UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO GABRIEL FROTA DE MOURA SANTOS

JOÃO PEDRO CAJADO DE LIMA GOMES

PEDRO HENRIQUE RIBEIRO CARVALHO

PEDRO PAULO RODRIGUESFILHO

NICOLAS GOMES HARNISCH

TRABALHO FINAL: GESTÃO DE FUNCIONÁRIOS

JOÃO GABRIEL FROTA DE MOURA SANTOS
JOÃO PEDRO CAJADO DE LIMA GOMES
PEDRO HENRIQUE RIBEIRO CARVALHO
PEDRO PAULO RODRIGUES FILHO
NICOLAS GOMES HARNISCH

TRABALHO FINAL: GESTÃO DE FUNCIONÁRIOS

Dissertação apresentada ao Curso de Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisitada pela disciplina de Laboratório de Programação.

Prof. Me. José Robertty de Freitas Costa
Universidade Federal do Ceará (UFC)

RESUMO

Este trabalho detalha o desenvolvimento de um sistema para gerenciamento de dados de uma empresa, implementado na linguagem de programação C. O projeto tem como finalidade principal modelar e manipular informações de funcionários, cargos e departamentos, demonstrando conceitos fundamentais de estrutura de dados e alocação dinâmica de memória.

A metodologia empregada baseia-se na utilização de structs para a modelagem dos objetos, vetores dinâmicos para o armazenamento em tempo de execução e arquivos de texto (.txt) para garantir a persistência dos dados. O sistema implementa as funcionalidades essenciais de um CRUD (Create, Read, Update, Delete) para cada uma das entidades, permitindo a inserção, busca, atualização e exclusão de registros. Adicionalmente, o sistema gera relatórios gerenciais, incluindo um relatório de alocação de funcionários por departamento e um relatório salarial com estatísticas como soma, média, maior e menor salário. Conclui-se que o sistema atende aos requisitos propostos, oferecendo uma solução robusta e eficiente para a gestão de dados com otimização do uso de memória.

Palavras-chave: gestão de dados; linguagem c; alocação dinâmica de memória; persistência de dados; estruturas de dados.

SUMÁRIO

1 INTRODUÇÃO

A gestão eficiente de informações é um pilar fundamental para o funcionamento de qualquer organização. Em um cenário empresarial, o controle sobre dados de funcionários, sua alocação em departamentos e a estrutura de cargos é essencial para a tomada de decisões estratégicas e para a organização administrativa. A necessidade de sistemas robustos, eficientes e customizáveis para essa finalidade motivou o desenvolvimento do presente trabalho.

O objetivo deste projeto é a concepção e implementação de um sistema de Gerenciamento de Funcionários em linguagem C. A escolha da linguagem C se justifica pela sua performance e pelo controle de baixo nível sobre a memória, permitindo a exploração de conceitos avançados de programação, como a alocação dinâmica e o gerenciamento explícito de recursos computacionais.

A pesquisa se concentra na modelagem de três entidades centrais do mundo real: o **Funcionário**, o **Cargo** e o **Departamento**. Para tal, foram utilizadas estruturas de dados (structs) que encapsulam os atributos de cada entidade. O sistema armazena os dados em vetores dinâmicos, cuja memória é gerenciada de forma inteligente para otimizar o uso de recursos, e garante a persistência das informações através da gravação e leitura de arquivos de texto.

Este relatório descreverá em detalhes a arquitetura do sistema, as estruturas de dados utilizadas, as funcionalidades implementadas — incluindo inserção, busca, atualização e exclusão de registros (CRUD) —, as estratégias de gerenciamento de memória e persistência, e os relatórios gerenciais desenvolvidos.

2 MODELAGEM DOS DADOS E ESTRUTURAS UTILIZADAS

A base para a construção do sistema foi a correta modelagem das entidades do domínio do problema. Para representar os objetos do mundo real, foram utilizadas structs, um recurso da linguagem C que permite agrupar diferentes tipos de dados em uma única unidade lógica.

2.1 Estrutura de Dados (Structs)

Foram definidas três structs principais para representar as entidades do sistema: Funcionário, Departamento e Cargo.

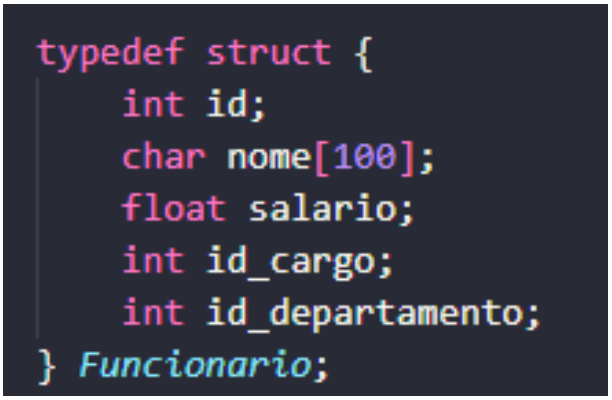
- **Cargo**: Representa uma posição ou função que pode ser ocupada na empresa. É a estrutura mais simples, contendo um identificador único e o nome do cargo.

```c
typedef struct {
    int id;
    char nome[50];
} Cargo;
```

- **Departamento:** Representa uma seção ou divisão da empresa. Assim como o cargo, possui um identificador e um nome.

```c
typedef struct {
    int id;
    char nome[100];
} Departamento;
```

- **Funcionário:** É a entidade central e mais complexa, agregando informações pessoais e profissionais. Contém um ID, nome, salário e as chaves estrangeiras (id_cargo e id_departamento) que estabelecem a ligação com as outras entidades.



2.1.1 Relacionamento entre as entidades

A arquitetura de dados do sistema foi projetada para ser normalizada, uma abordagem inspirada em bancos de dados relacionais para minimizar a redundância e melhorar a integridade dos dados. O relacionamento entre as entidades "Funcionário, Cargo e Departamento" não é estabelecido pela duplicação de estruturas, mas sim pelo uso de chaves de identificação, de forma análoga ao conceito de chaves primárias e estrangeiras.

A struct Funcionário atua como a entidade central que conecta as demais. Ela contém os campos id_cargo e id_departamento.

- **Funcionario.id_cargo**: Atua como uma chave estrangeira que aponta para o identificador único (id) de um registro no vetor de Cargo.
- **Funcionario.id_departamento**: Da mesma forma, serve como chave estrangeira para o id de um registro no vetor de Departamento.

Essa abordagem de referenciamento por ID, em vez de aninhar as structs Cargo e Departamento inteiras dentro da struct Funcionário, traz vantagens significativas e implica um fluxo de trabalho específico para a consulta de dados.

3 FUNCIONALIDADES DO SISTEMA (CRUD)

O sistema implementa as quatro operações fundamentais de gerenciamento de dados, conhecidas pelo acrônimo CRUD (Create, Read, Update, Delete), para cada uma das três entidades.

3.1 Inserção de Dados

A funcionalidade de inserção permite adicionar novos registros de cargos, departamentos ou funcionários. O sistema solicita ao usuário que digite os dados de cada campo. A inserção é realizada ao final do vetor dinâmico correspondente, e o contador de elementos é incrementado. O gerenciamento de memória associado a esta operação é detalhado na seção 4.1.

```c
int inserir_cargo(Cargo **vetor, int *tamanho, int *capacidade) {
    if (*capacidade == 0) {···
    else if (*tamanho >= *capacidade * 0.75) {···

    if (*vetor == NULL) {···

    Cargo novo_cargo;

    printf("Digite o ID do cargo: ");
    scanf("%d", &novo_cargo.id);

    printf("Digite o Nome do cargo: ");
    scanf("%s", novo_cargo.nome);

    (*vetor)[*tamanho] = novo_cargo;
    (*tamanho)++;

    printf("Cargo inserido com sucesso!\n");
    return 1;
}
```

3.1.1 Busca de Dados

As funções de busca permitem ao usuário localizar um registro específico. A busca por “Cargo e Departamento” oferece a opção de procurar por ID ou por nome. A busca de “Funcionário” é implementada por ID. A função percorre o vetor correspondente e, ao encontrar o registro com o identificador correspondente, exibe todas as suas informações na tela.

3.1.2.1 Atualização de Dados

A atualização de um registro é realizada em duas etapas: primeiro, o usuário informa o ID do registro que deseja modificar; em seguida, o sistema o localiza através de uma função de busca. Se encontrado, o sistema solicita os novos dados ao usuário e os sobrescreve diretamente na posição do vetor onde o registro se encontra.

```
int atualizar_cargo(Cargo *vetor, int tamanho) {
    if (tamanho == 0) {···

    int id_procurado;
    printf("\n--- Atualizar Cargo ---\n");
    printf("Digite o ID do cargo que deseja atualizar: ");
    scanf("%d", &id_procurado);

    int indice_encontrado = -1;
    for (int i = 0; i < tamanho; i++) {
        if (vetor[i].id == id_procurado) {
            indice_encontrado = i;
            break;
        }
    }

    if (indice_encontrado != -1) {
        printf("Cargo encontrado: %s\n", vetor[indice_encontrado].nome);

        printf("Digite o novo nome para o cargo: ");
        scanf("%s", vetor[indice_encontrado].nome);

        printf("Cargo atualizado com sucesso!\n");
        return 1;
    } else {···
}
```

3.1.2.1.1 Exclusão de Dados

Para excluir um registro, o sistema primeiro localiza o item pelo ID fornecido. Uma vez encontrado o índice do elemento a ser removido, a exclusão é efetuada através da técnica de deslocamento: todos os elementos posteriores à posição do item excluído são movidos uma posição para trás no vetor. Por fim, o contador de tamanho do vetor é decrementado.

4 GERENCIAMENTO DE MEMÓRIA E PERSISTÊNCIA DE DADOS

Dois dos requisitos mais importantes do projeto foram o gerenciamento eficiente da memória em tempo de execução e a capacidade de salvar e carregar os dados para garantir sua persistência entre sessões de uso do programa.

4.1 Gerenciamento Dinâmico de Memória

Para evitar o desperdício de memória com vetores estáticos de tamanho fixo, optou-se pelo uso de alocação dinâmica. Os vetores de “Funcionário, Cargo e Departamento” são ponteiros que têm sua memória alocada e gerenciada em tempo de execução.

- **Crescimento (Growth):** Durante a inserção, antes de adicionar um novo elemento, o sistema verifica se o vetor atingiu sua capacidade máxima. Em caso afirmativo, a memória alocada é dobrada utilizando a função realloc. Essa estratégia de crescimento exponencial garante que as operações de realocação sejam infrequentes, otimizando a performance.

```c
if (*capacidade == 0) {
    *capacidade = 4;
    *vetor = malloc(*capacidade * sizeof(Funcionario));
}
else if (*tamanho >= *capacidade) {
    *capacidade *= 2;
    *vetor = realloc(*vetor, *capacidade * sizeof(Funcionario));
}
```

- **Redução (Shrink):** Durante a exclusão, após remover um elemento, o sistema verifica se o nível de ocupação do vetor caiu para um limiar baixo (neste caso, 25% da capacidade). Se essa condição for atendida, o vetor é realocado para a metade de seu tamanho atual, liberando memória não utilizada de volta para o sistema operacional.

```c
if (*tamanho > 0 && *tamanho <= *capacidade * 0.25) {
    int nova_capacidade = *capacidade / 2;

    if (nova_capacidade < 10) {
        nova_capacidade = 10;
    }
```

- **Liberação:** Ao final da execução do programa, toda a memória alocada para os vetores dinâmicos é liberada explicitamente através da função “free”, prevenindo vazamentos de memória.

4.1.1 Persistência de Dados em Arquivos

Para que os dados inseridos não sejam perdidos ao fechar o programa, foram implementadas rotinas de salvamento e carregamento. As funções “salvar_dados” percorrem os vetores em memória e escrevem cada registro em um arquivo de texto (.txt), utilizando um formato delimitado por ponto e vírgula para separar os campos. A função “carregar_dados” (comentada no main.c, mas parte do escopo do projeto) realizaria o processo inverso: leria os dados dos arquivos na inicialização do programa e os carregaria nos vetores dinâmicos.

```c
void salvarFuncionario(){


    FILE * arquivoFun;
    arquivoFun = fopen("Funcionario.txt", "w");

    if(arquivoFun == NULL){
        printf("Erro arquivo nao encontrado!\n");
        return;
    }

    for(int i = 0; i < quant_funcionarios; i++){
        fprintf(arquivoFun, "%d;%s;%.2f;%d;%d\n",
            funcionarios[i]->id,
            funcionarios[i]->nome,
            funcionarios[i]->salario,
            funcionarios[i]->id_cargo,
            funcionarios[i]->id_departamento);
    }

    fclose(arquivoFun);
}
```

5 RELATÓRIOS GERENCIAIS

O sistema é capaz de gerar dois tipos de relatórios analíticos, que consolidam os dados cadastrados para fornecer informações úteis para a gestão.

5.1 Relatório por Departamento

Esta funcionalidade permite visualizar todos os funcionários alocados em um departamento específico. O sistema primeiro exibe uma lista de todos os departamentos cadastrados. O usuário então digita o ID do departamento desejado. Em seguida, o programa percorre o vetor de funcionários e imprime os dados (ID, nome e salário) de todos aqueles cujo “id_departamento” corresponde ao ID fornecido.

5.1.1 Relatório Salarial

O relatório salarial oferece uma visão geral sobre a folha de pagamento da empresa. Ele processa o vetor de todos os funcionários cadastrados e calcula as seguintes métricas:

- **Total de Funcionários:** A contagem total de registros.
- **Soma de Todos os Salários:** O valor total da folha de pagamento.
- **Média Salarial:** A média aritmética dos salários.
- **Maior Salário:** O valor do salário mais alto registrado.
- **Menor Salário:** O valor do salário mais baixo registrado.

Essas informações são então exibidas de forma clara e organizada para o usuário.

```
float soma_salarios = 0.0;
float maior_salario = 0.0;
float menor_salario = -1.0;

for (int i = 0; i < quant_funcionarios; i++) {
    float salario_atual = funcionarios[i].salario;

    soma_salarios += salario_atual;

    if (salario_atual > maior_salario) {
        maior_salario = salario_atual;
    }

    if (menor_salario == -1.0 || salario_atual < menor_salario) {
        menor_salario = salario_atual;
    }
}

float media_salarial = (float)soma_salarios / quant_funcionarios;
```

6 DEMONSTRAÇÕES DE USO

A seguir, são apresentadas simulações de tela para demonstrar o fluxo de utilização do sistema.

6.1 Menu Principal

Ao ser iniciado, o sistema apresenta um menu principal com todas as funcionalidades disponíveis para o usuário.

```
--- MENU DE GESTAO ---
0. Sair
1. Inserir
2. Atualizar
3. Excluir
4. Buscar
5. Relatorio por Departamento
6. Relatorio de Salario
Digite sua opcao:
```

6.1.1 Exemplo de Inserção

O usuário escolhe a opção 1 (Inserir) e, em seguida, a opção 1 (Funcionário). O sistema solicita os dados.

```
--- Inserir ---
1. Funcionario
2. Departamento
3. Cargo
0. Voltar
Digite: 1

    Inserir Novo Funcionario
Digite o ID: 101
Digite o Nome: Ana Silva
Digite o salario: 4500.00
Digite o ID do cargo: 10
Digite o ID do departamento: 1
```

6.1.2.1 Exemplo de Busca

O usuário escolhe a opção 4 (Buscar) e, em seguida, a opção 1 (Funcionário). O sistema pede o ID a ser buscado.

```
--- Buscar ---
1. Funcionario
2. Departamento
3. Cargo
0. Voltar
Digite: 1

Digite o ID do Funcionario a ser buscado: 101

--- Funcionario Encontrado ---
ID: 101
Nome: Ana Silva
Salario: R$4500.00
ID Cargo: 10
ID Departamento: 1
```

6.1.2.1.1 Exemplo de Relatório

O usuário escolhe a opção 6 para gerar o relatório salarial.

```
Digite sua opcao: 6

--- Relatorio de Salarios ---
Total de Funcionarios: 1
Soma de todos os salarios (Folha salarial): R$4500.00
Media Salarial: R$4500.00
Maior Salario: R$4500.00
Menor Salario: R$4500.00
```

7 CONCLUSÃO

O desenvolvimento do sistema de Gerenciamento de Funcionários culminou em uma aplicação em linguagem C que não apenas atende, mas valida com sucesso todos os objetivos primários estabelecidos no início deste trabalho. O projeto se propôs a criar uma solução robusta para a manipulação de dados empresariais, com um foco particular na aplicação de conceitos fundamentais da ciência da computação, como a modelagem com estruturas de dados, o gerenciamento explícito e dinâmico da memória e a persistência de informações em arquivos.

“Funcionário, Cargo e Departamento” através de structs e o relacionamento entre elas por meio de chaves de identificação provou ser uma solução eficiente e normalizada, evitando a redundância de dados e otimizando o uso de memória. A implementação completa das quatro operações essenciais — Inserir, Atualizar, Excluir e Buscar (CRUD) — para cada entidade, confere ao sistema a funcionalidade completa esperada de uma plataforma de gestão de dados. Um dos pilares técnicos do projeto, o gerenciamento dinâmico de memória, foi implementado com sucesso, utilizando malloc para a alocação inicial e realloc para o crescimento e a redução inteligente dos vetores, adaptando o consumo de recursos à demanda de dados em tempo de execução e prevenindo o desperdício. A capacidade de salvar os dados em arquivos de texto garante a durabilidade e a integridade das informações entre as sessões de uso, um requisito indispensável para a aplicação prática do software.

O principal desafio enfrentado foi inerente à própria linguagem C: a necessidade de gerenciar manualmente a memória e garantir a integridade referencial dos dados — responsabilidades que em sistemas de banco de dados são abstraídas. Essa complexidade, no entanto, representou a mais valiosa oportunidade de aprendizado, proporcionando uma compreensão profunda sobre o funcionamento de ponteiros, alocação de memória e a lógica por trás de operações de banco de dados.

Conclui-se, portanto, que este trabalho representa mais do que a entrega de um software funcional; ele é a materialização de um estudo aprofundado e prático sobre os pilares da programação estruturada e do gerenciamento de recursos em C, resultando em uma solução de software coesa e um valioso exercício acadêmico.

8 REFERÊNCIAS

CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. 3. ed. Rio de Janeiro: Elsevier, 2012.

DEITEL, Paul; DEITEL, Harvey. **C: como programar**. 6. ed. São Paulo: Pearson Prentice Hall, 2011.

KERNIGHAN, Brian W.; RITCHIE, Dennis M. **The C Programming Language**. 2. ed. Englewood Cliffs: Prentice Hall, 1988.

MARTIN, Robert C. **Arquitetura limpa: o guia do artesão para estrutura e design de software**. São Paulo: Alta Books, 2019.