Algoritmos e Estruturas de Dados I

Profª Priscilla Abreu

priscilla.abreu@ime.uerj.br

2022.1

Algoritmos e Estruturas de Dados I

Roteiro da aula

- Alocação Dinâmica

Revisando...

Algoritmos e Estruturas de Dados I

Listas

O QUE É UMA LISTA?

Consideraremos como listas conjuntos sem repetições!

Uma lista é um conjunto de dados relacionados, e de número variável de elementos.

- Exemplo:
  - Lista de alunos de uma turma;
  - Lista de aprovados em um concurso;
  - Lista de produtos de uma loja;
  - ...

Cada elemento da lista terá uma chave – identificador único.

Algoritmos e Estruturas de Dados I

Listas Linear

Listas lineares

Listas lineares gerais
SEM restrição de inserção e remoção de elementos

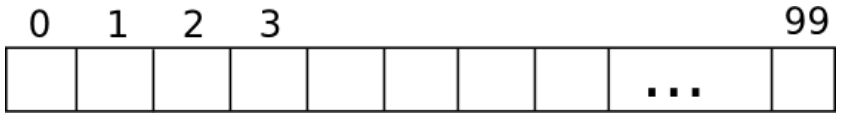
Listas particulares
COM restrição de inserção e remoção de elementos

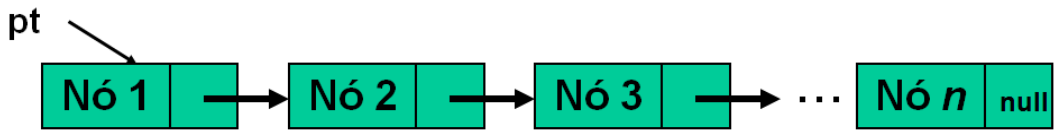
Algoritmos e Estruturas de Dados I

O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa na memória (contínua ou não) de cada dois nós consecutivos na lista.

Existem dois tipos de alocação:

- Alocação sequencial
- Alocação encadeada

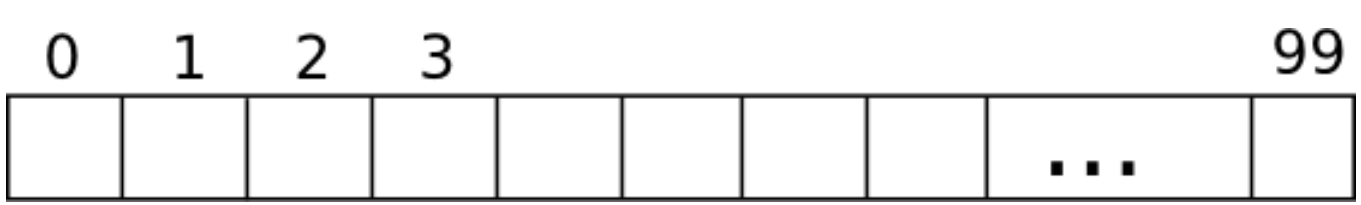




Algoritmos e Estruturas de Dados I

Lista Linear Sequencial

- Uso de vetores.



  - MAX é a quantidade máxima de elementos que a lista poderá armazenar.
  - **n representa o número de elementos.**

Algoritmos e Estruturas de Dados I

Lista Linear com alocação sequencial

- Estruturas sequenciais podem ter seu espaço alocado de forma estática ou dinâmica.
- Através da alocação dinâmica, conseguimos alocar espaços de memória em tempo de execução e modificar o espaço alocado, também em tempo de execução.

Algoritmos e Estruturas de Dados I

Alocação Dinâmica

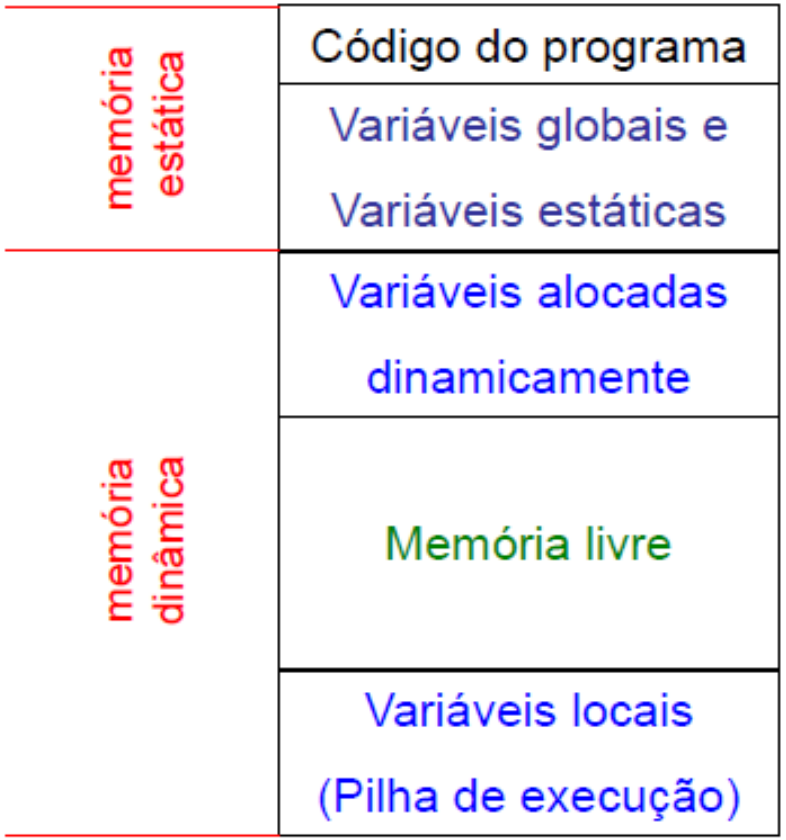
- Processo de solicitar e utilizar memória durante a execução de um programa.
- Visa que um programa utilize apenas a memória necessária pra sua execução, sem desperdícios.
- Deve ser utilizada quando não se sabe inicialmente quanto espaço de memória será necessário para o armazenamento de valores.
- Funções para solicitação e liberação de espaço:
    - **Aloca**
    - **Libera**

Algoritmos e Estruturas de Dados I

Alocação Dinâmica

Funções da linguagem C:

- void * malloc(int qty_bytes_alloc);
- void * calloc(int qtd, int size);
- void * realloc(void * pointer, int new_size);
- free(void * pointer);



Algoritmos e Estruturas de Dados I

Alocação Dinâmica

Exemplo:

```
int *v = (int *) malloc(10 * sizeof(int));
```

Tenta alocar um bloco de memória para armazenar 10 inteiros.

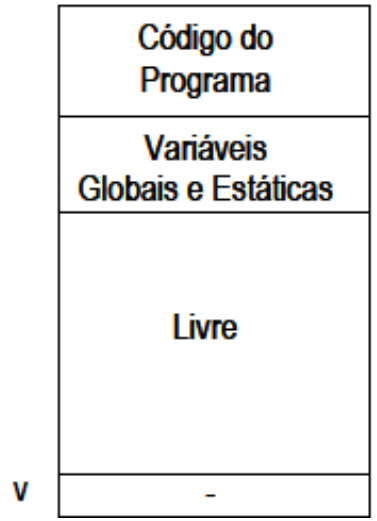
- Se for possível alocar o espaço de 10 inteiros, a função malloc retorna o endereço inicial do bloc de memória;
- Caso contrário, a função retorna o valor NULL;
- Por isso, v é definido como um ponteiro.

Algoritmos e Estruturas de Dados I

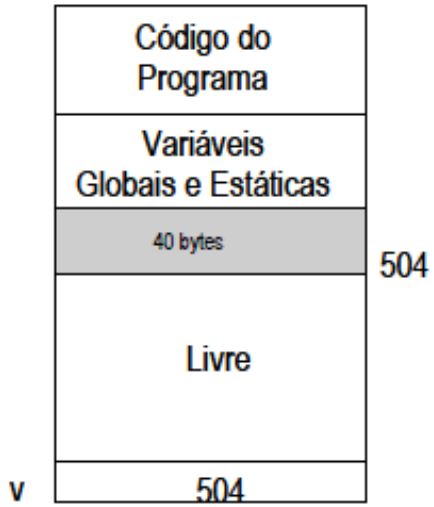
Alocação Dinâmica

```
v = (int *) malloc(10*sizeof(int));
```

1 - Declaração: int *v
Abre-se espaço na pilha para o ponteiro (variável local)



2 - Comando: v = (int *) malloc (10*sizeof(int))
Reserva espaço de memória da área livre e atribui endereço à variável



Algoritmos e Estruturas de Dados I

Alocação Dinâmica

Importante verificar se foi possível alocar o espaço solicitado:

```
...
v = (int*) malloc(10*sizeof(int));
if (v==NULL)
{
   printf("Memoria insuficiente.\n");
   exit(1);   /* aborta o programa e retorna 1 para o sist. operacional */
}
...

free(v);
```

Algoritmos e Estruturas de Dados I

Alocação Dinâmica

- Considerando o espaço alocado, v pode ser tratado como um vetor declarado estaticamente:
    - v aponta para o inicio da área alocada
    - v[0] acessa o espaço para o primeiro elemento
    - v[1] acessa o segundo
    - .... até v[9].

Algoritmos e Estruturas de Dados I

Alocação Dinâmica – exemplo

```
#include <stdio.h>
#include <stdlib.h>
int main (){
      int *v;
      int qtde;
      printf("Informe quantas valores deseja informar: ");
      scanf("%d", &qtde);
      v = (int*) malloc(qtde*sizeof(int));
      if (v == NULL){
            printf(“Erro de alocação!”);
            return;
      }
```

Algoritmos e Estruturas de Dados I

Alocação Dinâmica – exemplo

```
    for (i=0; i<qtde;i++){
        printf("Informe um valor: ");
        scanf("%d", &v[i]);
    }
    printf("\nValores: \n");
    for (i=0; i<qtde;i++){
        printf("%d ", v[i]);
    }
    free(v);
}
```

Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Realocando o espaço do vetor

Realloc

(tipo*) realloc(tipo *apontador, int novo_tamanho)

```
int *x, i;
x = (int *) malloc (200 * sizeof(int));
for (i = 0; i<200; i++){
   printf(“Valor: ”); scanf(&x[i]);
}
x = (int *) realloc (x, 400 * sizeof(int));
x = (int *) realloc (x, 100 * sizeof(int));
free(x);
```

Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Realocando o espaço do vetor

Realloc

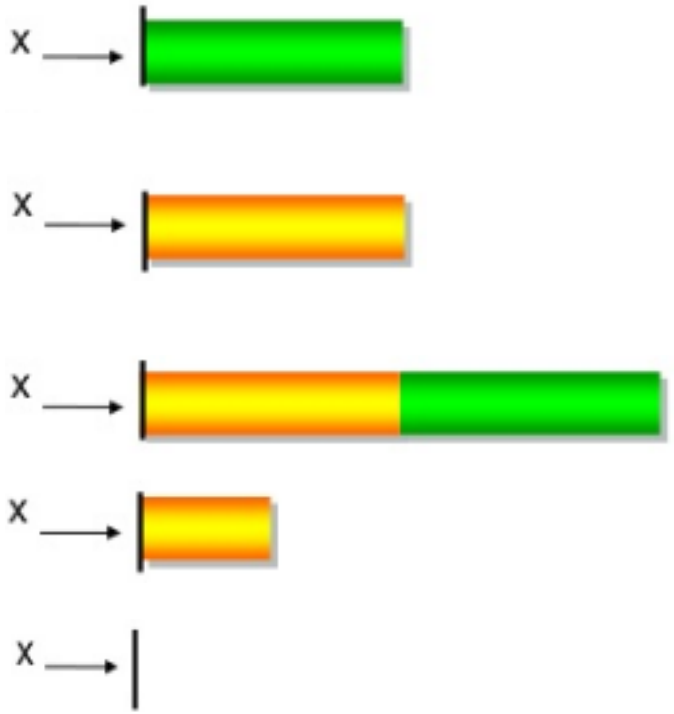
```
int *x, i;
x = (int *) malloc (200 * sizeof(int));

for (i = 0; i<200; i++){
    printf("Valor: "); scanf(&x[i]);
}

x = (int *) realloc (x, 400 * sizeof(int));

x = (int *) realloc (x, 100 * sizeof(int));

free(x);
```



Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Structs

```
#include <stdio.h>
#include <stdlib.h>
#define max 5
typedef struct {
        char nome[40];
        float salario;
}ST_DADOS;

int main(){
        ST_DADOS *p;
        p = (ST_DADOS *) malloc(max*sizeof(ST_DADOS));
```

Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Structs

```
    if (p!=NULL){
            int i;
            for (i=0;i<max;i++){
                    fflush(stdin);
                    printf("\nEntre com o nome: ");
                    fgets(p[i].nome,40,stdin);
                    printf("Entre com o salario: ");
                    scanf("%f", &p[i].salario);
            }
            free(p);
    }
    else{
            printf("\nErro de memória!\n");
    }
}
```

Algoritmos e Estruturas de Dados I

Alocação Dinâmica – considerações

Através desse modo de alocação dinâmica, conseguimos alocar espaços de memória em tempo de execução e modificar o espaço alocado, também em tempo de execução.

No entanto, todo o espaço alocado dinamicamente reservou um bloco de memória com endereços sequenciais, que foram manipulados como vetores.

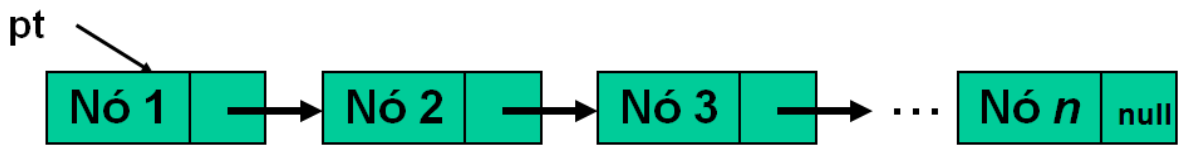
Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

Mas considerando contextos em que as inserções acontecem aleatoriamente, de acordo com a vontade do usuário, nem sempre é possível conseguir espaços em posições contínuas.

Com base no conceito de alocação dinâmica, veremos como alocar espaço de memória dinamicamente elemento a elemento, sem necessariamente os endereços alocados serem endereços sequenciais.

Utilizaremos LISTAS ENCADEADAS!

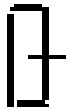


Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

- A proposta desse tipo de lista é que a estrutura esteja, inicialmente, vazia.
- A cada necessidade de inserção de um elemento na lista, um espaço de memória será solicitado para armazenar o novo dado e depois será adicionado na lista, através de alocação dinâmica.

Lista

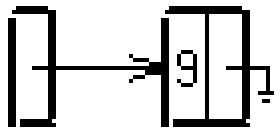


Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

- A proposta desse tipo de lista é que a estrutura esteja, inicialmente, vazia.
- A cada necessidade de inserção de um elemento na lista, um espaço de memória será solicitado para armazenar o novo dado e depois será adicionado na lista, através de alocação dinâmica.

Lista

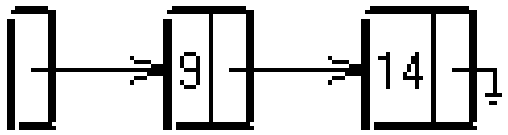


Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

- A proposta desse tipo de lista é que a estrutura esteja, inicialmente, vazia.
- A cada necessidade de inserção de um elemento na lista, um espaço de memória será solicitado para armazenar o novo dado e depois será adicionado na lista, através de alocação dinâmica.

Lista

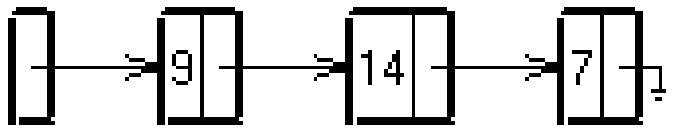


Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

- A proposta desse tipo de lista é que a estrutura esteja, inicialmente, vazia.
- A cada necessidade de inserção de um elemento na lista, um espaço de memória será solicitado para armazenar o novo dado e depois será adicionado na lista, através de alocação dinâmica.

Lista

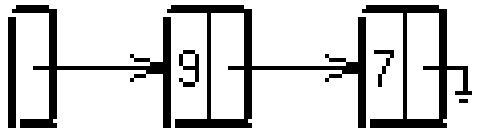


Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

- A proposta desse tipo de lista é que a estrutura esteja, inicialmente, vazia.
- A cada necessidade de inserção de um elemento na lista, um espaço de memória será solicitado para armazenar o novo dado e depois será adicionado na lista, através de alocação dinâmica.

Lista

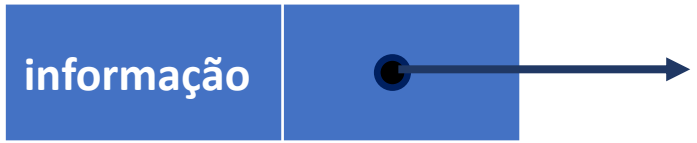


Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

- Desta forma, ao invés dos elementos estarem em sequência na memória, como na lista sequencial, os elementos podem ocupar quaisquer célula de memória.
- Para manter a relação de ordem entre os elementos, cada elemento indica qual é o seguinte, armazenando a referência (endereço) do próximo elemento.

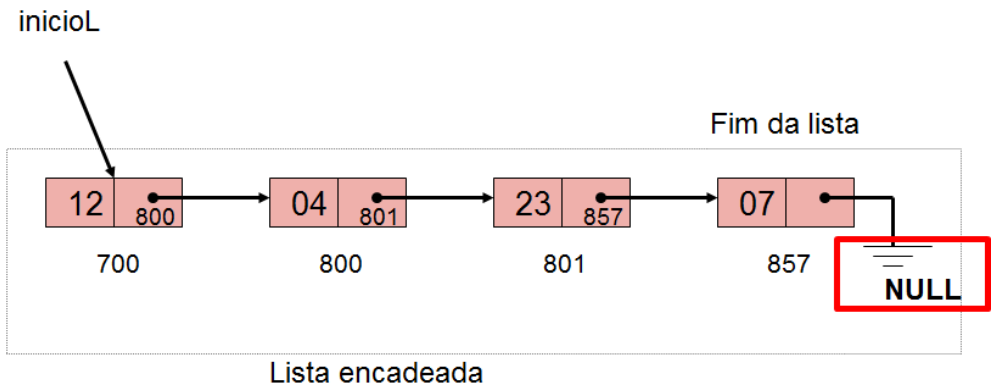
NÓ
Elemento da lista



Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

- Nó da lista é representado por pelo menos dois campos:
  - a informação armazenada;
  - o ponteiro para o próximo elemento da lista.
- a lista é representada por um ponteiro para o primeiro nó;
- o campo próximo do último elemento é NULL.




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

no *inicioL;
```

Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

```
#include <stdio.h>
#include <stdlib.h>
typedef struct no{
      int info;
      struct no *prox;
}no;
no *inicioL;
```

OPERAÇÕES:

- **Criar lista**
- **Lista vazia**
- **Inserir**
- **Percorrer**
- **Remover**

Algoritmos e Estruturas de Dados I

Alocação Dinâmica – Listas Encadeadas

```
void inicializa_lista () {
   inicioL = NULL;
}

int lista_vazia () {
   if (inicioL== NULL)
            return 1;
   return 0;
}
```

Algoritmos e Estruturas de Dados I

Inserção em listas encadeadas

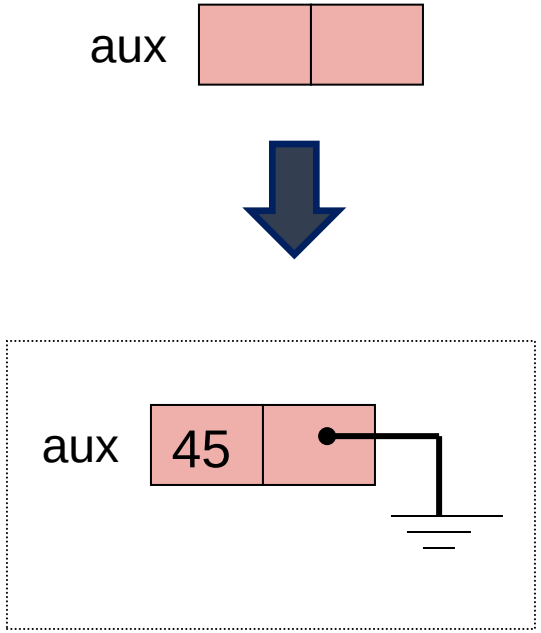
- Inserção no início
- Inserção em posições aleatórias
- Inserção no final

Antes de inserir um elemento na lista é necessário alocar um espaço para seu armazenamento.

Algoritmos e Estruturas de Dados I

Inserção em listas encadeadas

```
void inserir(int valor){
    no  *aux;
    aux = (no*) malloc(sizeof(no));
    if (aux != NULL){
            aux ->info= valor;
    }
   ...
}
```



Algoritmos e Estruturas de Dados I

Inserção em listas encadeadas

```
void inserir(int valor){
   no  *aux;
   aux = (no*) malloc(sizeof(no));
   if (aux != NULL){
          aux ->info= valor;
   }
  ...
}
```

Com a criação do elemento a ser inserido, já é possível fazer a inserção na lista!

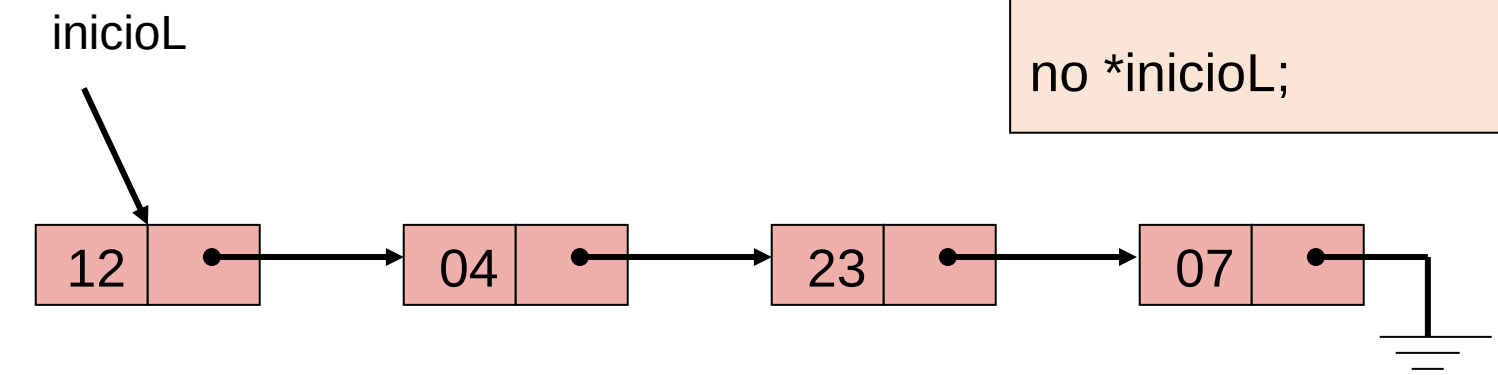
Algoritmos e Estruturas de Dados I

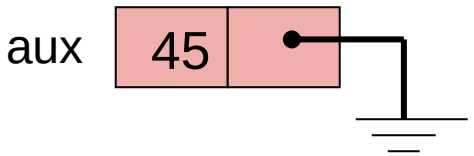
Inserção em listas encadeadas

- Inserção no início

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

no *inicioL;
```





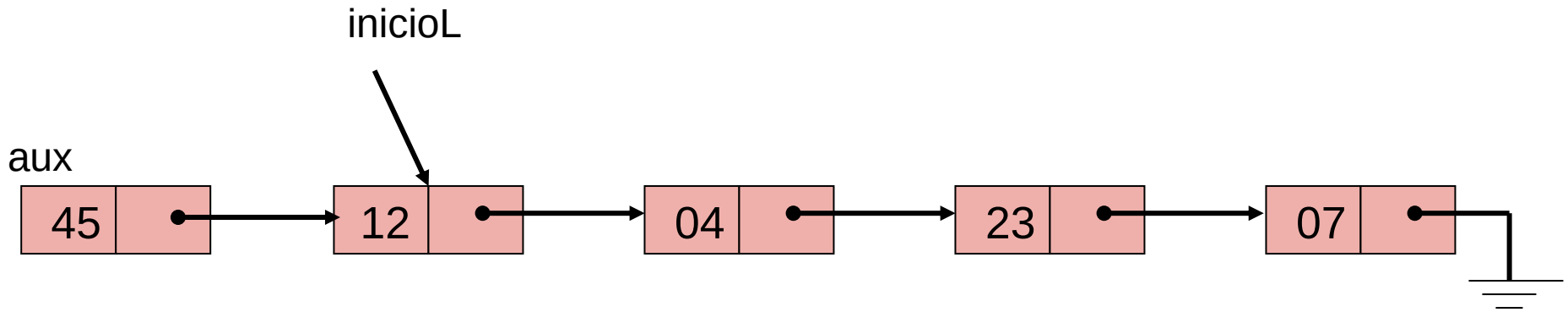
Elemento a ser inserido

Algoritmos e Estruturas de Dados I

Inserção em listas encadeadas

- Inserção no início

Em seguida, o elemento criado (aux) precisa ser o primeiro da lista. Logo, ele deve indicar como seu próximo elemento, o que era até aquele momento o primeiro da lista (inicioL).

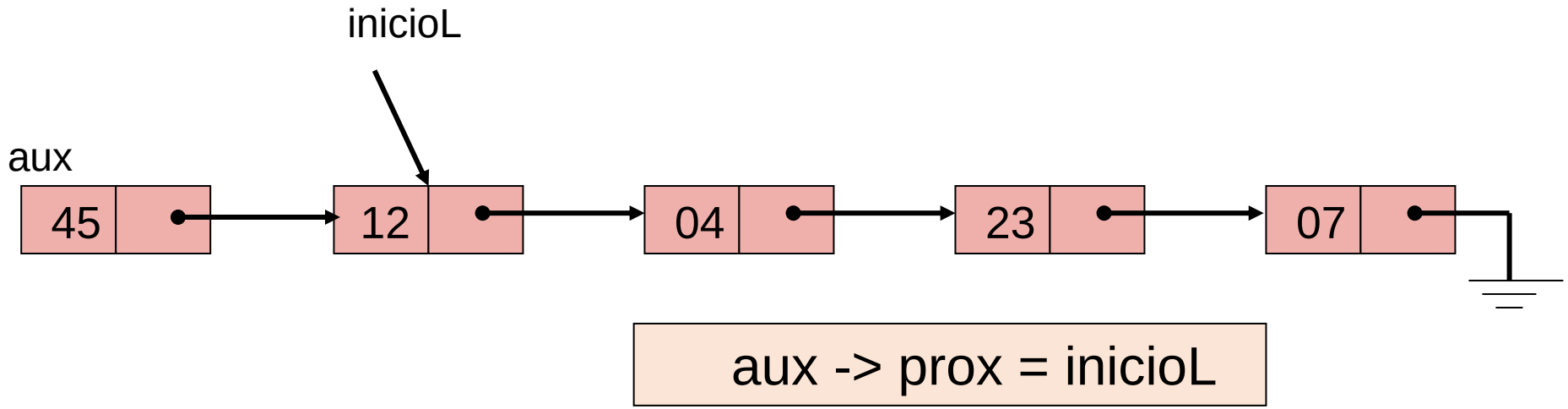


Algoritmos e Estruturas de Dados I

Inserção em listas encadeadas

- Inserção no início

Em seguida, o elemento criado (aux) precisa ser o primeiro da lista. Logo, ele deve indicar como seu próximo elemento, o que era até aquele momento o primeiro da lista (inicioL).

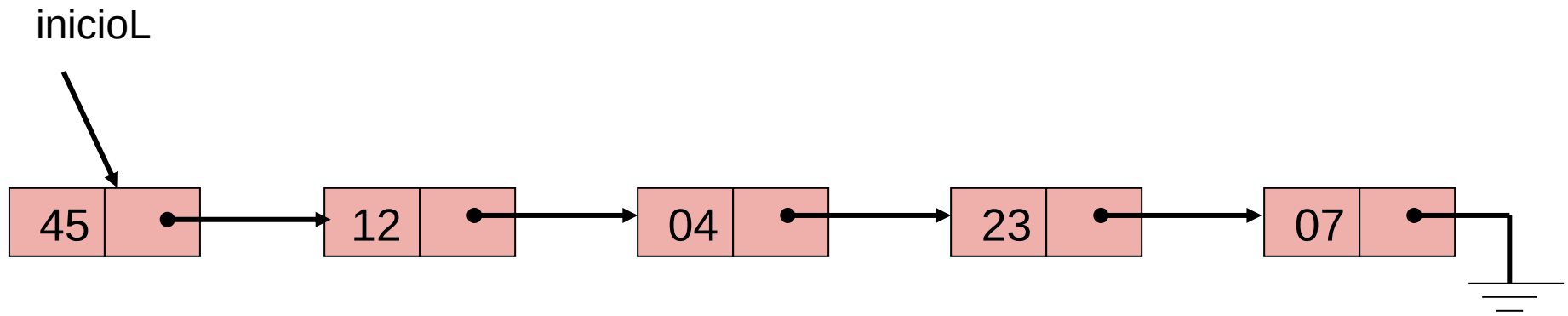


Algoritmos e Estruturas de Dados I

Inserção em listas encadeadas

- Inserção no início

Agora, o ponteiro que indica o início da lista deve ser atualizado e passar a referenciar o elemento que acabou de ser inserido no início da lista.




```
inicioL = aux;
```

Algoritmos e Estruturas de Dados I

Inserção em listas encadeadas

- Inserção no início

```
void inserir(int valor){
        no  *aux;
        aux = (no*) malloc(sizeof(no));
        if (aux != NULL){
                aux ->info= valor;
                aux -> prox = inicioL;
                inicioL = aux;
        }
}
```

COMO PERCORRER A LISTA?

Algoritmos e Estruturas de Dados I

Percurso

Percurso

Para percorrer uma lista precisamos de uma variável auxiliar. Tal variável será responsável por acessar cada elemento. Isso não pode ser feito com a variável **inicioL**, pois esta indica o início da lista e se modificarmos seu valor perderemos a referência de onde inicia a lista.

Algoritmos e Estruturas de Dados I

Percurso

Percurso

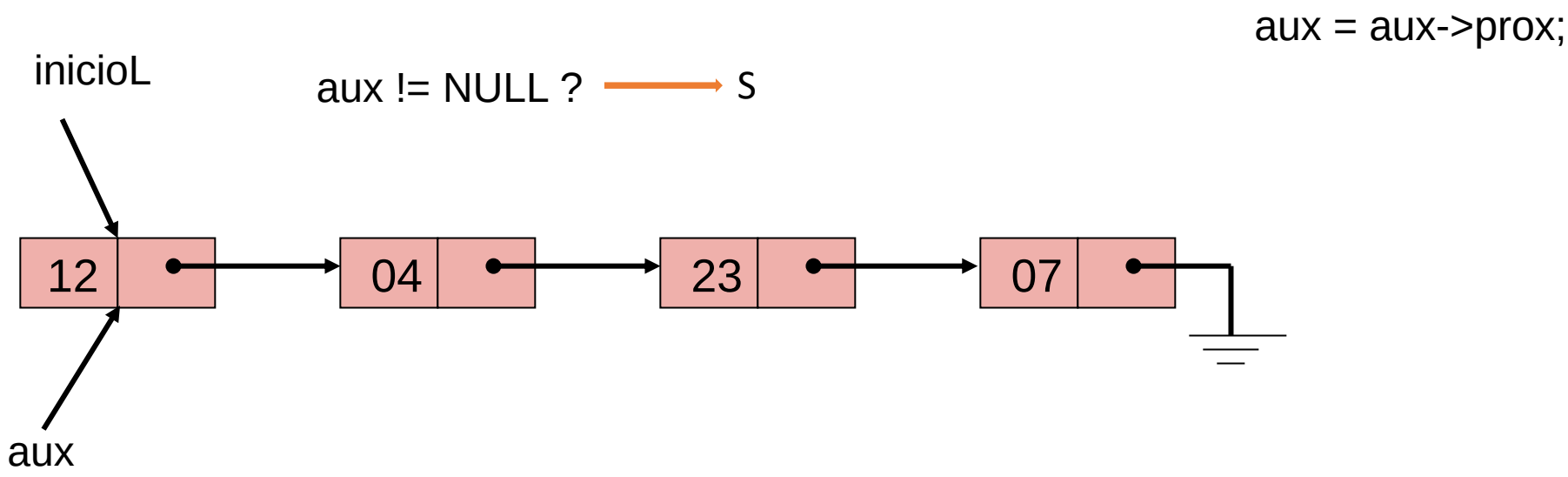
A variável auxiliar (aux) será inicializada com a referência de início da lista e irá percorrendo a lista até chegar ao seu final.

O final é encontrado ao atingir o valor NULL.

Algoritmos e Estruturas de Dados I

Percurso

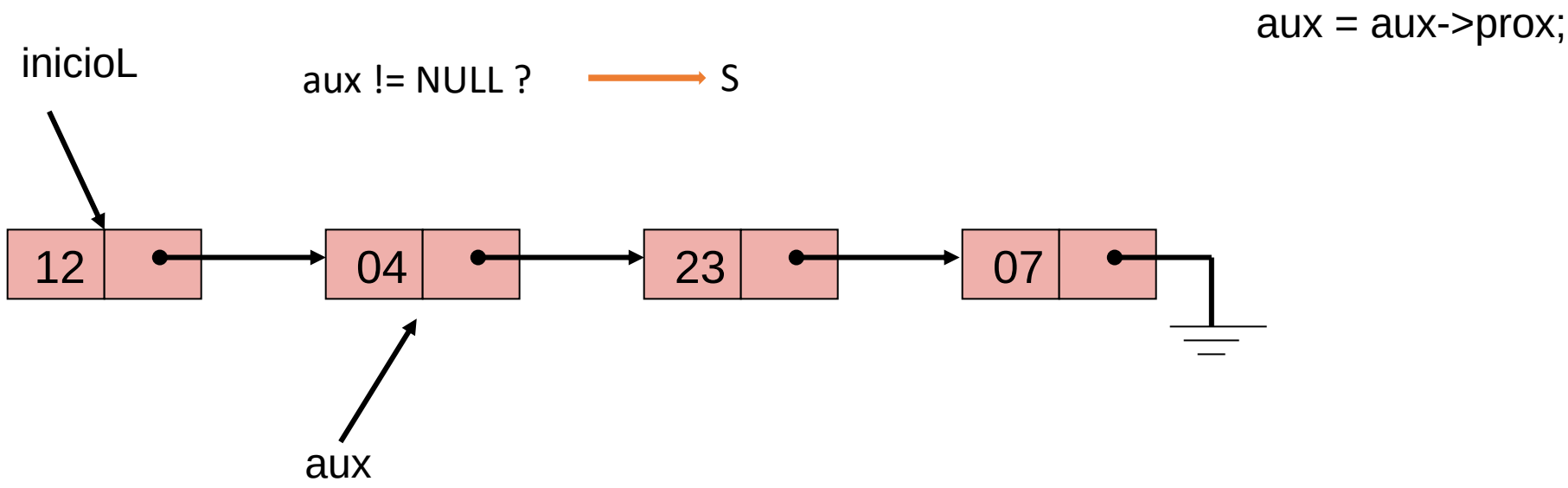
Percurso – condição de parada para percorrer



Algoritmos e Estruturas de Dados I

Percurso

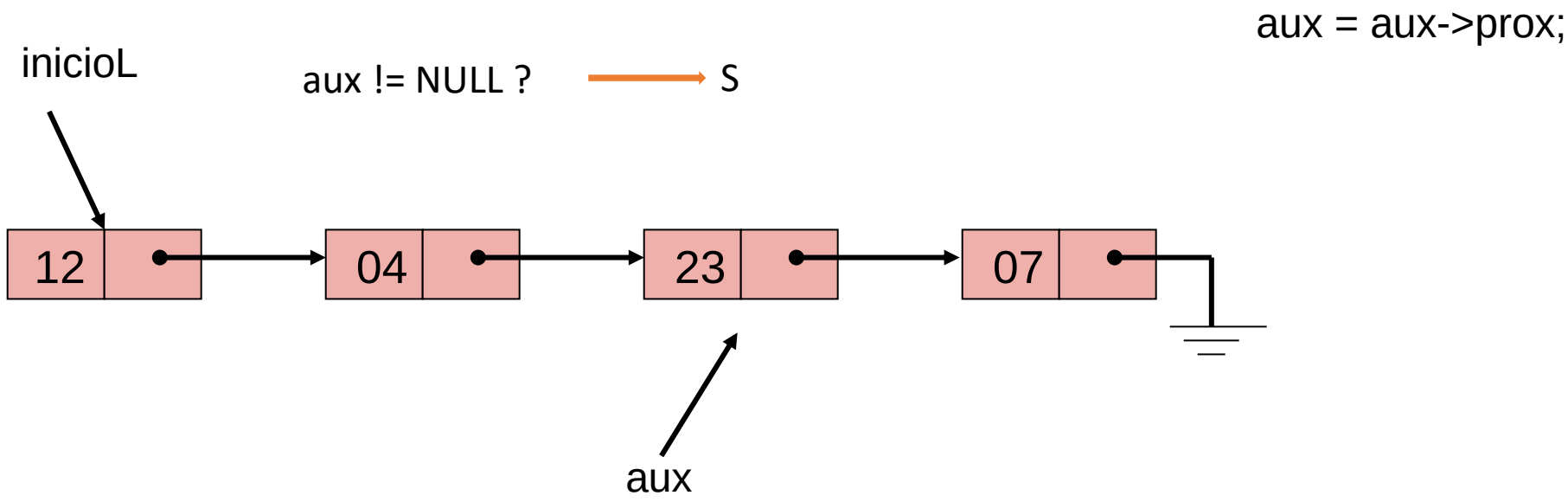
Percurso – condição de parada para percorrer



Algoritmos e Estruturas de Dados I

Percurso

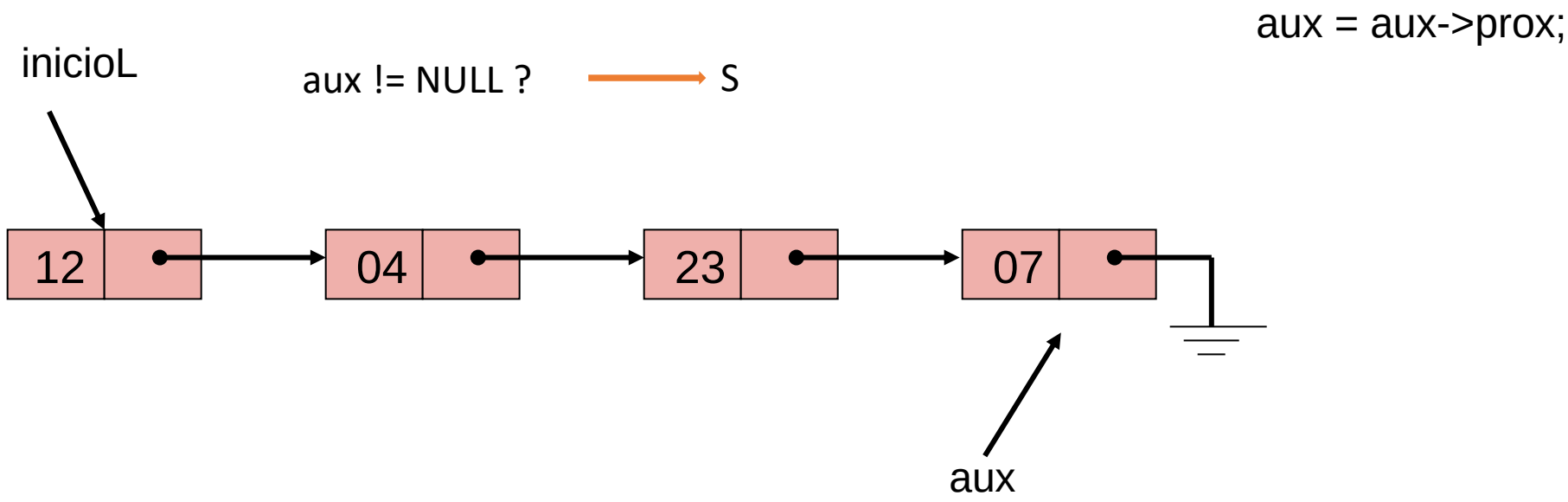
Percurso – condição de parada para percorrer



Algoritmos e Estruturas de Dados I

Percurso

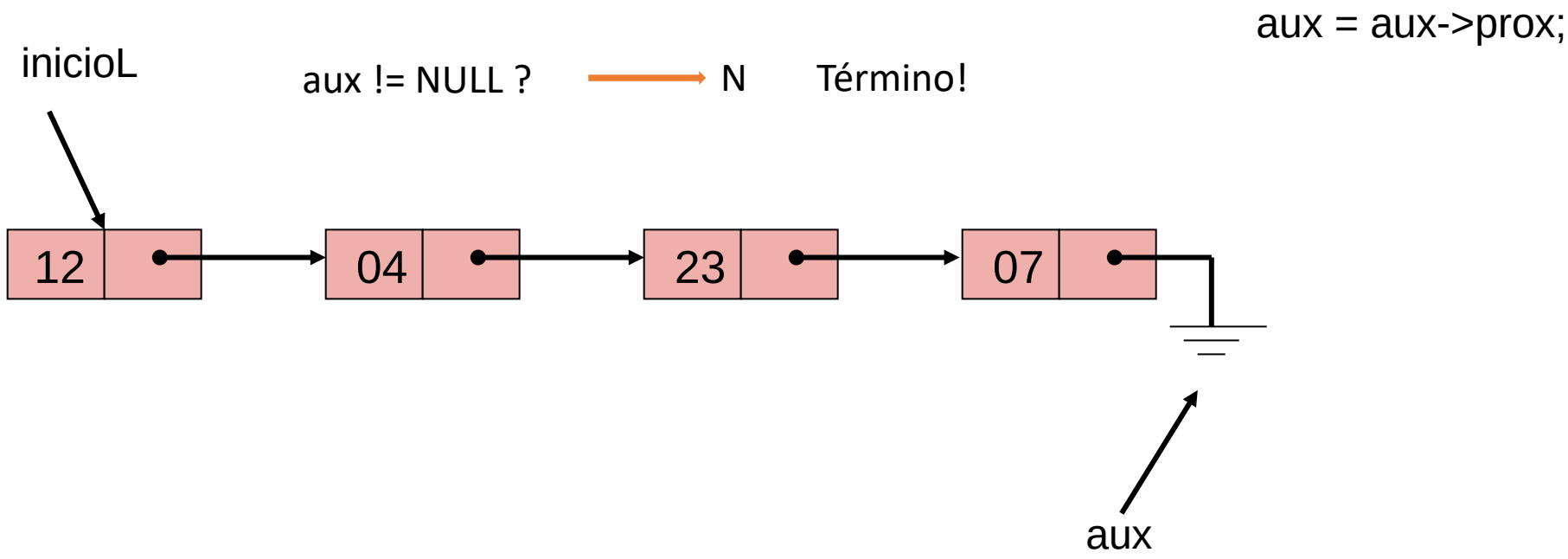
Percurso – condição de parada para percorrer



Algoritmos e Estruturas de Dados I

Percurso

Percurso – condição de parada para percorrer



Algoritmos e Estruturas de Dados I

Percurso

```
void percorrer () {
        if(!lista_vazia()){
                no * aux;
                aux = inicioL;
                while (aux!= NULL) {
                        printf("%d", aux->info);
                        aux = aux->prox;
                }
        }
        else{
                printf("\n Lista vazia!\n");
        }
}
```

E para inserir um elemento ao final da lista, como fazer?

Algoritmos e Estruturas de Dados I

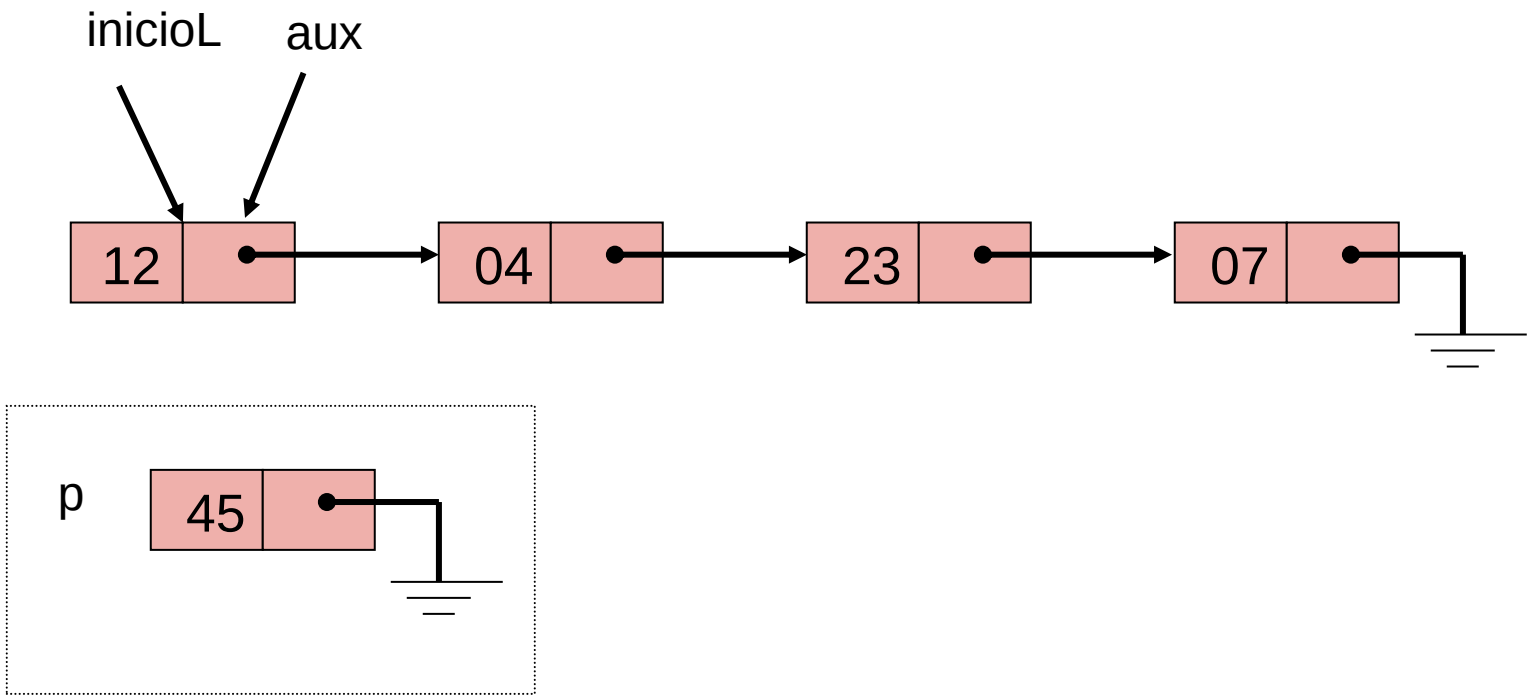
Inserção no final

A variável auxiliar (aux) será inicializada com a referência de início da lista e irá percorrendo a lista até chegar no último elemento, sem chegar no valor NULL.

Estando com a referência desse último elemento, já é possível ligar seu campo **prox** ao nó a ser inserido na lista.

Algoritmos e Estruturas de Dados I

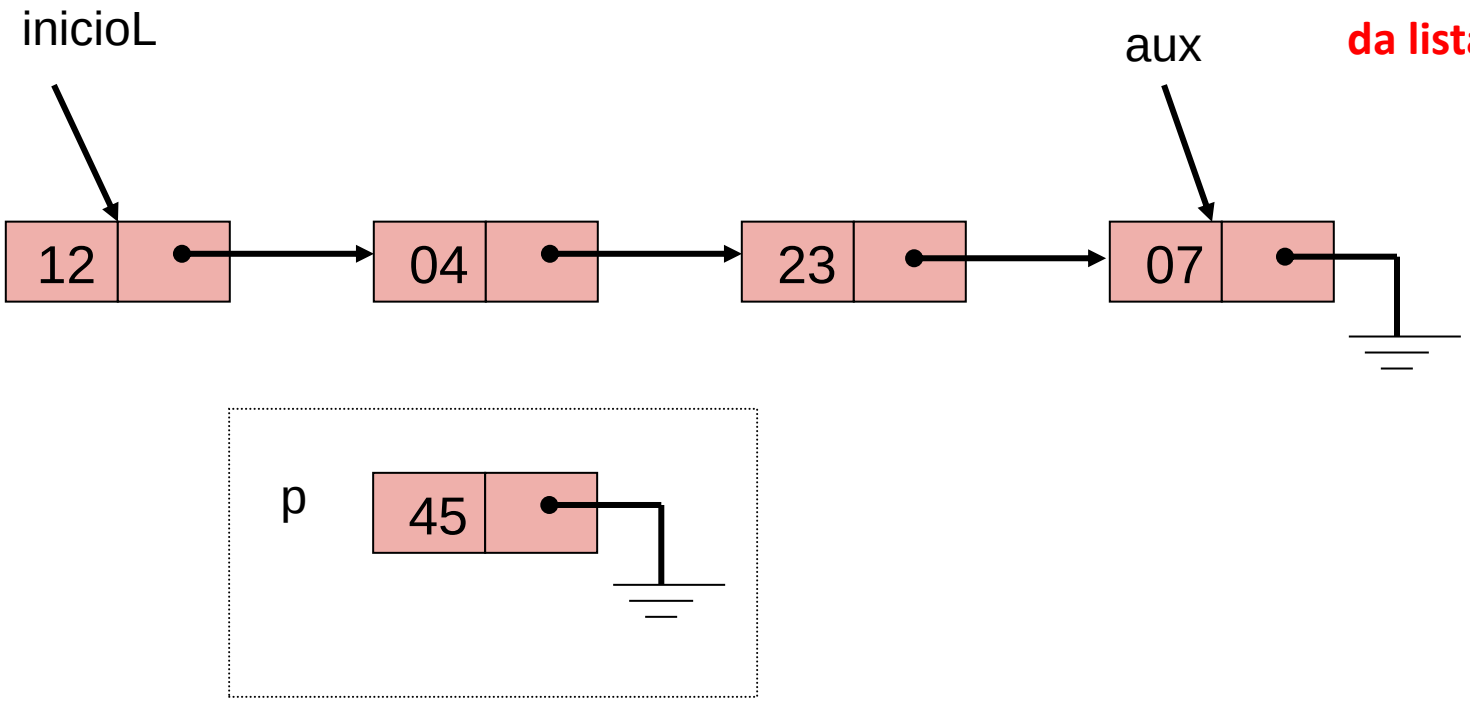
Inserção no final



Algoritmos e Estruturas de Dados I

Inserção no final

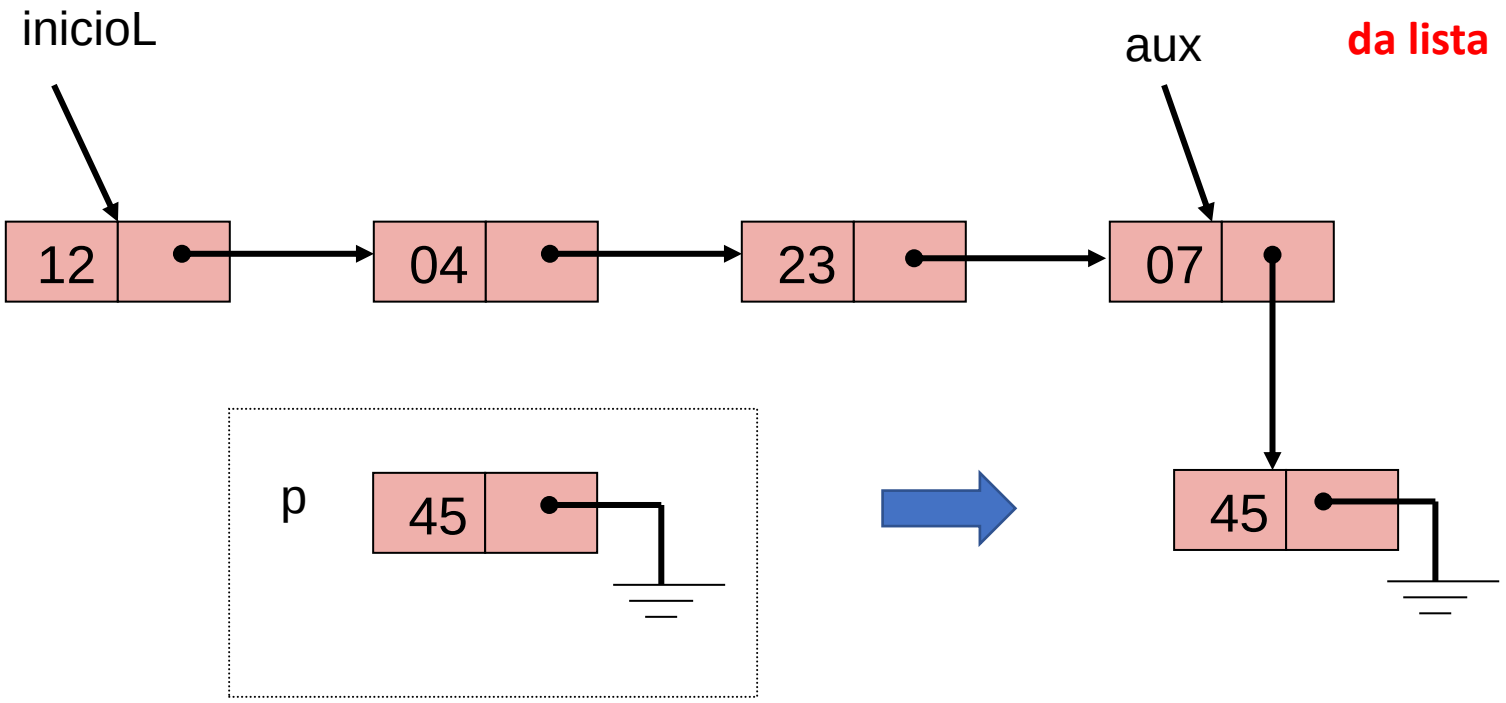
Ponteiro utilizado para encontrar o último elemento da lista



Algoritmos e Estruturas de Dados I

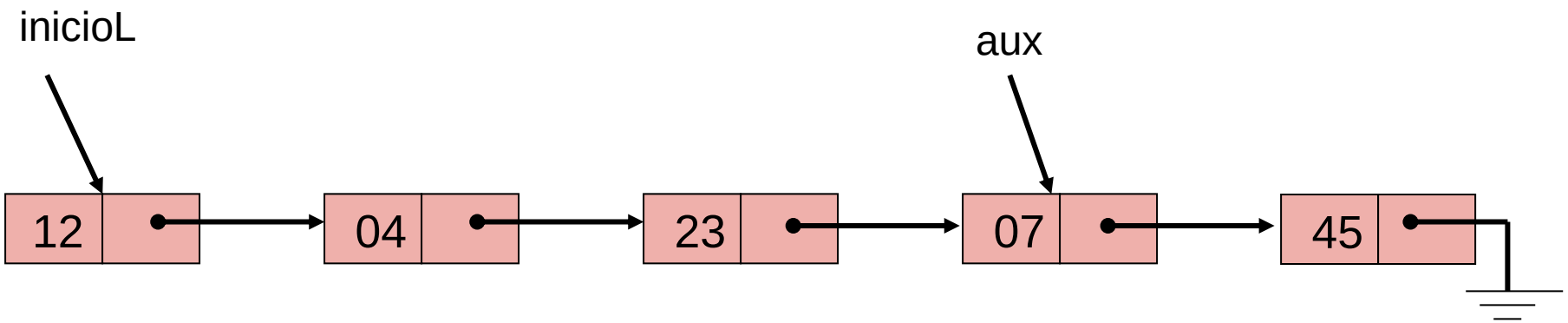
Inserção no final

Ponteiro utilizado para encontrar o último elemento da lista



Algoritmos e Estruturas de Dados I

Inserção no final




```
aux->prox = p;
```

Algoritmos e Estruturas de Dados I

Inserção no final

```
void inserir_fim (int valor) {
    no *aux, *p;
    aux = (no*) malloc(sizeof(no));
    if (aux != NULL){
            aux ->info= valor;
            aux -> prox = NULL;
            if (lista_vazia()){
                    inicioL=aux;
            }
            else{
                    p = inicioL;
                    while (p->prox != NULL)
                            p = p -> prox;
                    p->prox = aux;
            }
    }
}
```

Algoritmos e Estruturas de Dados I

EXERCÍCIO

Faça um programa para preenchimento de uma lista encadeada de números inteiros, utilizando as funções apresentadas. Você deve apresentar ao usuário um menu com as seguintes opções:

1- Inserir
2- Exibir a lista
3- Sair

FIM