

# CI1001 - Programação I

André Grégio, Fabiano Silva, Luiz Albini e Marcos Castilho  
Departamento de Informática – UFPR, Curitiba/PR

## Nona lista de exercícios

---

### Tipo Abstrato de Dados Lista

Implementação de outra possível biblioteca e de funções complementares

---

Neste trabalho você deve criar duas bibliotecas distintas em C:

1. `lib_lista.c`: implementa as funções que estão no arquivo de *headers* `lib_lista.h` (será fornecido). Essas funções manipulam uma lista encadeada de nodos, conforme a especificação que segue;
  - (a) A implementação deve manipular diretamente a estrutura de dados fornecida no arquivo de *headers*;
  - (b) Sua implementação deve garantir que todo espaço de memória alocado dinamicamente seja liberado quando não for mais necessário;
  - (c) Use o programa `valgrind` para garantir que o item anterior seja atendido!!!
  - (d) Um exemplo de saída correta para o `valgrind`:

```
ci1001@fradim:~/tmp/$ valgrind ./testa
==5051== HEAP SUMMARY:
==5051==    in use at exit: 0 bytes in 0 blocks
==5051==   total heap usage: 23 allocs, 23 frees, 1,376 bytes allocated
==5051==
==5051== All heap blocks were freed -- no leaks are possible
==5051==
==5051== For counts of detected and suppressed errors, rerun with: -v
==5051== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

- (e) A parte importante é a linha contendo  
`All heap blocks were freed -- no leaks are possible;`

2. `lib_lista_complementar.c`: implementa as funções que estão no arquivo de *headers* `lib_lista_complementar.h` (será fornecido).

- (a) A implementação **não** deve manipular diretamente a estrutura de dados lista, mas deve **usar** as funções da biblioteca `lib_lista.h`.

Observações:

- Você deve estar atento para não realizar instruções sobre listas inválidas, sob pena de obter um *segmentation fault*. Por exemplo, uma lista não inicializada ou destruída não pode ser impressa.
- Faça boas apresentações dos códigos (legibilidade, identificação, nomes adequados de variáveis e de constantes). Bons comentários no código são fundamentais, mas não exagere no número deles, use-os com bom senso em situações pertinentes.

---

## Especificação da estrutura de dados

- O tipo *t\_nodo* é uma estrutura que contém três campos:
  - um elemento do tipo *int*;
  - um apontador do tipo *t\_nodo* que aponta para o próximo nodo da lista;
  - um apontador do tipo *t\_nodo* que aponta para o nodo anterior da lista;
- A lista contém dois nodos especiais do tipo *t\_nodo*, que servirão como sentinelas e visam facilitar a implementação de algumas funções;
- O tipo *t\_lista* é uma estrutura que contém quatro campos:
  - um apontador para o início da lista de nodos do tipo *t\_nodo*;
  - um apontador para o final da lista de nodos do tipo *t\_nodo*;
  - um apontador para algum nodo da lista de nodos do tipo *t\_nodo*.
  - um inteiro que contém o número de elementos da lista.

A figura 1 ilustra esta estrutura de dados (caixas contendo barras na diagonal representam NULL):

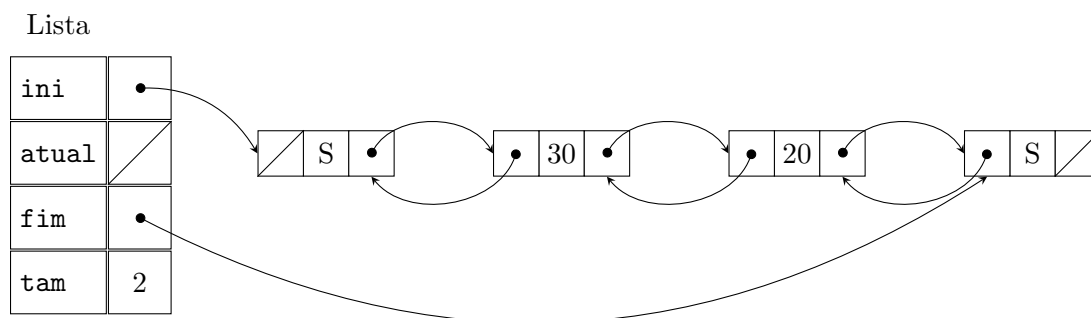


Figura 1: Exemplo da estrutura de dados *lista*.

Observe que, neste exemplo, o número de elementos é 2, a lista contém os números 30 e 20. Os nodos rotulados com **S** são as sentinelas do início e do final.

A figura 2 ilustra uma configuração da estrutura que representa uma *lista vazia*.

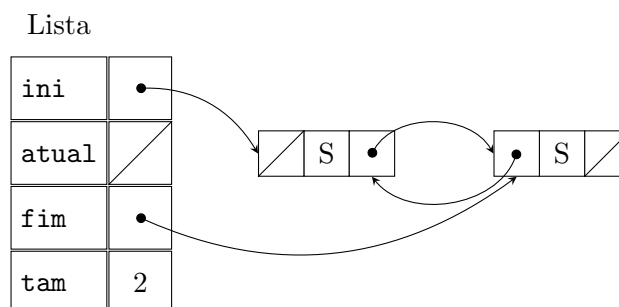


Figura 2: Uma lista vazia.

O ponteiro `atual` servirá para podermos percorrer a lista, sem precisarmos conhecer a estrutura dela. Vai fazer o papel de um índice que percorre um vetor.

### **ENTREGÁVEIS:**

1. Um arquivo `main.c` que leia listas, as imprima, concatene, copie, intercale e realize as demais funções da biblioteca `lib_lista_complementar.h`. O seu programa principal deve ser capaz de ler uma lista.
2. Os arquivos `lib_lista.c` e `lib_lista_complementar.c`.