UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE (UFRN)
INSTITUTO METRÓPOLE DIGITAL (IMD)

BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO (BTI)

Disciplina: Introdução a Técnicas de Programação (ITP) - 2025.2

Unidade 2: Relatório Técnico

Projeto: Prontuário Eletrônico de Pacientes

Aluno: Anderson Pedro do Nascimento

1. Introdução

Contexto e Objetivos

Este relatório documenta a evolução do projeto "Prontuário Eletrônico de Pacientes", atualizado para atender aos requisitos da Unidade 2 da disciplina de Introdução a Técnicas de Programação. O objetivo central desta etapa foi refatorar o sistema desenvolvido na Unidade 1, que operava com memória estática e limitada, para incorporar conceitos avançados da linguagem C, com foco principal no **gerenciamento dinâmico de memória** e manipulação de estruturas de dados complexas.

Funcionalidades Implementadas

A versão 2.0 do sistema introduz a capacidade de cadastrar um número ilimitado de pacientes (restrito apenas pela memória do hardware), graças à implementação de alocação dinâmica. Além disso, foram adicionadas funcionalidades de busca textual por nome (Strings), verificação automática de duplicidade de cadastros (Loops Aninhados) e um módulo de triagem de saúde mental baseado no questionário PHQ-9 (Matrizes).

---

2. Metodologia

Ferramentas e Abordagem

O projeto foi desenvolvido utilizando a linguagem C (padrão C99), compilado com GCC e editado no Visual Studio Code em ambiente Windows. A abordagem de desenvolvimento foi incremental: partindo do código da Unidade 1, as estruturas estáticas (vetores fixos) foram substituídas gradualmente por ponteiros conforme passado em aula, seguidas pela implementação das novas funcionalidades.

Organização do Código

A organização modular foi mantida e aprimorada. A função main atua como controladora de fluxo e gerenciadora do ciclo de vida da memória (alocação inicial e liberação final). As funcionalidades específicas foram encapsuladas em funções dedicadas (cadastrarPacientes, buscarPacientePorNome, verificacaoDuplicidade), garantindo a legibilidade e facilitando a manutenção. O histórico de evolução do código foi documentado através de commits frequentes no repositório Git.

---

3. Análise do Código (Novos Conteúdos da U2)

3.1. Alocação Dinâmica e Ponteiros

A alteração estrutural mais profunda foi a substituição dos vetores globais (ex: char nomes[50][100]) por ponteiros duplos (ex: char **nomes). Essa mudança exigiu o uso intensivo de ponteiros para manipular os endereços de memória onde os dados estão armazenados.

- **Gerenciamento de Memória:** A estratégia adotada utiliza a função malloc para a alocação inicial e realloc para a expansão. Na função cadastrarPacientes, o sistema verifica se o limite atual (capacidadeAtual) foi

atingido. Se sim, a capacidade é dobrada (* 2), e os vetores principais são realocados.

- **Prevenção de Memory Leaks:** Para garantir que não haja vazamentos de memória, foi implementada uma rotina de limpeza ao final da execução. Um laço percorre todos os pacientes cadastrados liberando as alocações individuais (strings e linhas da matriz) com free(), seguido pela liberação dos ponteiros.

3.2. Matrizes

O conceito de matrizes foi implementado no módulo do inventário PHQ-9. Foi criada uma estrutura dinâmica, que funciona como uma matriz onde cada linha representa um paciente e as colunas armazenam as respostas (qualificação: inteiros de 0 a 3) para as 9 perguntas do questionário. A manipulação é feita através de laços for para coletar as respostas no cadastro e para calcular a pontuação total na exibição dos detalhes.

3.3. Strings

A manipulação de strings foi expandida para além da entrada/saída básica. Na nova função buscarPacientePorNome, foi utilizada a função strstr() da biblioteca <string.h>. Ela permite verificar se o termo digitado pelo usuário é uma substring do nome de algum paciente cadastrado, possibilitando uma busca parcial eficiente (ex: buscar "Silva" e encontrar "Maria Silva").

3.4. Estruturas de Repetição Aninhadas

Para aplicar o conceito de loops aninhados, foi desenvolvida a função verificacaoDuplicidade. O algoritmo utiliza dois laços for: um externo (índice i) e um interno (índice j, iniciando em i + 1). Isso permite comparar cada paciente com todos os subsequentes na lista, identificando nomes idênticos através da função strcmp() e alertando o usuário sobre redundâncias.

---

4. Dificuldades e Soluções

Desafios Técnicos

O principal desafio foi a gestão correta da lógica de realocação (realloc). Durante o desenvolvimento, um erro de lógica fazia com que a alocação dos dados individuais do novo paciente (malloc para nome/sintomas) ficasse dentro do bloco condicional if de expansão da memória. Isso impedia o cadastro de pacientes quando a memória *não* precisava ser expandida.

Superação e Aprendizados

O problema foi superado reestruturando a função cadastrarPacientes para separar claramente a verificação de capacidade da alocação do novo registro. Outra dificuldade foi a compreensão da sintaxe de ponteiros para ponteiros (char **), superada através do estudo prático de como as strings são armazenadas em C. O uso de getchar() para limpeza de buffer continuou sendo uma prática necessária para evitar erros de leitura.

---

5. Conclusão

A Unidade 2 representou um salto qualitativo no projeto. A transição para memória dinâmica transformou o Prontuário Eletrônico em uma aplicação escalável e mais profissional. O domínio sobre ponteiros e gerenciamento manual de memória, embora complexo, provou-se essencial para criar um software eficiente em C. Para uma futura atualização tendo em vista que a unidade 3 será prova então não será possível uma mudança no momento,mas o passo natural será a implementação de persistência de dados em arquivos para evitar a perda de informações ao fechar o programa.

---

6. Respostas às Perguntas Orientadoras

Gerais:

1. **Conceitos da U2 aplicados:** Foram aplicados alocação dinâmica (malloc, realloc, free), ponteiros e ponteiros duplos, matrizes dinâmicas, manipulação avançada de strings (strstr) e estruturas de repetição aninhadas.
2. **Organização e Manutenção:** A divisão em funções isolou a complexidade da alocação dinâmica na função de cadastro, permitindo que o resto do código (como o menu) permaneça limpo e fácil de manter.
3. **Principais Desafios:** O correto uso de realloc sem perder os dados anteriores e a garantia de liberar toda a memória alocada para evitar *leaks*.

Específicas da U2: 4. **Implementação de Matrizes:** As matrizes foram implementadas dinamicamente como vetores de ponteiros (int **), onde cada ponteiro aponta para um vetor de inteiros, permitindo linhas de tamanho flexível se necessário. 5. **Estratégia de Memória:** A estratégia adotada foi o crescimento exponencial (dobrar a capacidade) para minimizar o custo computacional de chamadas frequentes ao realloc. 6. **Garantia contra Vazamentos:** A ausência de vazamentos é garantida pela execução sistemática de free() em todas as estruturas alocadas, realizada no encerramento do programa. 7. **Vantagens da Alocação Dinâmica:** Removeu o limite artificial de 50 pacientes e permitiu que o programa consuma apenas a memória necessária para os dados existentes. 8. **Ponteiros e Eficiência:** O uso de ponteiros permitiu manipular grandes volumes de dados (como o banco de dados de pacientes) sem a necessidade de copiar arrays inteiros na memória, aumentando a performance.