UNIVERSIDADE FEDERAL DE LAVRAS [Introdução aos Algoritmos] [Marluce]

Projeto Prático: Sistema de Cadastro em Arquivos com Ordenação

Tema: Cadastro de Frutas Tropicais

Alunos: Matheus Fellipe Araújo Marques

Henrique Araújo Furtini

Lavras, 28 de novembro de 2025

1. Introdução

O objetivo central deste trabalho foi implementar um sistema de cadastro completo utilizando a linguagem C++. A proposta consistiu em criar um software capaz de carregar dados de um arquivo CSV para um vetor alocado dinamicamente na memória. Isso permite ao usuário manipular esses registros livremente antes de decidir se deseja salvar as alterações no arquivo original.

Para a base de dados, escolhemos o tema "Frutas Tropicais". Essa escolha foi estratégica: definimos cinco campos que misturam dados numéricos e textuais, incluindo strings com espaços, o que atende aos requisitos de complexidade de leitura do projeto.

O foco do desenvolvimento esteve no domínio prático de conceitos essenciais da disciplina. Priorizamos a alocação dinâmica, o redimensionamento manual de vetores e a implementação "do zero" de algoritmos de busca e ordenação, abrindo mão propositalmente de facilidades como a biblioteca `<vector>` ou a função `sort()`. Todo o processo de codificação e testes ocorreu em ambiente Linux, seguindo as diretrizes especificadas.

2. Descrição das Estruturas e Lógica do Programa

A estrutura do sistema gira em torno de uma `struct` principal e de funções modulares que gerenciam um vetor dinâmico.

Estruturas de Dados: A base de tudo é a `struct Frutas`, que agrupa os dados de cada registro: *id* (int), *calorias* (int), *nome* (string), *continente* (string, permitindo espaços) e *coloração* (string). Para gerenciar isso na memória, utilizamos um ponteiro `Frutas* vetor`. O controle desse array dinâmico é feito através de duas variáveis auxiliares: `capacidade_total` (tamanho alocado) e `n_elementos_vetor` (posições ocupadas). Além disso, implementamos uma flag de controle (`string ordenadoPor`) para validar se a busca binária pode ou não ser executada.

Lógica do Programa:

1. **Inicialização:** Tudo começa no `main`, que aloca o vetor dinâmico com uma capacidade inicial de 40 posições e zera o contador de elementos.
2. **Carregamento Inteligente:** A função `carregar` abre o arquivo CSV. Adotamos a estratégia de ler a linha inteira primeiro e processá-la via `stringstream`. Isso foi crucial para evitar erros de leitura entre Windows e Linux (devido aos caracteres de fim de linha diferentes), garantindo que a conversão de strings para inteiros funcionasse perfeitamente.
3. **Redimensionamento Automático:** Se o vetor encher (seja durante o carregamento ou na inserção manual), a função `adicionar_elementos` entra em ação. Ela aloca um novo vetor com 10 posições extras, copia os dados antigos, apaga o vetor velho da memória e atualiza os ponteiros.
4. **Menu de Controle:** O fluxo do programa é mantido por um loop `do-while` no `main`, onde um `switch-case` direciona o usuário para a funcionalidade desejada.
5. **Inserção de Dados:** Ao adicionar uma nova fruta, o sistema verifica se há espaço (redimensionando se necessário). O dado é inserido no final e, importante: a flag de ordenação é resetada para "nenhum", pois a nova inserção pode quebrar a ordem alfabética ou numérica existente.
6. **Remoção Física:** Para remover, buscamos o item pelo ID. Ao encontrá-lo, realizamos uma remoção real: todos os elementos subsequentes são deslocados uma posição para a esquerda ("puxados"), sobrescrevendo o item removido e garantindo que não existam buracos no vetor.
7. **Ordenação (QuickSort):** O usuário pode ordenar por ID ou Nome. Implementamos o algoritmo *QuickSort* manualmente para reorganizar o vetor em memória, atualizando a flag `ordenadoPor` ao final do processo.
8. **Busca Binária:** Antes de buscar, o sistema verifica se o vetor está ordenado pelo critério correto. Se estiver, ele executa a *Busca Binária* (alta eficiência); caso contrário, avisa o usuário que é necessário ordenar os dados antes.
9. **Listagem:** A função `mostrarDados` percorre o vetor e imprime os elementos na tela na ordem em que estão atualmente na memória.
10. **Gravação (Persistência):** A função `gravarArquivo` abre o CSV em modo de escrita, sobrescrevendo o anterior. Ela recria o cabeçalho e salva cada registro do vetor linha por linha, consolidando as alterações.

3. Ordem dos Dados no Arquivo

O arquivo `frutastropicais.csv` serve como persistência dos dados. A primeira linha atua como cabeçalho, e as subsequentes seguem o padrão: `#ID, Nome, Continente, Calorias, Coloracao` Exemplo: `1, Banana, America do Sul, 89, Amarela`

4. Acertos e Erros Durante o Desenvolvimento

A construção deste projeto foi fundamental para solidificar a teoria, mas a prática trouxe desafios que exigiram bastante *debugging*.

Acertos: A decisão de usar o *QuickSort* e a *Busca Binária* foi acertada. Conseguimos notar a eficiência na pesquisa de dados quando o vetor estava ordenado. Além disso, a modularização do código (separando bem as responsabilidades de cada função) salvou muito tempo na hora de encontrar erros, mantendo o `main` limpo. O gerenciamento de memória, que era nossa maior preocupação inicial, funcionou de forma estável.

Erros e Desafios:

- **O "Bug de 0 Frutas" (Portabilidade):** Este foi o erro mais complexo. O código compilava, mas dizia "0 frutas carregadas" quando rodava no Windows. Descobrimos que o Windows usa `\r\n` para quebra de linha, enquanto o Linux usa apenas `\n`. O `getline` estava deixando o caractere `\r` "sujo" no buffer, o que fazia a conversão de inteiros (`stoi`) falhar na linha seguinte.
    - *Solução:* Refatoramos a função `carregar` para ler a linha inteira primeiro e tratar o buffer na memória via `stringstream`. Isso tornou o código robusto para qualquer sistema.
- **Erro de Caminho (Path):** Perdemos algum tempo com o erro de "Arquivo não encontrado", simplesmente porque a IDE rodava o executável em uma subpasta diferente de onde estava o CSV. Ajustar o diretório de trabalho resolveu a questão.
- **Buffer do teclado:** Tivemos um problema onde o programa entrava em loop infinito se o usuário digitasse uma letra num campo numérico. O `cin` entrava em estado de erro. Resolvemos isso implementando uma limpeza de buffer e tratamento de exceção simples.

5. Conclusão

Este projeto foi mais do que apenas um exercício de codificação; foi uma experiência real de engenharia de software em pequena escala. Conseguimos entregar um sistema 100% funcional que cumpre todos os requisitos, desde a manipulação de ponteiros até a implementação de algoritmos complexos.

O maior aprendizado, contudo, veio dos erros. Entender como o sistema operacional lida com arquivos e como gerenciar a memória manualmente nos deu uma visão muito mais clara do que acontece "por baixo do capô" da linguagem C++. O resultado final é um programa robusto, portável e eficiente.