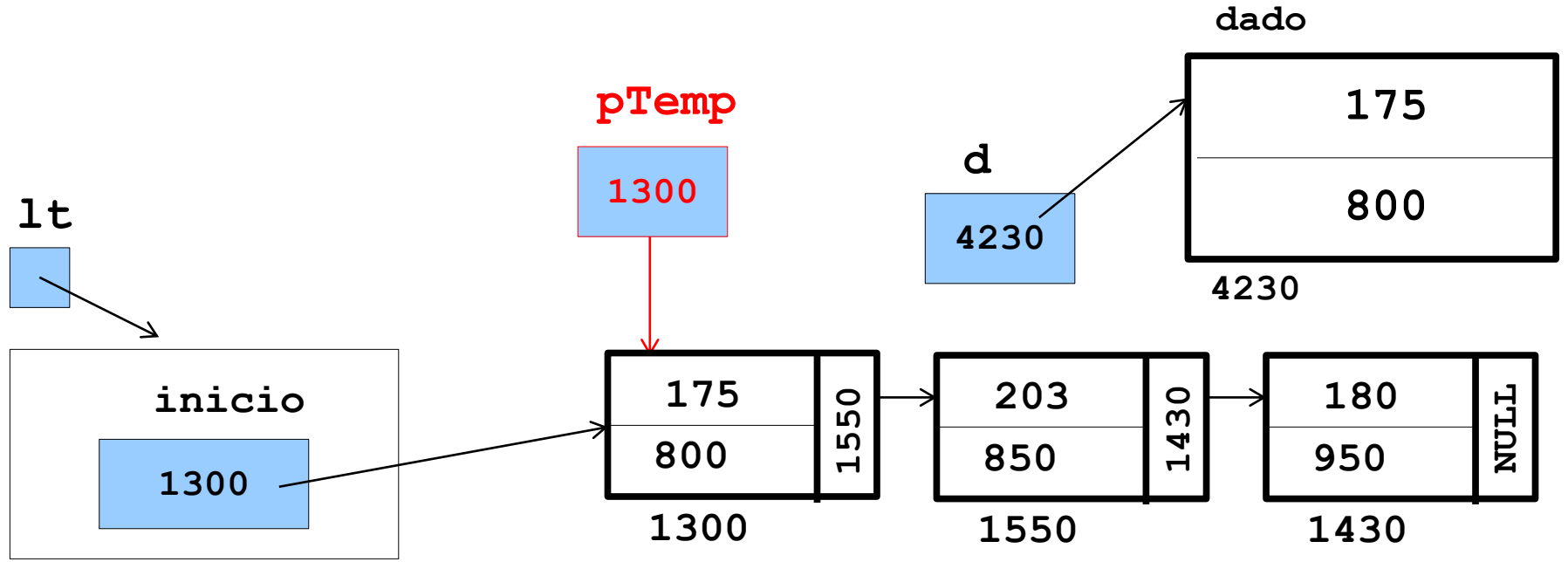
ListaSE - excluiDoInicio

```
int  excluiDoInicio(ListaSE *lt, Dado *d) {
  Nodo *pTemp;
  if (lt->inicio==NULL)
     return LISTA_VAZIA;
  else {
     *d = lt->inicio->info;
     ...
  }

}
```



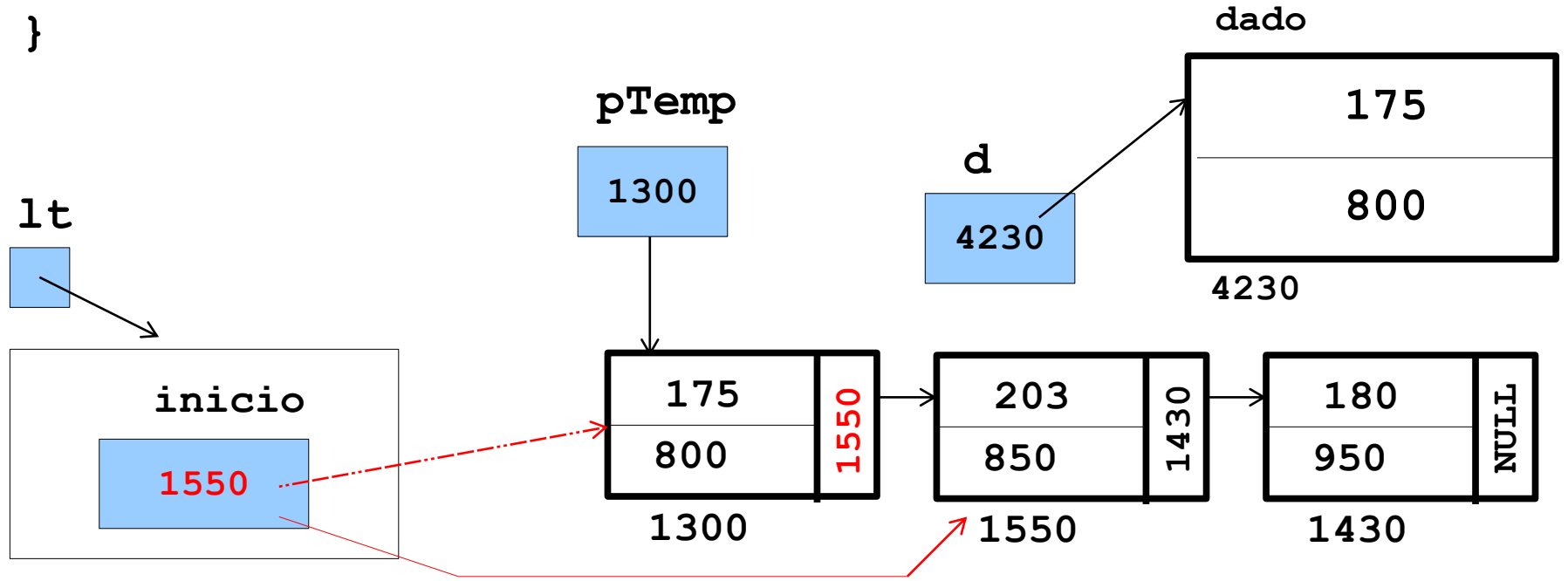
ListaSE - excluiDoInicio

```
int  excluiDoInicio(ListaSE *lt, Dado *d) {
  Nodo *pTemp;
  if (lt->inicio==NULL)
     return LISTA_VAZIA;
  else {
     *d = lt->inicio->info;
     ptemp = lt->inicio;
     ...
  }

}
```



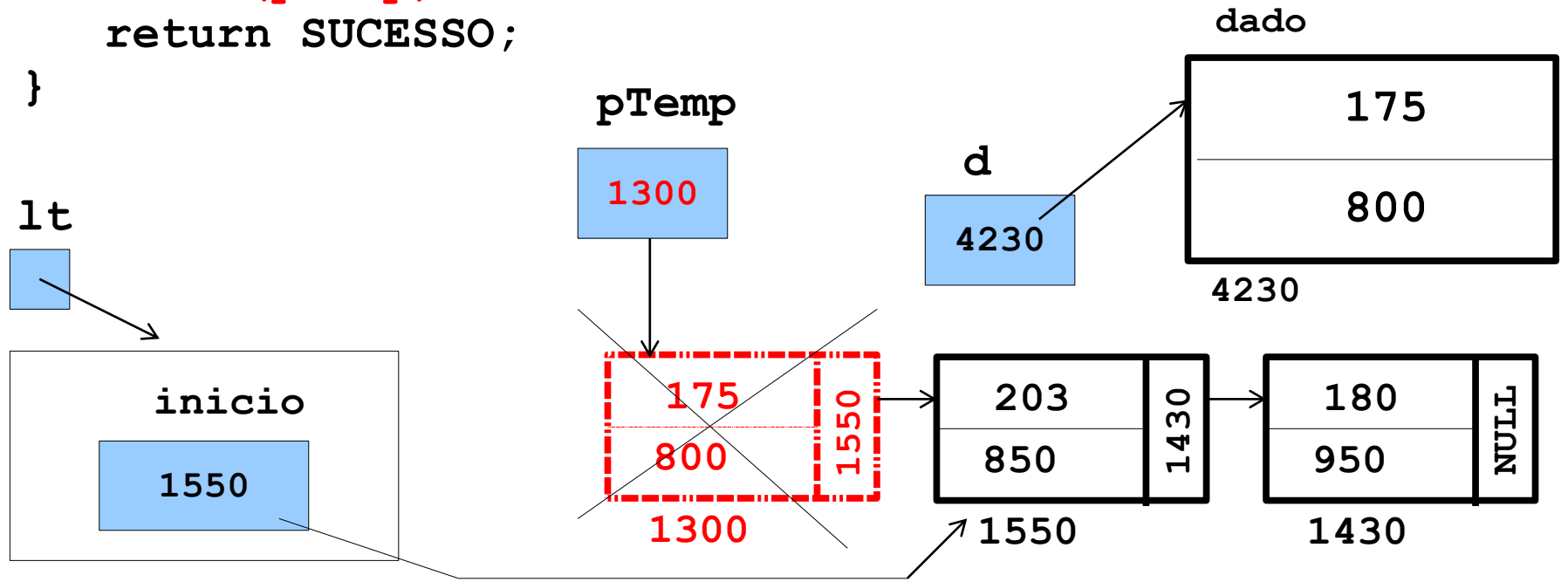
ListaSE - excluiDoInicio

```
int  excluiDoInicio(ListaSE *lt, Dado *d) {
  Nodo *pTemp;
  if (lt->inicio==NULL)
     return LISTA_VAZIA;
  else {
     *d = lt->inicio->info;
     ptemp = lt->inicio;
     lt->inicio = lt->inicio->prox;
     ...
  }

}
```



ListaSE - excluiDoInicio

```
int  excluiDoInicio(ListaSE *lt, Dado *d) {
  Nodo *pTemp;
  if (lt->inicio==NULL)
     return LISTA_VAZIA;
  else {
     *d = lt->inicio->info;
     ptemp = lt->inicio;
     lt->inicio = lt->inicio->prox;
     free (pTemp);
     return SUCESSO;
  }

}
```



ListaSE - incluiNoFim

```
int  incluiNoFim(ListaSE *lt, Dado d) {
...
}
```

Dicas:

1) Quando a lista está vazia, **inicio** deve receber o endereço do novo nodo.

2) Quando a lista contém nodos, **prox** do último deve receber o endereço do novo nodo.

3) Para encontrar o endereço do último interromper a repetição quando **`pAux->prox`** for igual a **`NULL`**.

Estrutura de Dados

Aula 9

Listas Simplesmente Encadeadas

Alocação Dinâmica de Memória

Projeto **ListaSE**