

# Programação

## 5: Estruturas Dinâmicas

### 5.1: Listas Ligadas Simples

Francisco Pereira ([xico@isec.pt](mailto:xico@isec.pt))

---

# Problema

- Criar um programa para gerir os livros de uma biblioteca
  - Estrutura provisória

```
typedef struct objeto livro;

struct objeto{
    char titulo[100], autor[100];
    int cota;
};
```

# Problema

---

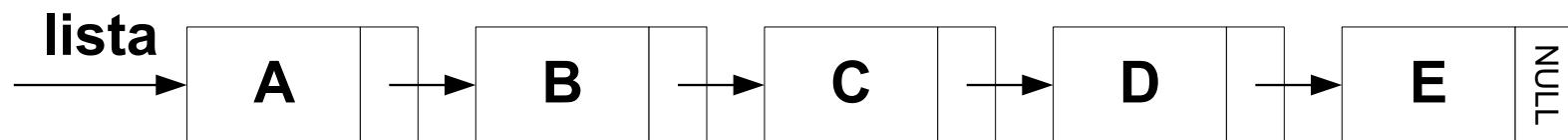
- Operações
  - Listar, Adicionar, Eliminar livros
- Requisitos
  - O número de livros varia ao longo da execução
  - Os livros estão ordenados por cota
    - Operações de inserção e eliminação num vetor dinâmico são algo complexas e demoradas

Gestão dinâmica  
de memória

Listas Ligadas

# Listas ligadas

- **Lista ligada:**



- Estrutura de dados dinâmica constituída por um conjunto variável de elementos (i.e., **estruturas**).
- Cada estrutura contém:
  - Campos com informação
  - Ponteiro para o próximo elemento da lista.
- Principal vantagem: Flexibilidade

# Lista ligadas: Características

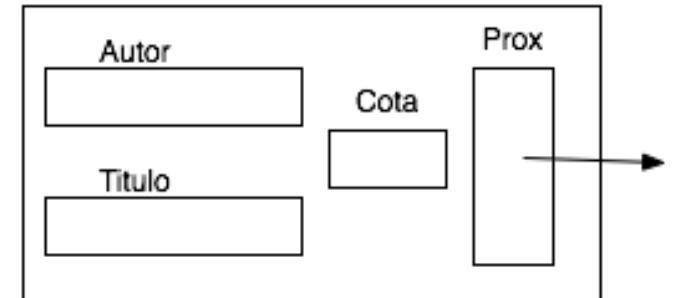
---

- Número de elementos varia ao longo do tempo
  - No início, a lista está vazia
- Ponteiro de lista aponta para o primeiro elemento
  - Única variável declarada.
- Cada elemento da lista aponta para o seguinte
- Acesso sequencial

# Gestão dos livros da biblioteca

- Definição do tipo de dados a utilizar
  - Campos com informação: Título, autor e cota
  - **Novo campo:** Ponteiro para o próximo elemento da lista

```
typedef struct objeto livro, *plivro;  
  
struct objeto{  
    char titulo[100], autor[100];  
    int cota;  
    plivro prox;  
};
```



# Listas ligadas: Operações básicas

---

1. Criar uma lista.
2. Verificar se uma lista está vazia.
3. Listar/Pesquisar informação:
  - a) Mostrar toda a informação armazenada.
  - b) Procurar um elemento específico.
4. Adicionar um elemento.
5. Retirar um elemento.
6. Destruir uma lista ligada.

# Operação 1: Criar uma lista ligada

- Arranjar um ponteiro de lista:
  - No início, a lista está vazia.

```
int main()
{
    plivro lista = NULL;
```

...

Variável local da  
função `main( )`

## Operação 2: Verificar se a lista está vazia

```
int lista_vazia(plivro p);
```

- Recebe como argumento o ponteiro de lista.
- Devolve 1 se a lista estiver vazia (0, caso contrário).

```
int lista_vazia(plivro p)
{
    if(p == NULL)
        return 1;
    else
        return 0;
}
```

# Operação 3A: Percorrer uma lista ligada

```
void mostra_info(plivro p);
```

- Recebe como argumento o ponteiro da lista.
  - Escreve o autor e o título de todos os livros.

```
void mostra_info(plivro p)
{
    while(p != NULL)
    {
        printf("%s\t%s\t%d\n",
               p->autor, p->titulo, p->cota);
        p = p->prox;
    }
}
```

Salto para o próximo  
elemento da lista

# Operação 3B: Procurar um elemento

```
void procura_cota(plivro p, int c);
```

- Procura o livro que tenha a cota passada por argumento.

```
void procura_cota(plivro p, int c)
{
    while(p != NULL && p->cota != c)
        p = p->prox;
    if(p != NULL)
        printf("%s\t%s\n", p->autor, p->titulo);
    else
        printf("Cota inexistente\n");
}
```

## Operação 4: Adicionar um elemento

---

- Inserir um novo elemento numa lista ligada
- Três etapas:
  1. Arranjar espaço em memória para o novo nó.
  2. Preencher o novo nó com informação.
  3. Inserir o novo nó na lista.

# Operação 4: Adicionar um elemento

- Tipos de inserção:

- No início.

```
plivro insere_inicio(plivro p);
```

- No final.

```
plivro insere_final(plivro p);
```

- Entre dois elementos (inserção ordenada).

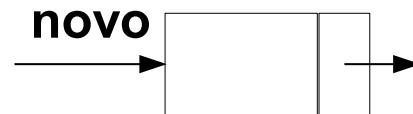
```
plivro insere_ord(plivro p);
```

# Passo 1: Arranjar espaço em memória

```
plivro insere_inicio(plivro p)
{
    plivro novo;

    novo = malloc(sizeof(livro));
    if(novo == NULL)
    {
        printf("Erro na alocacao de memoria\n");
        return p;
    }
    ...
}
```

Se alocação de  
memória falhar, o  
processo termina



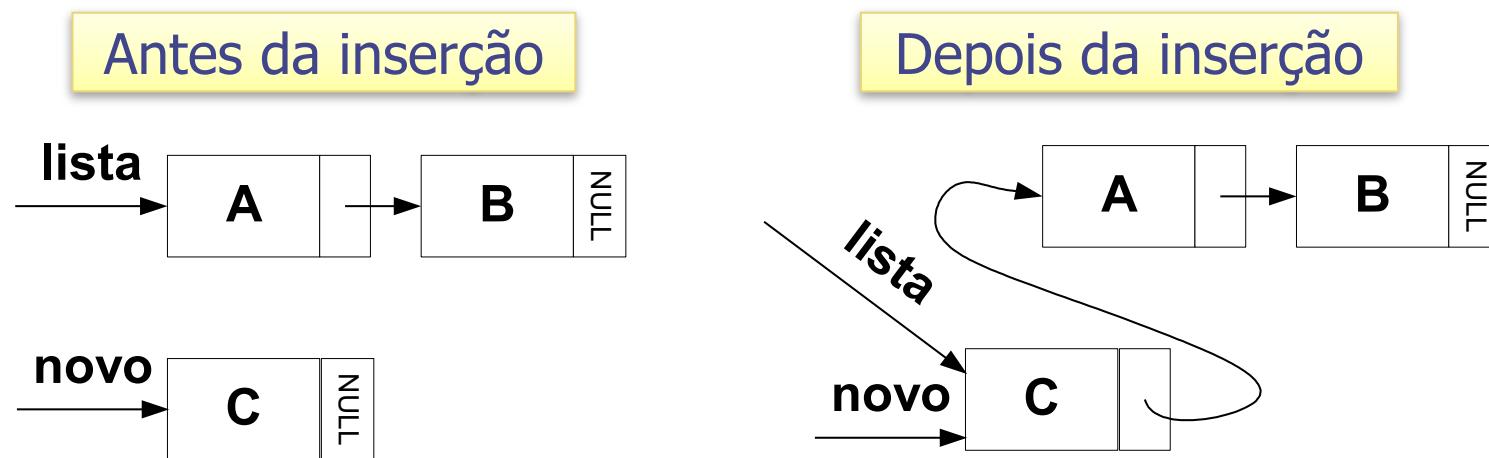
# Passo 2: Preencher nó com informação

```
void preenche(plivro p)
{
    printf("Titulo: ");
    scanf(" %99[^\\n]", p->titulo);
    printf("Autor: ");
    scanf(" %99[^\\n]", p->autor);
    printf("Cota: ");
    scanf("%d", &p->cota);
    p->prox = NULL;
}
```

Ponteiro para o  
nó a preencher

# Passo 3A: Inserir novo nó no início

- Exemplo:



- Caso particular:
  - E se a lista estiver vazia?

# Passo 3A: Inserir novo nó no início

```
plivro insere_inicio(plivro p)
{
    plivro novo;

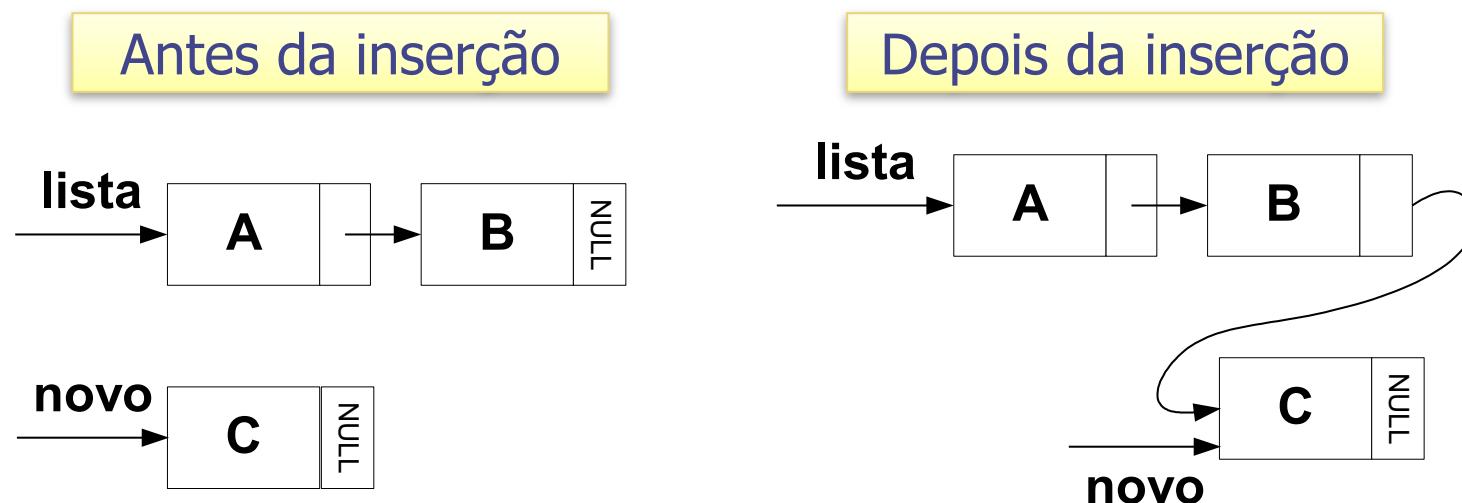
    if (novo = malloc(sizeof(livro))) == NULL)
        printf("Erro na alocacao de memoria\n");
    else
    {
        preenche(novo);
        novo->prox = p;
        p = novo;
    }
    return p;
}
```

Ponteiro para o início  
da lista modificada

Ponteiro para o  
início da lista

# Passo 3B: Inserir novo nó no final

- Exemplo:



- Caso particular:
  - E se a lista estiver vazia?

## Passo 3B: Inserir novo nó no final

```
plivro insere_final(plivro p)
{
    plivro novo, aux;

    novo = malloc(sizeof(livro));
    if(novo == NULL)
    {
        printf("Erro na alocacao de memoria\n");
        return p;
    }
    preenche(novo);
    if(p == NULL)
        p = novo;
    else
    {
        aux = p;
        while(aux->prox != NULL)
            aux = aux->prox;
        aux->prox = novo;
    }
    return p;
}
```

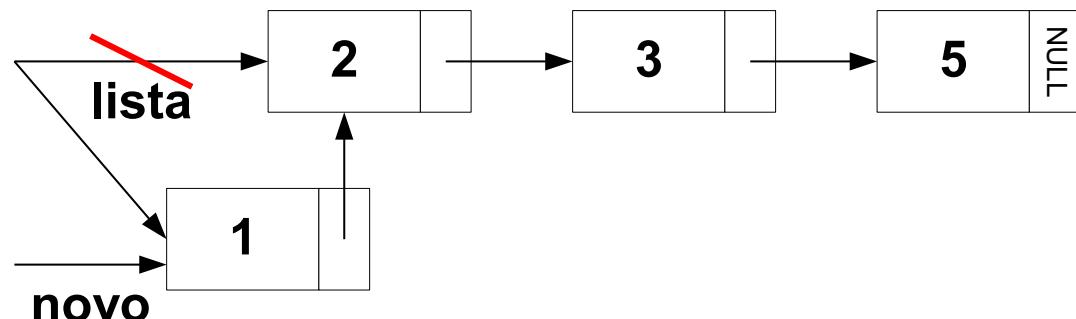
Caso particular

Caso geral

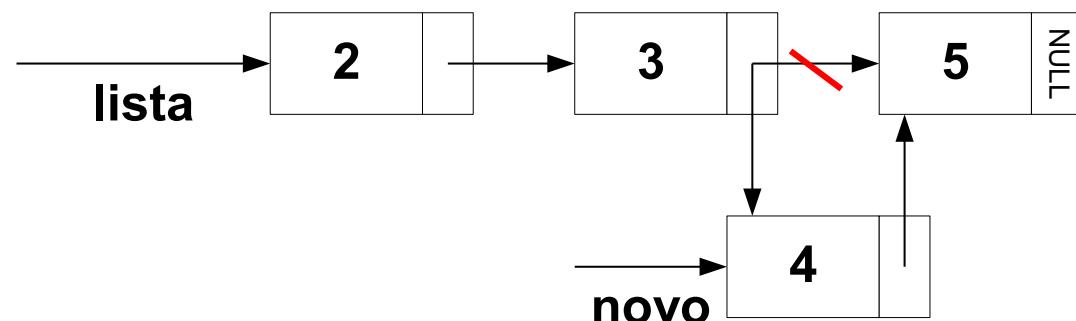
## Passo 3C: Inserção numa lista ordenada

- Considerar que os livros se encontram ordenados por ordem crescente de cota.

Inserção à  
cabeça da lista



Inserção entre  
dois elementos



# Passo 3C: Inserção numa lista ordenada

```
plivro insere_ord(plivro p)
{
    plivro novo, aux;

    novo = malloc(sizeof(livro));
    if(novo == NULL)
    {
        printf("Erro na alocacao de memoria\n");
        return p;
    }
    preenche(novo);
    if(p == NULL || novo->cota < p->cota)
    {
        novo->prox = p;
        p = novo;
    }
    else
    {
        aux = p;
        while(aux->prox != NULL &&
              novo->cota > aux->prox->cota)
            aux = aux->prox;
        novo->prox = aux->prox;
        aux->prox = novo;
    }
    return p;
}
```

Caso particular

Caso geral

# Operação 5: Eliminar um elemento

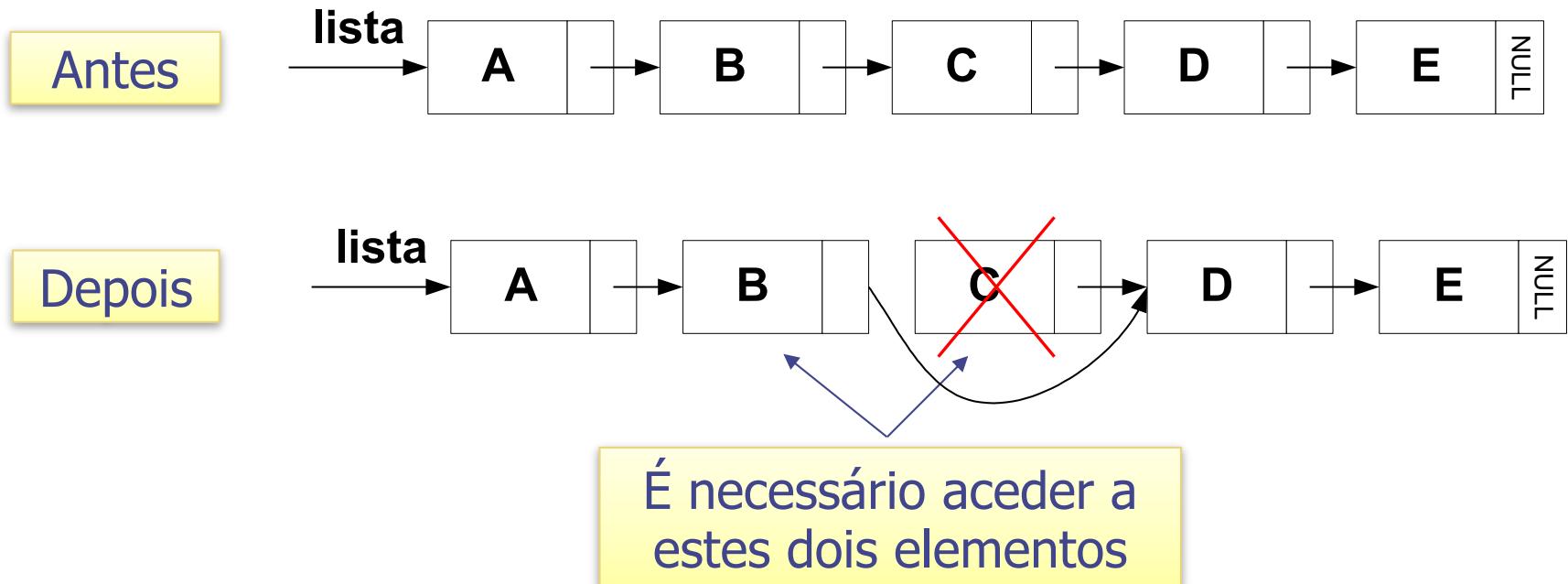
- Eliminar um elemento:
  - Localizar o elemento a eliminar.
  - Alterar a lista de modo a que este nó deixe de ser necessário.
  - Libertar o espaço ocupado pelo nó.

```
plivro elimina(plivro p, int c);
```

- Retira da lista o livro com cota  $c$  passada como argumento.

# Operação 5: Eliminar um elemento

- Exemplo: o livro C possui a cota procurada



- Caso particular:
  - E se o elemento a eliminar estiver no início da lista?

# Operação 5: Eliminar um elemento

```

plivro elimina(plivro p, int c)
{
    plivro atual, anterior = NULL;

    atual = p;
    while(atual != NULL && atual->cota != c)
    {
        anterior = atual;
        atual = atual->prox;
    }
    if(atual == NULL)
        return p;
    if(anterior == NULL)
        p = atual->prox;
    else
        anterior->prox = atual->prox;
    free(atual);
    return p;
}
  
```

Nó não encontrado

Primeiro nó da lista

Caso geral

# Operação 6: Destruir a lista

- Quando uma lista já não é necessária, o espaço ocupado pelos seus elementos deve ser libertado.

```
void liberta_lista(plivro p)
{
    plivro aux;

    while(p != NULL)
    {
        aux = p;
        p = p->prox;
        free(aux);
    }
}
```