# Guia de Apresentação por Integrante

Este guia foi feito para cada integrante explicar com segurança as funções sob sua autoria, justificando as estruturas de dados utilizadas e conduzindo uma pequena demonstração no programa.

Resumo das estruturas do projeto:

- Estudantes: lista simplesmente encadeada (inserção no início, busca por RA)
- Ocorrências (por estudante): lista duplamente encadeada circular com nó sentinela (navegação para frente e para trás; inserção em ordem por data)

Mapa de autoria (conforme comentários no código, redistribuição equilibrada):

- Carlos Vital: `estudantes_init`, `estudantes_adicionar`, `estudantes_listar`
- João Zanini: `estudantes_buscar`, `lista_criar`, `lista_destruir`, `lista_primeiro`, `lista_ultimo`
- João Manoel: `lista_inserir_ordenado` (+ helper `eh_mais_recente`), `comparar_datas`
- Cayo Vinícius: `estudantes_destruir`, `lista_remover_por_data`, `lista_contar_por_tipo`
- Felipe Laskos: `registrar_ocorrencia`, `listar_ocorrencias_por_ra`, `remover_ocorrencia`, `filtrar_por_tipo`, `parse_data`, `data_to_string`

Observação: `tipo_to_string` (inline em `lista.h`) dá suporte de exibição para todos.

---

## Carlos Vital

Papel no sistema

- Gerenciamento da lista de estudantes (lista simplesmente encadeada): inicialização, inclusão com validação de RA, busca, listagem e destruição.

Funções sob sua autoria

1. `void estudantes_init(ListaEstudantes *lista)` — arquivo: `estudantes.c`

- O que faz: zera a lista de estudantes (`head = NULL`, `tamanho = 0`).
- Por que lista simples? Inicialização trivial; atende ao requisito sem custo extra de estrutura.
- Complexidade: O(1). Edge cases: ponteiro válido (chamador garante).

2. `int estudantes_adicionar(ListaEstudantes *lista, int ra, const char *nome, const char *turma)` — `estudantes.c`

- O que faz: adiciona estudante no início se o RA não existir; cria a lista de ocorrências do aluno.
- Por que lista simples? Inserção O(1) no início; custo de busca O(n) é aceitável neste escopo.
- Validações: evita RA duplicado (`estudantes_buscar`); checa alocação e cria `lista_criar` para ocorrências.
- Complexidade: O(n) pela busca + O(1) para inserir.

3. `void estudantes_listar(ListaEstudantes *lista)` — `estudantes.c`

- O que faz: imprime RA, nome, turma e quantidade de ocorrências por estudante.
- Complexidade: O(n). Edge cases: lista vazia (imprime cabeçalho com 0).

5. `void estudantes_destruir(ListaEstudantes *lista)` — `estudantes.c`

- O que faz: libera todos os estudantes e destrói a lista de ocorrências de cada um (`lista_destruir`).
- Complexidade: O(n + soma k_i). Edge cases: lista já vazia.

Como apresentar (roteiro curto)

- Mostre a inclusão com RA único (tenta inserir o mesmo RA duas vezes para evidenciar a validação).
- Liste os estudantes e destaque a contagem de ocorrências.
- Explique por que a ordem dos estudantes não é requisito e por isso a inserção no início é suficiente.

Perguntas típicas e respostas

- Q: Por que não usar vetor para estudantes? R: Inserção no início e remoções implicariam deslocamentos; a lista simples é mais natural e suficiente.
- Q: Como garantir RA único? R: Busca linear antes de inserir (`estudantes_buscar`).

---

## João Zanini

Papel no sistema

- Busca de estudantes e criação/limpeza da lista de ocorrências e acesso direto às extremidades. Estrutura: lista duplamente encadeada circular com nó sentinela.

Funções sob sua autoria

1. `ListaOcorrencias* lista_criar()` — `lista.c`

- O que faz: cria lista dupla circular com sentinela; quando vazia, `sentinela->next == sentinela`.
- Por que com sentinela? Simplifica inserções/remoções nas pontas e evita checagens contra NULL.
- Complexidade: O(1) + alocações. Edge cases: falha de alocação (limpa e retorna NULL).

2. `void lista_destruir(ListaOcorrencias *lista)` — `lista.c`

- O que faz: percorre a partir do sentinela, libera cada nó, depois o sentinela e a estrutura da lista.
- Complexidade: O(k). Edge cases: `lista == NULL` é ignorado.

3. `OcorrenciaNode* lista_primeiro(ListaOcorrencias *lista)` — `lista.c`

- O que faz: retorna o nó mais recente (após o sentinela) ou NULL se vazia.
- Complexidade: O(1). Edge cases: `tamanho == 0`.

4. `OcorrenciaNode* lista_ultimo(ListaOcorrencias *lista)` — `lista.c`

- O que faz: retorna o nó mais antigo (antes do sentinela) ou NULL se vazia.
- Complexidade: O(1). Edge cases: `tamanho == 0`.

Como apresentar (roteiro curto)

- Demonstre a busca por RA funcionando (encontre um aluno existente e um inexistente).
- Explique o conceito de sentinela e a vantagem em listas circulares.
- Mostre que, com `lista_primeiro` e `lista_ultimo`, acessamos extremos em O(1) para navegação.

Perguntas típicas e respostas

- Q: Por que circular? R: Delimita início/fim pela própria comparação com o sentinela, sem NULL.
- Q: O que acontece se a lista estiver vazia? R: `tamanho == 0` e as funções retornam NULL de forma segura.

---

## João Manoel

Papel no sistema

- Regras de data, parsing e inserção ordenada das ocorrências, mantendo a ordem "mais recente primeiro".

Funções sob sua autoria

1. `OcorrenciaNode* lista_inserir_ordenado(ListaOcorrencias *lista, Data data, TipoOcorrencia tipo, const char *descricao)` — `lista.c`

- O que faz: insere mantendo ordem decrescente por data. Usa helper `eh_mais_recente(a, b)` (estático no mesmo arquivo).
- Por que inserir já ordenado? Evita ordenações posteriores; a navegação já sai na ordem temporal exigida.
- Complexidade: O(k) para achar posição + O(1) para ligar ponteiros. Edge cases: lista vazia (insere entre `s` e `s`).

2. `int comparar_datas(Data d1, Data d2)` — `lista.c`

- O que faz: compara datas (ano → mês → dia); >0 indica `d1` mais recente, 0 igual, <0 mais antiga (contrato no header).
- Uso: testes/validações/ordenação.
- Complexidade: O(1). Edge cases: datas iguais.

Observação: parsing e formatação de datas ficaram com Felipe na redistribuição.

Como apresentar (roteiro curto)

- Mostre duas ou três datas e explique a regra "mais recente primeiro".
- Demonstre que registrando datas fora de ordem, a listagem já sai ordenada.

Perguntas típicas e respostas

- Q: Como decide a posição de inserção? R: Compara ano, depois mês, depois dia via `eh_mais_recente`.
- Q: E se datas forem iguais? R: Permanece estável pelo percurso; se precisar desempatar, ampliar chave (ex.: tipo/descrição/id).

---

## Cayo Vinícius

Papel no sistema

- Destruição segura da lista de estudantes; remoção e contagem de ocorrências; filtros por tipo; operações integradas com estudantes.

Funções sob sua autoria

1. `int lista_remover_por_data(ListaOcorrencias *lista, Data data)` — `lista.c`

- O que faz: remove o primeiro nó cuja `data` casa (dia/mês/ano) e religa `prev/next`.
- Complexidade: O(k). Edge cases: lista vazia; data não encontrada.
- Observação: supõe unicidade de data por RA (documentado no README); se precisar, ampliar a chave de remoção.

2. `int lista_contar_por_tipo(ListaOcorrencias *lista, TipoOcorrencia tipo)` — `lista.c`

- O que faz: conta quantas ocorrências têm o `tipo` informado.
- Complexidade: O(k). Edge cases: `lista == NULL`.

Observação: funções de remoção e filtro no módulo de ocorrências foram alocadas ao Felipe.

Como apresentar (roteiro curto)

- Cadastre um aluno, registre 2–3 ocorrências com tipos variados, filtre por um tipo e mostre o agrupamento.
- Remova uma ocorrência por data e confirme que ela some da listagem.

Perguntas típicas e respostas

- Q: Como lida com início/fim da lista na remoção? R: Como é dupla com sentinela, basta religar `prev/next` sem ifs de nulo.
- Q: E se houver duas ocorrências com a mesma data? R: Remove a primeira encontrada; pode-se melhorar a chave.

---

## Felipe Laskos

Papel no sistema

- Integração entre cadastro de estudantes e lista de ocorrências: registro e navegação interativa por RA.

Funções sob sua autoria

1. `int registrar_ocorrencia(ListaEstudantes *estudantes, int ra, Data data, TipoOcorrencia tipo, const char *descricao)` — `ocorrencias.c`

- O que faz: busca o aluno por RA e insere ocorrência de forma ordenada na lista dele (`lista_inserir_ordenado`).
- Complexidade: O(n) para achar aluno + O(k) para inserir na lista.

2. `void listar_ocorrencias_por_ra(ListaEstudantes *estudantes, int ra)` — `ocorrencias.c`

- O que faz: inicia modo de navegação: `[n]` próxima, `[p]` anterior, `[q]` sair; mostra da mais recente para a mais antiga.
- Por que lista dupla circular? Permite navegar em ambas direções e detectar início/fim comparando com o sentinela.
- Complexidade: O(1) por passo de navegação; listar tudo é O(k).

3. `int parse_data(const char *str, Data *out)` — `lista.c`

- O que faz: valida e converte "DD/MM/AAAA" para `Data` (regras básicas de faixa).
- Complexidade: O(1). Edge cases: formato inválido (retorna 0), ponteiros nulos.

4. `char* data_to_string(Data d, char *buffer)` — `lista.c`

5. `int remover_ocorrencia(ListaEstudantes *estudantes, int ra, Data data)` — `ocorrencias.c`

- O que faz: acha o estudante por RA e delega a remoção para `lista_remover_por_data`.
- Complexidade: O(n) para encontrar o aluno + O(k) para remover na lista dele.

6. `void filtrar_por_tipo(ListaEstudantes *estudantes, TipoOcorrencia tipo)` — `ocorrencias.c`

- O que faz: percorre todos os alunos e imprime ocorrências do `tipo`, agrupando por estudante.
- Complexidade: O(n + soma k_i). Edge cases: nenhum item do tipo (apenas não imprime para aquele aluno).
- O que faz: converte `Data` para "DD/MM/AAAA" no `buffer` e retorna o buffer.
- Complexidade: O(1). Edge cases: buffer adequado (chamador garante).

Como apresentar (roteiro curto)

- Registre 3 ocorrências fora de ordem de data e mostre que a navegação já aparece ordenada.
- Mostre o `parse_data` falhando com uma data inválida e passando com uma válida.
- Navegue `n/p` e cite como o sentinela define início/fim.

Perguntas típicas e respostas

- Q: E se o RA não existir? R: A função valida e retorna falha (ou avisa na listagem).
- Q: Por que não armazenar em vetor por aluno? R: Inserções ordenadas e remoções no meio seriam O(k) com realocações; lista dupla evita isso e dá navegação natural.

---

## Dicas gerais de apresentação

- Comece situando o problema: “registrar ocorrências por aluno e navegar para frente e para trás”.
- Explique a escolha das listas:
  - Estudantes: lista simples basta (operações básicas e custo baixo de implementação).
  - Ocorrências: lista dupla circular com sentinela atende navegação bidirecional e facilita inserção/remoção.

- Cite big-O quando fizer sentido e aponte trade-offs (ex.: busca por RA é O(n), mas aceitável neste escopo).
- Durante a demo, use inputs pequenos e datas variadas para evidenciar a ordenação.

## Como demonstrar no programa (passo a passo)

1. Cadastrar estudante (opção 1): informe RA, nome e turma.
2. Registrar 3 ocorrências (opção 2) para esse RA, em datas variadas (ex.: 01/09/2025, 15/10/2025, 20/08/2025) e tipos diferentes.
3. Listar por RA (opção 3) e navegar com n/p.
4. Remover por RA+data (opção 4) e repetir a listagem para confirmar.
5. Filtrar por tipo (opção 5) para ver agrupamento.

## Glossário rápido

- Nó sentinela: nó especial que nunca contém dados de usuário; em listas circulares, ele aponta para si mesmo quando vazia e delimita início/fim.
- Circular: última ligação aponta de volta para o sentinela; não há ponteiros NULL internos.
- Mais recente primeiro: ordenação decrescente por (ano, mês, dia).