Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Unidade Curricular: Algoritmos e Programação

Relatório Relativo ao Trabalho Prático

Tema: Gestão de Estudantes

Realizado por: Hugo Afonso – 30032

Mateus Silva – 29989

Viseu, 2025

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Relatório relativo ao Trabalho Prático

Curso de Licenciatura em Engenharia Informática

Unidade Curricular de Algoritmos e Programação

Gestão de Alunos

Ano Letivo 2024/25

Viseu, 2025

RESUMO

No âmbito da unidade curricular de Análise e Programação, foi proposto o desenvolvimento de um projeto que visa a criação de uma aplicação para gestão de dados escolares. O projeto baseia-se em uma base de dados contendo informações detalhadas sobre os alunos, como código escolar, nome, data de nascimento, nacionalidade, número de matrículas, ECTS acumulados, ano do curso e média atual. A aplicação tem como objetivo principal permitir o armazenamento eficiente dessas informações, bem como a edição e pesquisa otimizada dos dados de todos os alunos a qualquer momento. O desenvolvimento deste projeto envolveu a aplicação de técnicas de análise e programação para garantir uma solução funcional e intuitiva focada em facilitar a gestão e manipulação dos dados escolares de forma eficaz.

Palavras-Chave: **Gestão de Dados, Base de Dados, Aplicação Escolar, Edição de informações, Pesquisa Otimizada.**

ÍNDICE

Índice de Figuras

1. Introdução

No âmbito da unidade curricular de Análise e Programação, foi proposto o desenvolvimento de um projeto com o objetivo de criar uma aplicação para a gestão de dados escolares. Este projeto parte de uma base de dados com informações sobre 2000 alunos, onde temos disponíveis dados pessoais, como o código escolar, nome, data de nascimento e nacionalidade, e informações escolares, como o número de matrículas, ECTS acumulados, ano do curso e média atual.

O objetivo principal é desenvolver uma aplicação que permita armazenar essas informações de forma organizada, além de possibilitar a edição e a pesquisa dos dados de maneira eficiente e otimizada. A aplicação deve responder às necessidades de uma gestão escolar eficiente, proporcionando funcionalidades intuitivas que facilitem a consulta e a atualização dos dados dos alunos.

Durante o desenvolvimento, destacou-se o uso de memória dinâmica, permitindo que os arrays de dados se ajustassem automaticamente ao número de registos necessários. Esta abordagem garantiu eficiência no uso de recursos e escalabilidade do programa, adaptando-se a diferentes volumes de dados sem desperdício de memória.

Ao longo do desenvolvimento do projeto, foram aplicados os conceitos fundamentais de análise e programação, garantindo que a solução apresentada seja funcional, prática e capaz de lidar com as exigências do cenário escolar.

Este documento está organizado em cinco capítulos que se seguem a esta introdução.

No segundo capítulo propõe-se a Planificação do Projeto.

Segue-se o capítulo três, onde abordamos a Resolução do Programa.

O capítulo quatro aborda a Testagem do Programa.

Termina-se com o capítulo cinco, onde se apresentam as conclusões deste trabalho.

2. Planificação

A planificação do projeto foi cuidadosamente estruturada com o objetivo de assegurar o cumprimento rigoroso dos requisitos previamente definidos. Para alcançar este propósito, foram seguidas, de forma organizada e sistemática, as seguintes etapas principais:

2.1. Estruturação de Dados

As estruturas de dados utilizadas no projeto foram pensadas para otimizar o desempenho e a organização da informação:

2.1.1. Estrutura Estudante

A estruturação dos dados no programa foi realizada utilizando as structs, que permitem organizar as informações de forma clara e eficiente. Como por exemplo, a utilização da **- Struct** Estudante, onde foi desenhada para armazenar informações pessoais dos alunos, como nome, data de nascimento e nacionalidade. A utilização de ponteiros para as strings permite alocação dinâmica, garantindo que a memória utilizada seja proporcional ao tamanho real dos dados.

```
61  //Struct para tratar todos os dados relativos aos estudantes
62  typedef struct estudante {
63      int codigo; //int para prevenir, caso o código tenha, imagine-se, 6 digitos
64      char * nome; //Declaramos um ponteiro para posteriormente alocar memória dinamicamente consoante o tamanho do nome
65      Data nascimento;
66      char * nacionalidade; //Criamos um array do tipo nacionalidade, que irá conter todas as nacionalidades
67  }Estudante;
```

Figura 1- Struct Estudante

2.1.2. Estrutura Dados

A **- Struct** Dados é utilizada para armazenar informações académicas dos alunos, como número de matrículas, créditos ECTS acumulados, ano do curso e média atual. Além disso, inclui indicadores sobre a situação académica, como risco de prescrição e se o aluno é finalista.

```c
typedef struct dados_escolares {
    int codigo;
    short matriculas;
    short ects;
    short ano_atual;
    float media_atual;
    char prescrever;
    char finalista;
}Dados;
```

Figura 2- Struct Dados

2.1.3. Estrutura Uni

A - Struct Uni é a estrutura central do programa, responsável por agrupar e organizar todos os dados relacionados à gestão escolar. Esta estrutura combina os arrays de alunos e dados académicos, além de incluir informações auxiliares, como estatísticas globais e informações adicionais, para gerir os arrays de forma eficiente. A - Struct Uni reúne todas as informações numa única estrutura, facilitando a gestão e a manipulação dos dados, suporta o crescimento dinâmico da base de dados sem comprometer a eficiência e permite a extensão futura para adicionar novas funcionalidades, como estatísticas mais avançadas ou integração com outros sistemas.

```c
//Os arrays destas structs DEVEM ser ORDENADOS
typedef struct uni{
    Estudante * aluno;
    int tamanho_aluno; //tamanho atual do array aluno
    int capacidade_aluno; //tamanho alocado do array aluno
    Dados * escolares;
    int tamanho_escolares;
    int capacidade_escolares;
    Estatisticas stats;
} Uni;
```

Figura 3- Struct Uni

O uso das Structs no projeto é essencial para organizar a informação de forma eficiente e estruturada. Desde os dados individuais até as estatísticas globais, cada Struct desempenhou um papel vital na implementação de uma aplicação funcional e escalável. A integração entre estas estruturas garantiu uma solução robusta, com potencial para ser expandida e adaptada às necessidades futuras.

2.2. Algoritmos

Os algoritmos implementados no programa são variados, cada um com o objetivo de tornar o programa o mais eficiente possível, priorizando alguns aspetos que foram considerados mais importantes, no caso o facto de manter sempre os arrays ordenados. Esta escolha foi pensada previamente e foi levada a cabo devido ao facto de existirem muitas procuras de alunos durante o programa (como em listagens, inserções, etc) e, neste caso, o método mais rápido seria com uma procura binária (complexidade O(log n)), que requer o array ordenado.

Para além disto, ao carregar os dados de ficheiros .TXT, é necessário realizar diversas verificações sobre os códigos dos alunos (duplicados e inexistência de dados pessoais), que caso fossem realizadas com o array desordenado, levariam a um incremento da complexidade temporal. No entanto, estas vantagens trazem um custo, que é a introdução ou eliminação manual de um estudante, caso em que é necessário deslocar todo o array (O(n)).

Para concretizar isto, foi recorrido ao algoritmo de merge sort, procura binária, entre outros usados nas verificações.

2.2.1. Ordenação com Merge Sort

O Merge Sort é um algoritmo de ordenação eficiente (complexidade O(n log n)), já bastante conhecido. Como tal, e não havendo a necessidade de reinventar, foi apenas copiado do código base do algoritmo, efetuando os ajustes necessários ao contexto da aplicação, como observado na **Figura 4- Função** Merge Aluno e na **- Função Merge Sort** Aluno. O mesmo foi feito para a **Estrutura Dados**. É de salientar que este algoritmo apenas é usado após carregar os dados pela primeira vez para o programa, e foi escolhido não só pela sua eficiência mas também pelo facto dele deixar os códigos na ordem em que os encontrou, caso sejam iguais.

```c
void merge_aluno(Uni * bd, int inicio, int meio, int fim) {
    int tamanho_esquerda = meio - inicio + 1; //Limitar a metade esquerda
    int tamanho_direita = fim - meio;

    //Criar arrays temporários para esquerda e direita
    Estudante * esquerda = malloc(tamanho_esquerda * sizeof(Estudante));
    Estudante * direita = malloc(tamanho_direita * sizeof(Estudante));

    //Copiar o conteúdo para os arrays temporários. É COPIADO TODO O CONTEÚDO DO ARRAY
    for(int i = 0; i < tamanho_esquerda; i++)
        esquerda[i] = bd->aluno[inicio + i];

    for(int i = 0; i < tamanho_direita; i++)
        direita[i] = bd->aluno[meio + 1 + i];

    int indice_esquerda = 0;
    int indice_direita = 0;
    int indice = inicio;

    while(indice_esquerda < tamanho_esquerda && indice_direita < tamanho_direita) {
        if (esquerda[indice_esquerda].codigo <= direita[indice_direita].codigo) {
            bd->aluno[indice] = esquerda[indice_esquerda];
            indice_esquerda++;
        }
        else {
            bd->aluno[indice] = direita[indice_direita];
            indice_direita++;
        }
        indice++;
    }
    //Copiar elementos do array da direita que sobraram (ex: num da esquerda > num direita)
    while(indice_esquerda < tamanho_esquerda) {
        bd->aluno[indice] = esquerda[indice_esquerda];
        indice_esquerda++;
        indice++;
    }

    //Copiar elementos do array da esquerda(ex: o primeiro while terminou pois o tamanho do array esquerda < direita)
    while(indice_direita < tamanho_direita) {
        bd->aluno[indice] = direita[indice_direita];
        indice_direita++;
        indice++;
    }

    free(esquerda);
    free(direita);
}
```

Figura 4- Função Merge Aluno

```c
void merge_sort_aluno(Uni * bd, int inicio, int fim) {
    if (inicio < fim) { //Se inicio >= fim, o array tem 1
        int meio = inicio + (fim - inicio) / 2;
        merge_sort_aluno(bd, inicio, meio); //Ordena a primeira metade
        merge_sort_aluno(bd, meio + 1, fim); //Ordena a segunda metade
        //Isto vai ser usado recursivamente, pelo que cada metade vai ser novamente cortada a metade,... até o array ter um elemento

        merge_aluno(bd, inicio, meio, fim); //Combina as duas metades ordenadas
    }
}
```

Figura 5- Função Merge Sort Aluno

2.2.2. Pesquisa Binária

A **pesquisa binária** é um dos algoritmos mais eficientes para localizar elementos em um conjunto de dados ordenados, como já mencionado anteriormente.

No programa, a pesquisa binária é utilizada para localizar estudantes com base no código na **- Função Procurar Código** Aluno, onde o código é o código do aluno a ser pesquisado e bd o ponteiro para a **- Struct** Uni**,** chegado ao final a função, em caso de sucesso se o índice encontrado for maior que 0, será retornado o índice onde código foi encontrado +1, se o índice não for encontrado a posição de inserção será menor do que 0, então será retornado -(posição

de inserção + 1). Isto acontece (não se retornar os índices diretamente) para distinguir de um possível erro, caso em que a posição de inserção será igual a 0 (ponteiros inválidos/array vazio).

```
1136  int procurar_codigo_aluno(int codigo, Uni * bd) {
1137      if (!bd || !bd->aluno || bd->tamanho_aluno <= 0) {
1138          return 0;
1139      }
1140      if (bd->aluno[0].codigo > codigo) return -1; //Se o array está sempre ordenado e o codigo é menor que que do array 0, então não existe e está abaixo do indice zero
1141      int limInf, meio, limSup;
1142      limInf = 0;
1143      limSup = bd->tamanho_aluno - 1;
1144
1145      while (limSup >= limInf) {
1146          meio = (limSup + limInf) / 2;
1147          if (bd->aluno[meio].codigo == codigo) return (meio + 1) ; //Dá return do indice
1148          else {
1149              if (bd->aluno[meio].codigo < codigo) limInf = meio + 1;
1150              else limSup = meio - 1;
1151          }
1152      }
1153      return -(limInf + 1); //Retorna a posição de inserção + 1(fazer < 0 para verificar código)
1154      //o +1 está a precaver no caso de limInf ser 0, para distinguir do return de um erro
1155  }
```

Figura 6- Função Procurar Código Aluno

Para que o algoritmo da pesquisa binária funcione, no caso da **- Função Procurar Código** Aluno será necessária, a ordenação do array onde os códigos dos alunos estejam do menor para o maior, o que nos é garantido pelo algoritmo descrito em **Ordenação com Merge Sort**.

Ao integrar este método em um sistema de gestão escolar, o programa assegura que consultas e operações dependentes de localização de registos sejam rápidas e escaláveis.

2.2.3. Inserção Ordenada

A inserção ordenada é uma estratégia que mantém os dados organizados quando novos elementos são adicionados ao array. No projeto, essa abordagem foi implementada para evitar a necessidade de reordenar o array completo após cada inserção, garantindo eficiência em termos de tempo e organização.

No programa, a **- Função Ordenar ao** Inserir é utilizada para adicionar novos estudantes ou dados académicos ao array, mantendo-o ordenado, com o uso do algoritmo da inserção ordenada.

```c
void ordenar_ao_inserir(int codigo, Uni * bd, int indice_aluno, int indice_escolares) {
    //Ordenar aluno
    //Copia-se por partes para evitar ponteiros duplicados
    for(int i = bd->tamanho_aluno; i > indice_aluno; i--) {
        //Libertamos a memória do elemento atual
        free(bd->aluno[i].nome);
        free(bd->aluno[i].nacionalidade);
        //Alocamos nova memória para o elemento atual, para não termos ponteiros duplicados a apontar para o mesmo sítio
        bd->aluno[i].nome = malloc(strlen(bd->aluno[i - 1].nome) + 1);
        bd->aluno[i].nacionalidade = malloc(strlen(bd->aluno[i - 1].nacionalidade) + 1);

        if (bd->aluno[i].nome && bd->aluno[i].nacionalidade) {
            strcpy(bd->aluno[i].nome, bd->aluno[i - 1].nome);
            strcpy(bd->aluno[i].nacionalidade, bd->aluno[i - 1].nacionalidade);
        }

        //Copiar o resto da struct
        bd->aluno[i].codigo = bd->aluno[i - 1].codigo;
        bd->aluno[i].nascimento = bd->aluno[i - 1].nascimento;
    }
    //Precisamos de colocar os dados do novo aluno, no caso o código, os outros inicializamos para não ficar com os dados do anterior
    bd->aluno[indice_aluno].codigo = codigo;
    bd->aluno[indice_aluno].nascimento.dia = 0;
    bd->aluno[indice_aluno].nascimento.mes = 0;
    bd->aluno[indice_aluno].nascimento.ano = 0;
    bd->aluno[indice_aluno].nacionalidade = (char *) malloc (MAX_STRING_NACIONALIDADE * sizeof(char));
    bd->aluno[indice_aluno].nome = (char *) malloc (TAMANHO_INICIAL_NOME * sizeof(char));
    if (bd->aluno[indice_aluno].nacionalidade && bd->aluno[indice_aluno].nome) {
        strcpy(bd->aluno[indice_aluno].nome ,"-1");
        strcpy(bd->aluno[indice_aluno].nacionalidade, "-1");
    }
    //Incrementar tamanho total
    bd->tamanho_aluno++;

    //Ordenar escolares
    for(int i = bd->tamanho_escolares; i > indice_escolares; i--) {
        bd->escolares[i] = bd->escolares[i - 1];
    }
    bd->escolares[indice_escolares].codigo = codigo;
    bd->escolares[indice_escolares].matriculas = -1;
    bd->escolares[indice_escolares].ects = -1;
    bd->escolares[indice_escolares].ano_atual = -1;
    bd->escolares[indice_escolares].prescrever = '-';
    bd->escolares[indice_escolares].finalista = '-';
    bd->escolares[indice_escolares].media_atual = -1;

    bd->tamanho_escolares++;
}
```

Figura 7- Função Ordenar ao Inserir

O algoritmo de inserção ordenada localiza a posição correta para o novo elemento dentro do array já ordenado. Ele desloca os elementos existentes para abrir espaço na posição desejada e, em seguida, insere o novo elemento, onde o algoritmo compara o valor do novo elemento com os elementos existentes, do menor para o maior, desloca os elementos maiores para frente até encontrar a posição correta e insere o novo elemento na posição encontrada.

A implementação da inserção ordenada no projeto é uma solução eficiente para garantir que os dados estejam sempre organizados, melhorando a integração com outros algoritmos, como a pesquisa binária.

Quando queremos eliminar um estudante (que é feito numa função diferente, a função ordenar_ao_eliminar()), a lógica é a mesma, mas no sentido contrário.

2.2.4. Manipulação de Strings, Normalização UTF-8 e Portabilidade

No projeto, a manipulação de strings desempenha um papel essencial, uma vez que muitos dados, como os nomes dos estudantes e as nacionalidades, são armazenados como strings.

A - **Função Validar** Nome não apenas valida o nome, mas também oferece um conjunto robusto de verificações e correções automáticas, como ajuste de memória, remoção de caracteres desnecessários e verificação rigorosa dos caracteres permitidos. Isto torna-a uma peça essencial para manter a consistência e segurança no armazenamento de dados textuais dentro do programa.

```
1704  int validar_nome(Estudante * aluno, char * nome, const char modo) {
1705      //Verificar nome vazio
1706      if (!nome || nome[0] == '\0') { //NULL != '\0'
1707          if (modo == '1') printf("\nNome em branco!\n");
1708          return 0;
1709      }
1710      int comprimento = strlen(nome);
1711      //Retirar o \n no final, se tiver, mas apenas se a string for apenas \n não contamos como válido
1712      if (nome[comprimento - 1] == '\n' && comprimento != 1) {
1713          nome[comprimento - 1] = '\0';
1714          comprimento--;
1715      }
1716      // Verificar o último caractere
1717      if (nome[0] == ' ' || nome[comprimento - 1] == ' ') {
1718          if (modo == '1') printf("\nO nome tem espaços indevidos!\n");
1719          return 0;
1720      }
1721      for (int i = 0; i < comprimento; i++) {
1722          //Verificar se tem separador
1723          if (nome[i] == SEPARADOR) {
1724              if (modo == '1') printf("\nO nome contém um caracter separador inválido (%c).\n", SEPARADOR);
1725              return 0;
1726          }
1727          // Verificar se há dois espaços seguidos
1728          if (i < comprimento -1 && nome[i] == ' ' && nome[i+1] == ' ') {
1729              if (modo == '1') printf("\nO nome não pode conter espaços consecutivos!\n");
1730              return 0;
1731          }
1732          //Verificar caracteres inválidos
1733          if (!iswalpha(nome[i]) && nome[i] != ' ' && nome[i] != '-') { //iswalpha apenas retorna válido a-z e A-Z, logo as outras condições validam espaços e hifens
1734              if (modo == '1') printf("\nNome contém caracteres inválidos!\n");
1735              return 0;
1736          }
1737
1738      }
1739      //Se estamos aqui é válido, e precisamos de verificar se o nome cabe no array
1740      if (comprimento > TAMANHO_INICIAL_NOME - 1) {
1741          //Realocar o nome
1742          if (!realocar_nome(aluno, modo)) {
1743              return 0;
1744          }
1745      }
1746      return 1;
1747  }
```

Figura 8- Função Validar Nome

Assim como a - **Função Validar** Nome, ainda temos presente no codigo uma função que também valida uma string, a função validar_nacionalidade(), onde é feito todo o mesmo processo, mas para validar a nacionalidade dos alunos.

Além disso, devido à necessidade de suportar caracteres especiais, como acentos e cedilhas, a normalização UTF-8 foi implementada para garantir compatibilidade com esses mesmos caracteres especiais.

Tal foi feito recorrendo à - **Função Colocar Terminal** UTF8 onde houve a necessidade de dividir a normalização em duas partes, uma para os sistemas Windows e outra para os sistemas Linux, macOS, etc. Tal foi possível devido à variável de ambiente "_WIN32", que é definida automaticamente por todos os sistemas Windows. Nesse caso, usamos as funções da biblioteca <windows.h> para evitar a desformatação (o setlocale não funciona a 100%). No entanto, ainda é necessário converter o uso de valores decimais para usar vírgulas em vez de

pontos, caso em que o setlocale(LC_NUMERIC, “Portuguese”) ajuda bastante, pois só altera a parte numérica.

Em todos os outros casos, utiliza-se o setlocale padrão.

```
4386  void colocar_terminal_utf8() {
4387      #ifdef _WIN32
4388          //SetConsoleOutputCP retorna 0 se houver um erro
4389          if ((SetConsoleOutputCP(CP_UTF8) == 0)||SetConsoleCP(CP_UTF8) == 0) {
4390              printf("Ocorreu um erro ao configurar o terminal do Windows para UTF-8.\n");
4391              printf("A aplicação irá continuar. Desformatação será visível. Para resolver, reinicie a aplicação.\n");
4392          }
4393          setlocale(LC_NUMERIC, "Portuguese"); //Apenas afeta os números , ou seja, muda a notação de floats de '.' para ','
4394      #else
4395          setlocale(LC_ALL, "pt_PT.UTF-8");
4396      #endif
4397  }
```

Figura 9- Função Colocar Terminal UTF8

Ainda sobre o facto do programa ser concebido para vários sistemas, a **- Função Limpar** Terminal tem um papel imprescindível no programa, já que é utilizada com bastante frequência para oferecer uma melhor experiência a navegar no terminal, e segue a mesma lógica da anterior.

```
4339  void limpar_terminal() {
4340      #ifdef _WIN32 //_WIN32 é uma variável local automaticamente pre-definida pelo windows em todos os sistemas
4341          system("cls"); //Sistemas windows
4342      #else
4343          system("clear"); //Sistemas linux, macos, etc
4344      #endif
4345  }
```

Figura 10- Função Limpar Terminal

2.3. Sistema de Ficheiros

O sistema de ficheiros é um dos pilares do projeto, mantendo os dados inalterados entre sessões e permitindo que as informações sejam armazenadas, carregadas e manipuladas de maneira mais eficiente. O sistema foi projetado para suportar dois formatos principais: ficheiros binários e ficheiros de texto, além de incluir um sistema de logging para registo de erros.

2.3.1. Ficheiros Binários

Os ficheiros binários são utilizados para armazenar e carregar dados dos arrays de estudantes e informações académicas de forma compacta e eficiente. Este formato é ideal para manter a integridade dos dados, pois não depende de formatações específicas ou delimitadores, e minimiza o tamanho dos ficheiros.

A **- Função Guardar Dados** Bin é responsável por gravar o estado atual dos arrays em um ficheiro binário, preservando informações importantes para usos futuros.

```c
void guardar_dados_bin(const char * nome_ficheiro, Uni * bd, const char modo) {
    //Abrir ficheiro binário em modo de escrita
    FILE * ficheiro = fopen(nome_ficheiro, "wb");
    if (!ficheiro) {
        if (modo == '1') printf("Ocorreu um erro ao guardar os dados.\n");
        pressione_enter();
        return;
    }
    //Checksum
    unsigned long checksum = calcular_checksum(bd);
    fwrite(&checksum, sizeof(unsigned long), 1, ficheiro);

    //Configurações
    fwrite(&autosaveON, sizeof(char), 1, ficheiro);

    //Dados dos arrays
    fwrite(&bd->tamanho_aluno, sizeof(int), 1, ficheiro);
    fwrite(&bd->capacidade_aluno, sizeof(int), 1, ficheiro);

    fwrite(&bd->tamanho_escolares, sizeof(int), 1, ficheiro);
    fwrite(&bd->capacidade_escolares, sizeof(int), 1, ficheiro);

    //Escrever aluno
    for (int i = 0; i < bd->tamanho_aluno; i++) {
        fwrite(&bd->aluno[i].codigo, sizeof(int), 1, ficheiro);
        fwrite(&bd->aluno[i].nascimento, sizeof(Data), 1, ficheiro);

        //Para as strings, guardamos o tamanho e a própria string por uma questão de ser mais fácil de ler mais tarde
        size_t tamanho_nome = strlen(bd->aluno[i].nome) + 1;
        size_t tamanho_nacionalidade = strlen(bd->aluno[i].nacionalidade) + 1;

        fwrite(&tamanho_nome, sizeof(size_t), 1, ficheiro);
        fwrite(bd->aluno[i].nome, tamanho_nome, 1, ficheiro);

        fwrite(&tamanho_nacionalidade, sizeof(size_t), 1, ficheiro);
        fwrite(bd->aluno[i].nacionalidade, tamanho_nacionalidade, 1, ficheiro);
    }
    //Escrever escolares
    fwrite(bd->escolares, sizeof(Dados), bd->tamanho_escolares, ficheiro);

    //Estatisticas
    fwrite(&bd->stats, sizeof(Estatisticas), 1, ficheiro);

    fclose(ficheiro);
    if (modo == '1') {
        printf("Os dados foram guardados com sucesso em '%s'.\n", nome_ficheiro);
        pressione_enter();
    }
}
```

Figura 111- Função Guardar Dados Bin

A **- Função Guardar Dados** Bin com dois parâmetros principais. O primeiro é o nome do ficheiro binário onde os dados serão armazenados (ficheiro). O segundo é a estrutura principal (bd) que contém os arrays dinâmicos de estudantes e dados académicos. Após a execução, a função retorna 1 em caso de sucesso ou 0 se ocorrer algum erro durante a abertura do ficheiro ou a gravação dos dados.

O processo de gravação realizado pela função segue um fluxo bem definido para garantir a eficiência e a integridade dos dados. Primeiro, o ficheiro é aberto em modo binário de escrita (wb). Em seguida, o tamanho atual do array de estudantes é gravado como um valor inteiro, permitindo que a leitura futura saiba quantos registos esperar. Após isso, todos os elementos do array dinâmico são gravados sequencialmente no ficheiro. Por fim, o ficheiro é fechado para evitar perdas de dados ou corrupção do conteúdo.

Antes de realizar qualquer operação de gravação, a função valida se o ficheiro foi aberto corretamente. Caso a abertura falhe, uma mensagem de erro é exibida e a função retorna imediatamente, sem tentar gravar dados.

Ao carregar os dados de forma binária, a lógica é a mesma, basta ler os dados com o fread() na mesma ordem pela qual foram guardados. Além disso, é aqui que é alocada a memória para as structs caso não seja a primeira vez que o programa é executado.

2.3.2. Ficheiros de texto

Os ficheiros de texto são utilizados para exportar os dados de forma legível ou carregá-los de fontes externas. Este formato é mais acessível para inspeção manual e integração com outros sistemas.

A função carregar_dados_txt() é usada no primeiro carregamento de dados do programa e faz todas as verificações necessárias, para além de ordenar os arrays. Devido à sua grande extensão, não foi viável colocá-la aqui, no entanto será descrita. Esta função usa outras duas funções como a sua engrenagem: separar_parametros() e ler_linha_txt(). A última é usada para ler uma linha do ficheiro sem que correr o risco de perder dados porque a entrada é muito grande, já a primeira é uma espécie de strtok da <string.h>, mas já corrige o facto do último parâmetro ficar com '\n'. Isto seria escusado em condições normais, mas dado a função já ter sido feita antes do conhecimento da existência do strtok, e funcionar perfeitamente (e ainda removendo o '\n'), decidimos mantê-la, para evitar alterar a lógica da função desnecessariamente. Para além disto, a função trata de colocar os erros num ficheiro definido como ERROS_TXT, caso existam. Todos os parâmetros são lidos em ordem, e apenas são carregados para a struct caso todos sejam válidos.

A - **Função Guardar Dados** TXT grava os registos dos estudantes em um ficheiro de texto, com campos separados por tabulações.

```c
void guardar_dados_txt(const char * nome_ficheiro_dados, const char * nome_ficheiro_escolares, Uni * bd) {
    //Abrir ficheiros txt em modo de escrita
    FILE * aluno = fopen(nome_ficheiro_dados, "w");
    FILE * dados = fopen(nome_ficheiro_escolares, "w");

    if (!aluno) {
        printf("Ocorreu um erro ao abrir o ficheiro %s para guardar os dados.\n", nome_ficheiro_dados);
    }
    //Escrever aluno
    else {
        for(int i = 0; i < bd->tamanho_aluno; i++) {
            //Colocar \n apenas se não for nem a primeira nem a última entrada.
            if (i > 0) fprintf(aluno, "\n");
            fprintf(aluno, "%d%c%s%c%02hd-%02hd-%04hd%c%s",
                bd->aluno[i].codigo, SEPARADOR,
                bd->aluno[i].nome, SEPARADOR,
                bd->aluno[i].nascimento.dia, bd->aluno[i].nascimento.mes, bd->aluno[i].nascimento.ano, SEPARADOR,
                bd->aluno[i].nacionalidade);
            }
        fclose(aluno);
        printf("Os dados foram guardados com sucesso no ficheiro '%s'.\n", nome_ficheiro_dados);
    }

    if (!dados) {
        printf("Ocorreu um erro ao abrir o ficheiro %s para guardar os dados.\n", nome_ficheiro_escolares);
        pressione_enter();
        return; //Já não há mais nada a guardar
    }
    //Escrever escolares
    for(int i = 0; i < bd->tamanho_escolares; i++) {
        if (i > 0) fprintf(dados, "\n");
        fprintf(dados, "%d%c%hd%c%hd%c%hd%c%.1f",
            bd->escolares[i].codigo, SEPARADOR,
            bd->escolares[i].matriculas, SEPARADOR,
            bd->escolares[i].ects, SEPARADOR,
            bd->escolares[i].ano_atual, SEPARADOR,
            bd->escolares[i].media_atual);
    }
    fclose(dados);

    printf("Os dados foram guardados com sucesso no ficheiro '%s'.\n", nome_ficheiro_escolares);
    pressione_enter();
}
```

Figura 12- Função Guardar Dados TXT

A - **Função Guardar Dados** TXT recebe como parâmetros o nome do ficheiro para os dados dos alunos, o nome do ficheiro para os dados escolares e um ponteiro para a estrutura principal (Uni). O processo de gravação não retorna valores, mas emite mensagens de erro caso ocorra algum problema, como a impossibilidade de abrir os ficheiros.

O funcionamento da função segue um fluxo claro, onde os ficheiros especificados são abertos em modo de escrita. Se não existirem, são criados automaticamente, de seguida os registos são gravados e ordenados por código, garantindo a consistência necessária para operações futuras, todos os campos são separados por um delimitador definido (SEPARADOR), tornando os ficheiros legíveis e compatíveis com outros sistemas, após a gravação, os ficheiros são fechados, garantindo a integridade dos dados.

Caso ocorra algum problema durante o processo de gravação, como falhas ao abrir os ficheiros, a função exibe mensagens de erro no terminal. Isso assegura que os utilizadores sejam informados de eventuais problemas, permitindo correções rápidas para garantir que os dados sejam devidamente armazenados.

A **- Função Guardar Dados** TXT é essencial para o sistema de persistência de dados em formato de texto, permitindo que as informações sejam armazenadas de maneira legível e acessível. A manutenção da ordem dos registos, o uso de separadores e a criação automática de ficheiros garantem a organização e a confiabilidade do processo de gravação. Além disso, a sua flexibilidade e compatibilidade com outros sistemas tornam-na uma solução eficaz para a gestão de dados escolares.

2.3.3. Ficheiro Erros

A criação de um ficheiro de erros surgiu pela mera necessidade de manter todos os dados inseridos no programa, sem levar à completa exclusão dos mesmos. Desta forma, ao carregar os dados através de ficheiros .TXT, os dados que forem considerados errados, não só não serão eliminados, mas também serão acompanhados do motivo de exclusão.

Desta forma, é possível manter "logs" do programa, e fica também em aberto a possibilidade de usar este mesmo ficheiro para o registo de outros erros posteriores na execução do programa.

De momento, o ficheiro está a ser aberto em modo de escrita, "w", mas pode perfeitamente ser modificado para "a", caso seja frequente a utilização de dados .TXT. Esta escolha (manter o "w") foi feita tendo em mente que o utilizador irá carregar os dados .TXT apenas uma vez durante a utilização da aplicação, para manter a facilidade de uso e legibilidade.

3. Implementação

A implementação do projeto foi cuidadosamente estruturada para garantir que todas as funcionalidades atendessem aos objetivos propostos de forma eficiente. Este capítulo apresenta os principais componentes do sistema, abordando a gestão de dados e a interface apresentada ao utilizador.

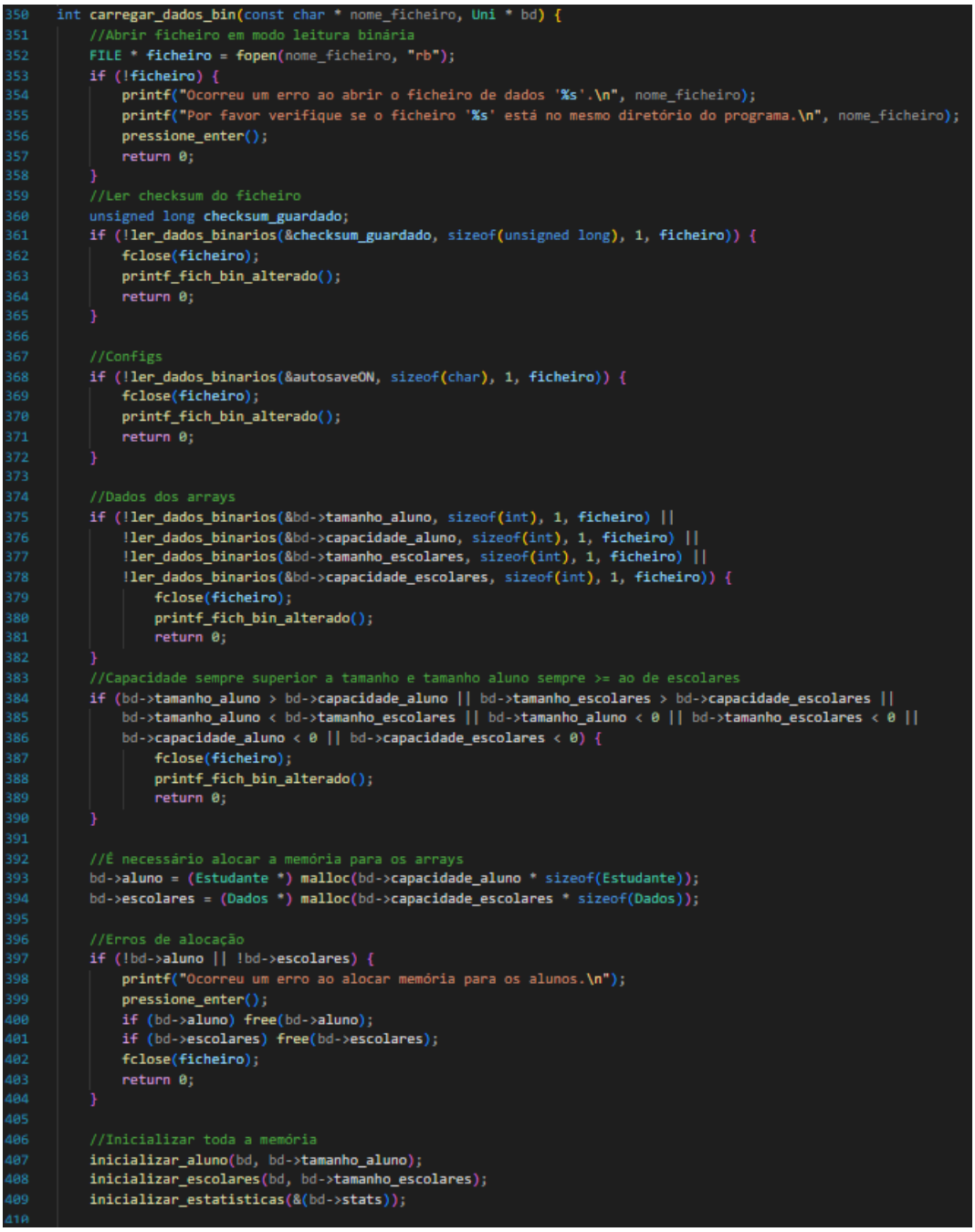
3.1. Gestão de Dados

A gestão de dados é responsável por armazenar, organizar e manipular as informações dos estudantes e dados escolares. Esta área inclui operações como inserção, consulta e remoção de registos, garantindo que os arrays dinâmicos permaneçam consistentes e ordenados.

3.1.1. Inicialização de Dados

A inicialização dos dados é uma etapa fundamental no sistema, responsável por configurar as estruturas de dados e preparar o programa para operar corretamente. No caso de ser a primeira iteração do programa, tudo passará pela função carregar_dados_txt(), já abordada antes. Caso contrário, será encargo da **- Função Carregar Dados** Bin pt.2, que é projetada para carregar os dados armazenados em ficheiros binários previamente guardados no sistema. Este processo permite que o programa recupere as informações dos alunos e dos dados escolares ao iniciar.

A Erro! A origem da referência não foi encontrada.inicializa dinamicamente a estrutura de dados da universidade a partir de um ficheiro binário especificado. Durante esse processo, a função valida a integridade dos dados utilizando um checksum, aloca a memória necessária para as estruturas e garante a inicialização apropriada de cada estrutura alocada, assegurando a consistência e a robustez do sistema ao lidar com erros de leitura ou formatos inválidos, além de garantir a integridade dos dados através do checksum, a função também verifica se o ficheiro pode ser aberto corretamente e se os dados podem ser lidos sem erros. Caso algum problema seja identificado, como erros de leitura ou inconsistência no checksum, a função retorna um valor de erro, permitindo uma gestão eficaz de falhas durante o carregamento dos dados.



```c
int carregar_dados_bin(const char * nome_ficheiro, Uni * bd) {
    //Abrir ficheiro em modo leitura binária
    FILE * ficheiro = fopen(nome_ficheiro, "rb");
    if (!ficheiro) {
        printf("Ocorreu um erro ao abrir o ficheiro de dados '%s'.\n", nome_ficheiro);
        printf("Por favor verifique se o ficheiro '%s' está no mesmo diretório do programa.\n", nome_ficheiro);
        pressione_enter();
        return 0;
    }
    //Ler checksum do ficheiro
    unsigned long checksum_guardado;
    if (!ler_dados_binarios(&checksum_guardado, sizeof(unsigned long), 1, ficheiro)) {
        fclose(ficheiro);
        printf_fich_bin_alterado();
        return 0;
    }

    //Configs
    if (!ler_dados_binarios(&autosaveON, sizeof(char), 1, ficheiro)) {
        fclose(ficheiro);
        printf_fich_bin_alterado();
        return 0;
    }

    //Dados dos arrays
    if (!ler_dados_binarios(&bd->tamanho_aluno, sizeof(int), 1, ficheiro) ||
        !ler_dados_binarios(&bd->capacidade_aluno, sizeof(int), 1, ficheiro) ||
        !ler_dados_binarios(&bd->tamanho_escolares, sizeof(int), 1, ficheiro) ||
        !ler_dados_binarios(&bd->capacidade_escolares, sizeof(int), 1, ficheiro)) {
            fclose(ficheiro);
            printf_fich_bin_alterado();
            return 0;
    }
    //Capacidade sempre superior a tamanho e tamanho aluno sempre >= ao de escolares
    if (bd->tamanho_aluno > bd->capacidade_aluno || bd->tamanho_escolares > bd->capacidade_escolares ||
        bd->tamanho_aluno < bd->tamanho_escolares || bd->tamanho_aluno < 0 || bd->tamanho_escolares < 0 ||
        bd->capacidade_aluno < 0 || bd->capacidade_escolares < 0) {
            fclose(ficheiro);
            printf_fich_bin_alterado();
            return 0;
    }

    //É necessário alocar a memória para os arrays
    bd->aluno = (Estudante *) malloc(bd->capacidade_aluno * sizeof(Estudante));
    bd->escolares = (Dados *) malloc(bd->capacidade_escolares * sizeof(Dados));

    //Erros de alocação
    if (!bd->aluno || !bd->escolares) {
        printf("Ocorreu um erro ao alocar memória para os alunos.\n");
        pressione_enter();
        if (bd->aluno) free(bd->aluno);
        if (bd->escolares) free(bd->escolares);
        fclose(ficheiro);
        return 0;
    }

    //Inicializar toda a memória
    inicializar_aluno(bd, bd->tamanho_aluno);
    inicializar_escolares(bd, bd->tamanho_escolares);
    inicializar_estatisticas(&(bd->stats));

```

Figura 13- Função Carregar Dados Bin pt.1

```c
    //Aluno
    for (int i = 0; i < bd->tamanho_aluno; i++) {
        if (!ler_dados_binarios(&bd->aluno[i].codigo, sizeof(int), 1, ficheiro)) {
            fclose(ficheiro);
            printf_fich_bin_alterado();
            return 0;
        }
        if (!ler_dados_binarios(&bd->aluno[i].nascimento, sizeof(Data), 1, ficheiro)) {
            fclose(ficheiro);
            printf_fich_bin_alterado();
            return 0;
        }
        //Ao carregar os respetivos tamanhos, as variáveis vão ficar com valor
        size_t tamanho_nome, tamanho_nacionalidade;

        //Nome
        if (!ler_dados_binarios(&tamanho_nome, sizeof(size_t), 1, ficheiro)) {
            fclose(ficheiro);
            printf_fich_bin_alterado();
            return 0;
        }
        bd->aluno[i].nome = (char *) malloc(tamanho_nome);
        if (!ler_dados_binarios(bd->aluno[i].nome, tamanho_nome, 1, ficheiro)) {
            fclose(ficheiro);
            printf_fich_bin_alterado();
            return 0;
        }

        //Nacionalidade
        if (!ler_dados_binarios(&tamanho_nacionalidade, sizeof(size_t), 1, ficheiro)) {
            fclose(ficheiro);
            printf_fich_bin_alterado();
            return 0;
        }
        bd->aluno[i].nacionalidade = (char *) malloc(tamanho_nacionalidade);
        if (!ler_dados_binarios(bd->aluno[i].nacionalidade, tamanho_nacionalidade, 1, ficheiro)) {
            fclose(ficheiro);
            printf_fich_bin_alterado();
            return 0;
        }
    }

    //Ler array de escolares
    if (!ler_dados_binarios(bd->escolares, sizeof(Dados), bd->tamanho_escolares, ficheiro)) {
        fclose(ficheiro);
        printf_fich_bin_alterado();
        return 0;
    }

    //Estatísticas
    if (!ler_dados_binarios(&bd->stats, sizeof(Estatisticas), 1, ficheiro)) {
        fclose(ficheiro);
        printf_fich_bin_alterado();
        return 0;
    }

    fclose(ficheiro);

    //Falta verificar se os checksums batem certo
    unsigned long checksum_atual = calcular_checksum(bd);
    if (checksum_atual != checksum_guardado) {
        printf_fich_bin_alterado();
        return 0;
    }

    return 1;
}
```

Figura 14- Função Carregar Dados Bin pt.2

3.1.2. Memória Dinâmica

A gestão de memória dinâmica é essencial para a eficiência do programa, permitindo a alocação e realocação de memória conforme necessário durante a execução. A lógica passou por criar um array de uma dimensão fixa e aumentar o mesmo caso surja necessidade disso.

Como tal, foram criadas diversas funções para o efeito, como realocar_aluno, realocar_escolares e realocar_nome. Aqui encontra-se um porém, a nacionalidade apesar de estar declarada como um ponteiro e de ser tratada como memória dinâmica (usando-se o free e malloc), tem um tamanho máximo predefinido (pois não há nenhuma nacionalidade com mais de um certo número de caracteres). Sendo assim, a declaração como ponteiro é uma

forma de permitir que, no futuro, seja possível aumentar o tamanho do array sem muitos contratempos.

Uma das grandes preocupações durante o projeto foi precisamente o uso da memória dinâmica, pois requer sempre um free no final. Em certos casos foi mais desafiante que outros, como na função procurar_nacionalidades(). Para procurar por fugas de memória, pretendia-se utilizar um programa chamado "Valgrind", no entanto, por falta de tempo, já não foi feito.

Outra das preocupações a lidar com memória dinâmica teve haver com ponteiros duplicados. Quando se atribui um ponteiro como por exemplo, uma string a outra, ambos os ponteiros (char *) estarão a apontar para o mesmo local na memória, o que leva a múltiplas preocupações, tal como o uso do free indevidamente e perda de dados.

3.1.3. Libertação de Memória

A libertação de memória é fundamental para evitar fugas de memória, como já mencionado. Para o fazer corretamente, foram verificados todos os caminhos possíveis de saída de uma função para, obviamente, libertar a memória antes disso.

Relativamente às structs, a sua memória apenas é libertada quando o programa termina, ou seja, quando o utilizador sai ou ocorre um erro (neste caso não são consideradas as exceções, pois é um pouco complexo e requeria significativamente mais código) através da função free_tudo(), que liberta toda a memória das structs, incluindo os campos do nome e nacionalidade. Obviamente que, caso seja necessário libertar memória para uma struct individualmente, se fará sem recorrer a esta função.

3.2. Interface

A interface do programa foi projetada para proporcionar uma interação intuitiva com o utilizador, permitindo o acesso rápido e organizado a todas as funcionalidades principais do sistema de gestão de dados escolares.

3.2.1. Menus

Os menus no sistema desempenham um papel crucial na interface com o utilizador, proporcionando uma navegação intuitiva e uma experiência de uso eficiente. O design dos menus foi cuidadosamente estruturado para garantir clareza e facilidade de utilização, como por exemplo na **- Função Menu** Principal.

Os menus utilizam elementos visuais como duplos traços ("==") e separadores para estruturar a interface, facilitando a distinção entre diferentes secções e opções. Estes elementos visuais são especialmente úteis para a compreensão rápida, elementos esses retirados da tabela ASCII, disponibilizada em (desenvolvedorinteroperavel.wordpress.com, s.d.).

```
2177  void menu_principal() {
2178      //https://desenvolvedorinteroperavel.wordpress.com/2011/09/11/tabela-ascii-completa/
2179      //Link da tabela ASCII completa de onde foram retirados as duplas barras do menu (a partir do 185 decimal)
2180      printf("╔═══════════════════════════════╗\n");
2181      printf("║          MENU PRINCIPAL       ║\n");
2182      printf("╠═══════════════════════════════╣\n");
2183      printf("║  1. Gerir estudantes          ║\n");
2184      printf("║  2. Consultar dados           ║\n");
2185      printf("║  3. Estatísticas              ║\n");
2186      printf("║  4. Ficheiros                 ║\n");
2187      printf("║  5. Aniversários              ║\n");
2188      printf("║  6. Opções                    ║\n");
2189      printf("║  0. Sair do programa          ║\n");
2190      printf("╚═══════════════════════════════╝\n\n");
2191  }
```

Figura 15- Função Menu Principal

As funções dos menus, implementadas entre as linhas 2133 e 2360 do ficheiro **funcoes.c**, são responsáveis por gerir a interação do utilizador com o sistema. Estas funções incluem a exibição das opções disponíveis, como é possível ver na **Figura 15,** o processamento das escolhas do utilizador e a chamada às funções correspondentes para executar as operações selecionadas, funções como **menu_gerir_estudantes**, **menu_consultar_dados**, **menu_estatisticas**, **menu_ficheiros**, **menu_opcoes**, **menu_aniversarios**, **menu_dias_da_semana**, **menu_formatos_disponiveis**, **menu_media_matriculas** e **guia_de_utilizacao**, são funções que exibem no ecrã um menu semelhante ao da **Figura 15**, seguindo os mesmos padrões de estética, mudando apenas o texto correspondente a cada opção de seleção nos menus.

```
2147  char mostrar_menu(void (*escrever_menu)(), char min_opcao, char max_opcao) {
2148      short valido = 0;
2149      char opcao = '0';
2150      do {
2151          limpar_terminal();
2152          escrever_menu(); //Escreve a função do menu
2153          printf("=>Escolha uma opção: ");
2154
2155          valido = scanf(" %c", &opcao); //scanf retorna 1 se conseguir ler corretamente
2156          validacao_menus(&valido, opcao, min_opcao, max_opcao);
2157
2158          if (valido == 1) {
2159              return opcao;
2160          }
2161      } while (valido == 0);
2162
2163      return '0'; //colocado aqui porque o compilador não gostava de não haver return
2164  }
```

Figura 16- Função Mostrar Menu

A - **Função Mostrar** Menu é uma função fundamental no que toca aos menus, pois suporta a personalização das opções exibidas em todos os menus e mostra mensagens de erro claras se as entradas forem inválidas, fazendo a ligação entre os menus, ou seja consoante a

escolha do utilizador num menu, o programa avança para o menu que corresponde a opção selecionada ou mostra aquilo que foi selecionado pelo utilizador no menu.

4. Funcionalidades

O sistema integra diversas funcionalidades que abrangem a gestão de estudantes e cálculos estatísticos, garantindo uma análise completa e eficiente dos dados armazenados.

4.1. Gestão de Estudantes

A gestão de estudantes engloba operações fundamentais como inserção, consulta, atualização e remoção de registos, o que possibilita uma administração mais detalhada e organizada dos dados acadêmicos.

4.1.1. Inserção de Estudante

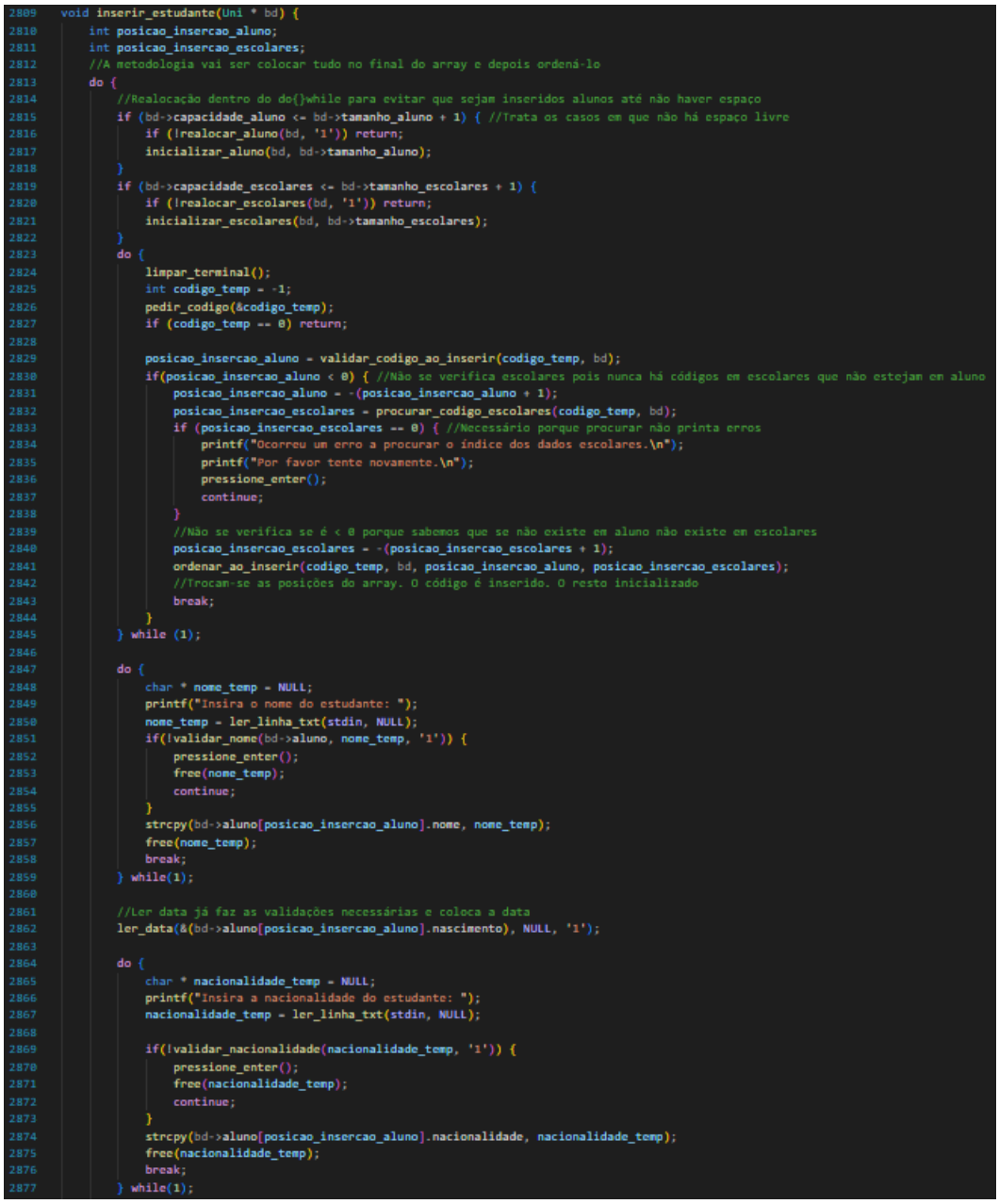
A inserção de estudantes é uma funcionalidade essencial do sistema, que permite registar novos alunos na base de dados da universidade. Esta funcionalidade é implementada principalmente pela Erro! A origem da referência não foi encontrada..

A **- Função Inserir** Estudante pt.3 é responsável por adicionar um novo registo de estudante à base de dados da universidade. Este processo é realizado de forma robusta, garantindo que todas as entradas sejam válidas, os arrays de dados sejam mantidos ordenados e as alocações de memória sejam geridas corretamente.

A **- Função Inserir** Estudante pt.3 valida rigorosamente cada entrada fornecida pelo utilizador, como código e nome, garantindo que os dados sejam armazenados no formato correto, além disso, mantém a ordenação do array por código, aloca dinamicamente memória para strings e, caso o limite de capacidade seja atingido, realoca os arrays para suportar novos registos.

Com a utilização de um loop de validação que aceita apenas entradas válidas, a função integra ferramentas auxiliares como a função **ler_string** e a função **ordenar_alunos**, permitindo a leitura de strings de tamanho variável e reorganização dos registos após cada inserção, automatizando o uso de memória o que assegura as operações subsequentes, como a **Pesquisa Binária**.

A implementação da **- Função Inserir** Estudante pt.3 reflete boas práticas de programação ao combinar validação de dados, ordenação dinâmica e gestão eficiente de memória, garantindo robustez no processo de inserção o que contribui diretamente na consistência e escalabilidade no programa de gestão académica.



```c
void inserir_estudante(Uni * bd) {
    int posicao_insercao_aluno;
    int posicao_insercao_escolares;
    //A metodologia vai ser colocar tudo no final do array e depois ordená-lo
    do {
        //Realocação dentro do do{}while para evitar que sejam inseridos alunos até não haver espaço
        if (bd->capacidade_aluno <= bd->tamanho_aluno + 1) { //Trata os casos em que não há espaço livre
            if (!realocar_aluno(bd, '1')) return;
            inicializar_aluno(bd, bd->tamanho_aluno);
        }
        if (bd->capacidade_escolares <= bd->tamanho_escolares + 1) {
            if (!realocar_escolares(bd, '1')) return;
            inicializar_escolares(bd, bd->tamanho_escolares);
        }
        do {
            limpar_terminal();
            int codigo_temp = -1;
            pedir_codigo(&codigo_temp);
            if (codigo_temp == 0) return;

            posicao_insercao_aluno = validar_codigo_ao_inserir(codigo_temp, bd);
            if(posicao_insercao_aluno < 0) { //Não se verifica escolares pois nunca há códigos em escolares que não estejam em aluno
                posicao_insercao_aluno = -(posicao_insercao_aluno + 1);
                posicao_insercao_escolares = procurar_codigo_escolares(codigo_temp, bd);
                if (posicao_insercao_escolares == 0) { //Necessário porque procurar não printa erros
                    printf("Ocorreu um erro a procurar o índice dos dados escolares.\n");
                    printf("Por favor tente novamente.\n");
                    pressione_enter();
                    continue;
                }
                //Não se verifica se é < 0 porque sabemos que se não existe em aluno não existe em escolares
                posicao_insercao_escolares = -(posicao_insercao_escolares + 1);
                ordenar_ao_inserir(codigo_temp, bd, posicao_insercao_aluno, posicao_insercao_escolares);
                //Trocam-se as posições do array. O código é inserido. O resto inicializado
                break;
            }
        } while (1);

        do {
            char * nome_temp = NULL;
            printf("Insira o nome do estudante: ");
            nome_temp = ler_linha_txt(stdin, NULL);
            if(!validar_nome(bd->aluno, nome_temp, '1')) {
                pressione_enter();
                free(nome_temp);
                continue;
            }
            strcpy(bd->aluno[posicao_insercao_aluno].nome, nome_temp);
            free(nome_temp);
            break;
        } while(1);

        //Ler data já faz as validações necessárias e coloca a data
        ler_data(&(bd->aluno[posicao_insercao_aluno].nascimento), NULL, '1');

        do {
            char * nacionalidade_temp = NULL;
            printf("Insira a nacionalidade do estudante: ");
            nacionalidade_temp = ler_linha_txt(stdin, NULL);

            if(!validar_nacionalidade(nacionalidade_temp, '1')) {
                pressione_enter();
                free(nacionalidade_temp);
                continue;
            }
            strcpy(bd->aluno[posicao_insercao_aluno].nacionalidade, nacionalidade_temp);
            free(nacionalidade_temp);
            break;
        } while(1);
```

Figura 17- Função Inserir Estudante pt.1

```
2879        do {
2880            short matriculas_temp = 0;
2881            printf("Insira o número de matrículas do estudante: ");
2882            if (scanf("%hd", &matriculas_temp) != 1) { //Verifica entradas inválidas como letras
2883                printf("Número de matrículas inválido! Insira um número inteiro positivo.\n");
2884                limpar_buffer();
2885                pressione_enter();
2886                continue; //Continue faz com que salte o resto do loop e passe à próxima iteração
2887            }
2888            else if (!verificar_e_limpar_buffer()) {
2889                printf("Entrada inválida! Por favor, escreva apenas o número de matrículas do aluno.\n");
2890                pressione_enter();
2891                continue;
2892            }
2893            if (matriculas_temp < 0 || matriculas_temp > MAX_MATRICULAS) {
2894                printf("Número de matrículas é inválido. Deve estar entre 0 e %d.\n", MAX_MATRICULAS);
2895                pressione_enter();
2896                continue;
2897            }
2898            bd->escolares[posicao_insercao_escolares].matriculas = matriculas_temp;
2899            break;
2900        } while(1);
2901
2902        do {
2903            short ects_temp = 0;
2904            printf("Insira o número de créditos ECTS do estudante: ");
2905            if (scanf("%hd", &ects_temp) != 1) { //Verifica entradas inválidas como letras
2906                printf("Créditos ECTS inválido! Insira um número inteiro positivo.\n");
2907                limpar_buffer();
2908                pressione_enter();
2909                continue; //Continue faz com que salte o resto do loop e passe à próxima iteração
2910            }
2911            else if (!verificar_e_limpar_buffer()) {
2912                printf("Entrada inválida! Por favor, escreva apenas o número de créditos ECTS do aluno.\n");
2913                pressione_enter();
2914                continue;
2915            }
2916            if (ects_temp < 0 || ects_temp > MAX_ECTS) {
2917                printf("O número de créditos ECTS é inválido. Deve estar entre 0 e %d.\n", MAX_ECTS);
2918                pressione_enter();
2919                continue;
2920            }
2921            bd->escolares[posicao_insercao_escolares].ects = ects_temp;
2922            break;
2923        } while(1);
2924
2925        do {
2926            short ano_atual_temp = 0;
2927            printf("Insira o ano atual de curso do estudante: ");
2928            if (scanf("%hd", &ano_atual_temp) != 1) { //Verifica entradas inválidas como letras
2929                printf("Ano atual de curso inválido! Insira um número inteiro positivo.\n");
2930                limpar_buffer();
2931                pressione_enter();
2932                continue; //Continue faz com que salte o resto do loop e passe à próxima iteração
2933            }
2934            else if (!verificar_e_limpar_buffer()) {
2935                printf("Entrada inválida! Por favor, escreva apenas o ano atual de curso do aluno.\n");
2936                pressione_enter();
2937                continue;
2938            }
2939            if (ano_atual_temp < 1 || ano_atual_temp > MAX_ANO_ATUAL ) {
2940                printf("O ano atual de curso é inválido. Deve estar entre 0 e %d.\n", MAX_ANO_ATUAL);
2941                pressione_enter();
2942                continue;
2943            }
2944            bd->escolares[posicao_insercao_escolares].ano_atual = ano_atual_temp;
2945            break;
2946        } while (1);
```

Figura 18- Função Inserir Estudante pt.2

```c
        do {
            float media_temp = 0.0;
            printf("Insira a média atual do estudante: ");
            if (scanf("%f", &media_temp) != 1) { //Verifica entradas inválidas como letras
                printf("Média atual inválido! Insira um número entre 0 e 20.\n");
                limpar_buffer();
                pressione_enter();
                continue; //Continue faz com que salte o resto do loop e passe à próxima iteração
            }
            else if (!verificar_e_limpar_buffer()) {
                printf("Entrada inválida! Por favor, escreva apenas a média do aluno.\n");
                pressione_enter();
                continue;
            }
            if (media_temp < 0 || media_temp > 20) {
                printf("A média atual é inválida. Deve estar entre 0 e 20.\n");
                pressione_enter();
                continue;
            }
            bd->escolares[posicao_insercao_escolares].media_atual = media_temp;
            break;
        } while(1);
        //Estatísticas estão desatualizadas
        bd->stats.atualizado = '0';

        printf("\nO código %d foi introduzido com sucesso!\n", bd->aluno[posicao_insercao_aluno].codigo);

        printf("\nQuer inserir mais estudantes? (S/N): ");
        if (!sim_nao()) return;
    }while(1); //Não precisamos de verificar repetir pois só chega aqui se repetir == 's'
    //Introduzimos um estudante, logo os dados estatísticos têm de ser recalculados.
}
```

Figura 19- Função Inserir Estudante pt.3

4.1.2. Consulta de Estudante

A consulta de estudantes permite pesquisar informações detalhadas sobre um aluno específico com base no seu respetivo nome. Esta funcionalidade é implementada pela **- Função Procurar Estudante por** Nome.

A funcionalidade de consulta de estudantes, implementada pela **- Função Procurar Estudante por** Nome, realiza buscas eficientes e intuitivas por parte do nome do estudante, sendo ideal para localizar registos em bases de dados com muitos elementos.

A função solicita ao utilizador uma parte do nome e utiliza operações de manipulação de strings para localizar estudantes que correspondam aos critérios fornecidos, exibindo os resultados de forma clara e organizada.

Com validação rigorosa dos inputs, normalização de strings para garantir uma correspondência consistente e a opção de exportar os resultados para ficheiros, esta funcionalidade proporciona flexibilidade e precisão na consulta de dados.

A implementação da função reflete boas práticas de programação, como manipulação segura de memória e integração de recursos úteis, tornando-a indispensável para a gestão eficiente de estudantes no sistema.

```c
void procurar_estudante_por_nome(Uni * bd) {
    short contador = 0;
    char * parte_nome;
    char formato[MAX_FORMATO]; //Espaço suficiente para .txt ou .csv (10)
    char separador;
    FILE * listagem;

    do {
        contador = 0;
        listagem = NULL;
        limpar_terminal();
        //Listagem
        listagem = pedir_listagem(formato);
        separador = obter_separador(listagem, formato);

        //Pedir nome
        do {
            parte_nome = NULL;
            printf("Introduza parte do nome do estudante a procurar: ");
            parte_nome = ler_linha_txt(stdin, NULL);
            if (!parte_nome || strlen(parte_nome) < 2) {
                printf("A entrada é inválida. Por favor, insira parte do nome do aluno, com um mínimo de 2 caracteres.\n");
                pressione_enter();
                if (parte_nome) free(parte_nome);
                continue;
            }
            break;
        } while(1);
        //Faz-se isto para evitar mudar o nome da variável
        char * orig = strdup(parte_nome);
        parte_nome = normalizar_string(parte_nome);

        printf("Resultados da pesquisa para \"%s\": \n\n", orig);
        for(int i = 0; i < bd->tamanho_aluno; i++) {
            //Colocar todos os dados a letras minúsculas
            char * nome = normalizar_string(bd->aluno[i].nome);
            if (nome) {
                //strstr(s1, s2) é uma função que retorna a primeira ocorrência de s2 em s1
                //https://www.geeksforgeeks.org/strstr-in-ccpp/
                if (strstr(nome, parte_nome) != NULL) {
                    listar(bd, i, listagem, separador, &contador);
                }
                free(nome);
            }
        }
        if (contador == 0)
            printf("Não foi encontrado nenhum estudante com parte do nome \"%s\".\n", orig);
        free(orig); //Libertamos o ponteiro original para não haver memory leaks

        pressione_enter();
        printf("\n--------------------FIM DE LISTAGEM--------------------\n\n");

        free(parte_nome);
        if (listagem) fclose(listagem);

        printf("\nQuer repetir a procura? (S/N): ");
        if (!sim_nao()) return;
    } while(1);
}
```

Figura 20- Função Procurar Estudante por Nome

4.1.3. Remoção de Estudante

A remoção de estudantes é implementada pela Erro! A origem da referência não foi encontrada.. Esta funcionalidade exclui permanentemente o registo de um aluno na base de dados.

A Erro! A origem da referência não foi encontrada. é responsável por remover registos de estudantes da base de dados, garantindo que todos os dados associados aos códigos fornecidos sejam excluídos.

Esta funcionalidade permite eliminar registos individuais ou intervalos de estudantes com base nos seus códigos, o que oferece flexibilidade na gestão da base de dados.

Após a exclusão, o array de estudantes é reorganizado para manter a ordenação por código, o que assegura que operações futuras, como a **Pesquisa Binária**, permaneçam eficientes.

Adicionalmente, a função liberta a memória alocada para os registos removidos e atualiza o tamanho do array, o que garante a consistência e a eficiência do sistema.

```
2993  void eliminar_estudante(Uni * bd) {
2994      do {
2995          limpar_terminal();
2996          printf("Quer eliminar alunos por intervalos? (S/N): ");
2997          //Intervalos
2998          if (sim_nao()) {
2999              int codigo_inf, codigo_sup;
3000              short contador;
3001              limpar_terminal();
3002
3003              //Limite inferior
3004              do {
3005                  limpar_terminal();
3006                  codigo_inf = -1;
3007                  printf("Intervalo inferior: \n");
3008                  pedir_codigo(&codigo_inf);
3009                  if (codigo_inf == 0) return;
3010                  //Intervalos podem ser de quaisquer valores, desde que sejam > 0 e de extensão máxima permitida
3011                  if (codigo_inf < 0) {
3012                      printf("Código inválido. Introduza um número inteiro positivo.\n");
3013                      pressione_enter();
3014                      continue;
3015                  }
3016                  //Casos em que não haveria nada a ser eliminado
3017                  else if (codigo_inf > bd->aluno[bd->tamanho_aluno - 1].codigo) {
3018                      printf("O código não existe e é superior a todos os existentes atualmente. Por favor introduza um novo código!\n");
3019                      pressione_enter();
3020                      continue;
3021                  }
3022                  break;
3023              } while(1);
3024
3025              pressione_enter();
3026              //Limite superior
3027              do {
3028                  limpar_terminal();
3029                  codigo_sup = -1;
3030                  printf("Intervalo superior: \n");
3031                  pedir_codigo(&codigo_sup);
3032                  if (codigo_sup == 0) return;
3033                  //Intervalos podem ser de quaisquer valores, desde que sejam > 0 e de extensão máxima permitida
3034                  if (codigo_sup < 0) {
3035                      printf("Código inválido. Introduza um número inteiro positivo.\n");\
3036                      pressione_enter();
3037                      continue;
3038                  }
3039                  //Casos em que não haveria nada a ser eliminado
3040                  else if (codigo_sup <= codigo_inf) {
3041                      printf("O código superior não pode ser igual ou inferior ao código inferior.\n");
3042                      printf("Por favor introduza um código superior a %d.\n", codigo_inf);
3043                      pressione_enter();
3044                      continue;
3045                  }
3046                  //Limite de eliminações por intervalo
3047                  else if ((codigo_sup - codigo_inf + 1) > MAX_ELIMINACOES_POR_INTERVALO) {
3048                      printf("O intervalo de códigos a eliminar não pode exceder %d.\n", MAX_ELIMINACOES_POR_INTERVALO);
3049                      printf("Por favor introduza um novo código superior.\n");
3050                      pressione_enter();
3051                      continue;
3052                  }
3053                  break;
3054              } while(1);
```

Figura 21- Função Eliminar Estudante pt.1

```
            contador = 0;
            //Eliminar todos os códigos do intervalo, se existirem
            for (int i = codigo_inf; i <= codigo_sup; i++) {
                if (ordenar_ao_eliminar(i, bd, '0') == 1) {
                    printf("O aluno %d foi eliminado!\n", i);
                    pausa_listagem(&contador);
                }
            }
            //ta a pausar quando encontra um codigo que nao existe

            if (contador == 0) {
                printf("Nenhum aluno foi encontrado dentro do intervalo.\n");
            }
        }
        //Um a um
        else {
            limpar_terminal(); //Caso de repetição
            do {
                int codigo_temp = -1;
                pedir_codigo(&codigo_temp);
                if (codigo_temp == 0) return;

                //Elimina e ordena o código dado, caso seja válido.
                if(ordenar_ao_eliminar(codigo_temp, bd, '1')) {
                    printf("O aluno com o código %d foi eliminado com sucesso!\n", codigo_temp);
                    break;
                }
            } while (1);
        }

        printf("\nQuer eliminar mais estudantes? (S/N): ");
        if(!sim_nao()) return;
    } while(1);
    bd->stats.atualizado = '0';
}
```

Figura 22- Função Eliminar Estudante pt.2

4.2. Cálculos

Os cálculos realizados pelo sistema oferecem uma visão abrangente do desempenho acadêmico e incluem métricas estatísticas e a determinação automatizada do período da Quaresma, utilizando algoritmos precisos e eficientes.

4.2.1. Estatísticas

A funcionalidade de cálculo de estatísticas oferece uma visão geral dos dados relacionados aos alunos, com indicadores como médias, número de finalistas e alunos em risco de prescrição. É implementada pela **- Função Calcular** Estatísticas, que processa os dados das matrículas e notas armazenados na **Estrutura U**.

A funcionalidade de cálculo de estatísticas, implementada pela **- Função Calcular** Estatísticas, processa os dados armazenados para fornecer métricas gerais sobre os estudantes. A função calcula a média de matrículas e média das notas dos alunos, o que permite avaliar o desempenho acadêmico global da universidade. Além disso, identifica alunos finalistas, ou seja, aqueles que já acumularam créditos suficientes para conclusão, e também aqueles em risco de prescrição, com base no número de matrículas e créditos obtidos.

```
3107  void calcular_estatisticas(Uni * bd) {
3108      //Inicializar caso já tenhamos chamado a função antes
3109      bd->stats.media_matriculas = 0.0;
3110      bd->stats.finalistas = 0;
3111      bd->stats.media = 0.0;
3112      bd->stats.risco_prescrever = 0;
3113
3114      for(int i = 0; i < bd->tamanho_escolares; i++) {
3115          //Matrículas
3116          bd->stats.media_matriculas += bd->escolares[i].matriculas;
3117          //Média
3118          bd->stats.media += bd->escolares[i].media_atual;
3119          //Finalistas
3120          if (bd->escolares[i].ects >= CREDITOS_FINALISTA) {
3121              bd->escolares[i].finalista = '1'; //Marcar o aluno como finalista
3122              bd->stats.finalistas++;
3123          }
3124          else bd->escolares[i].finalista = '0';
3125          //Risco de prescrição
3126          if ((bd->escolares[i].matriculas == 3 && bd->escolares[i].ects < ECTS_3MATRICULAS) ||
3127              (bd->escolares[i].matriculas == 4 && bd->escolares[i].ects < ECTS_4MATRICULAS) ||
3128              (bd->escolares[i].matriculas >= 5 && bd->escolares[i].finalista == '0')) {
3129                  bd->escolares[i].prescrever = '1';
3130                  bd->stats.risco_prescrever++;
3131          }
3132          else bd->escolares[i].prescrever = '0';
3133      }
3134      //Tratar da divisão por zero
3135      if (bd->tamanho_escolares != 0) {
3136          bd->stats.media_matriculas /= bd->tamanho_escolares;
3137          bd->stats.media /= bd->tamanho_escolares;
3138      }
3139      else {
3140          bd->stats.media_matriculas = 0.0;
3141          bd->stats.media = 0.0;
3142      }
3143      //Estatísticas estão atualizadas
3144      bd->stats.atualizado = '1';
3145  }
```

Figura 23- Função Calcular Estatísticas

4.2.2. Determinação do Período de Quaresma

O cálculo do período de Quaresma é um algoritmo matemático públicos, no caso, consultado no site (matematica.pt, s.d.).

A funcionalidade de determinação do período da Quaresma é construída com base na integração entre três funções principais, **- Função Calcular Domingo de** Páscoa, **- Função Calcular Quarta Feira** Cinzas e **- Função Calcular** Quaresma.

A **- Função Calcular Domingo de** Páscoa utiliza o algoritmo de Meeus/Jones/Butcher para determinar a data exata do Domingo de Páscoa para um dado ano. Com base nesse resultado, a **- Função Calcular Quarta Feira** Cinzas recua quarenta e seis dias para calcular a Quarta-feira de Cinzas, marcando o início da Quaresma.

Por fim, a **- Função Calcular** Quaresma consolida os cálculos, estabelecendo as datas de início e fim da Quaresma e ajustando corretamente as transições entre meses e anos.

```
4595  Data calcular_domingo_pascoa(int ano) {
4596      //int porque os números podem ficar muito grandes e estourar com short
4597      Data pascoa;
4598
4599      int a = ano % 19;
4600      int b = ano / 100;
4601      int c = ano % 100;
4602      int d = b / 4;
4603      int e = b % 4;
4604      int f = (b + 8) / 25;
4605      int g = (b - f + 1) / 3;
4606      int h = (19 * a + b - d - g + 15) % 30;
4607      int i = c / 4;
4608      int k = c % 4;
4609      int l = (32 + 2 * e + 2 * i - h - k) % 7;
4610      int m = (a + 11 * h + 22 * l) / 451;
4611      int mes = (h + l - 7 * m + 114) / 31;
4612      int dia = ((h + l - 7 * m + 114) % 31) + 1;
4613
4614      pascoa.dia = (short) dia;
4615      pascoa.mes = (short) mes;
4616      pascoa.ano = (short) ano;
4617      return pascoa;
4618  }
```

Figura 24- Função Calcular Domingo de Páscoa

```
4626  Data calcular_quarta_feira_cinzas(Data pascoa) {
4627      Data cinzas = pascoa;
4628      cinzas.dia -= DIAS_QUARESMA;
4629
4630      //Como o dia ficará negativo, temos de fazer as respetivas alterações
4631      while (cinzas.dia <= 0) {
4632          cinzas.mes--;
4633          if (cinzas.mes <= 0) {
4634              cinzas.mes = 12;
4635              cinzas.ano--;
4636          }
4637
4638          //Determinar os dias do mês anterior
4639          switch (cinzas.mes) {
4640              //Fevereiro
4641              case 2:
4642                  //Caso o ano seja bissexto
4643                  if ((cinzas.ano % 4 == 0 && cinzas.ano % 100 != 0) || (cinzas.ano % 400 == 0))
4644                      cinzas.dia += 29;
4645                  else
4646                      cinzas.dia += 28;
4647                  break;
4648              //Meses com 30 dias
4649              case 4: //Abril
4650              case 6: //Junho
4651              case 9: //Setembro
4652              case 11: //Novembro
4653                  cinzas.dia += 30;
4654                  break;
4655              //Meses com 31 dias
4656              default:
4657                  cinzas.dia += 31;
4658                  break;
4659          }
4660      }
4661
4662      return cinzas;
4663  }
```

Figura 25- Função Calcular Quarta Feira Cinzas

```
4677  void calcular_quaresma(int ano, Data * inicio, Data * fim) {
4678      if (!inicio || !fim) return;
4679
4680      //Calcular data inicial e final + 3
4681      *fim = calcular_domingo_pascoa(ano);
4682      *inicio = calcular_quarta_feira_cinzas(*fim);
4683
4684      //Quaresma acaba na quinta feira santa, logo temos de subtrair 3(domingo, sábado, sexta -> quinta)
4685      fim->dia -=3;
4686
4687      //Ajustar data
4688      if (fim->dia <= 0) {
4689          fim->mes--;
4690          //Fevereiro
4691          if (fim->mes == 2) {
4692              fim->dia += ((fim->ano % 4 == 0 && fim->ano % 100 != 0) || (fim->ano % 400 == 0)) ? 29 : 28;
4693          }
4694          //Meses com 30 dias
4695          else if (fim->mes == 4 || fim->mes == 6 || fim->mes == 9 || fim->mes == 11) {
4696              fim->dia += 30;
4697          }
4698          //31 dias
4699          else {
4700              fim->dia += 31;
4701          }
4702      }
4703  }
```

Figura 26- Função Calcular Quaresma

5. Conclusões

5.1. Bugs/Problemas

Nesta secção serão explorados alguns problemas e desafios que foram encontrados no desenvolvimento do programa, que serão listados abaixo.

Logo ao carregar os dados, foi visível um erro, especialmente após a introdução do ficheiro de erros, isto pois existia uma linha que era colocada no ficheiro de erros com metade dos parâmetros, apesar de ser efetivamente inválida. Acontece que na função separar_parametros() não estava a ser efetuada uma cópia da linha, e então estávamos a alterar o ponteiro original da linha, fazendo com que fosse cortada a meio.

Outro dos erros, também associado a ponteiros foi encontrado ao eliminar um estudante, caso em que, o free do nome e da nacionalidade estava a ser feito depois da ordenação, levando a que o elemento seguinte ao que foi eliminado ficasse com o nome e nacionalidade inválido.

No que toca aos menus e validação de inputs em geral, houve problemas com a função de limpar_buffer(), levando à criação da verificar_e_limpar_buffer(), isto pois limpar_buffer() estava a ser usada de uma forma preventiva, isto é, para limpar o buffer do teclado antes ou depois de pedir inputs, o que resultava no pedido de um enter pelo programa para continuar o programa. Isto foi abolido (passado a usar apenas quando há certezas de que o buffer está "sujo") e passou a fazer-se a validação com a segunda função, que verifica se o buffer está "sujo", e se estiver, limpa-o, caso esteja apenas o '\n', vai consumir esse caracter. Claro que continuamos a não poder usar a função da mesma forma que antes.

Ainda abordando os algoritmos de procura, apenas nos deparamos com um erro, na função de mergesort, ou melhor, na sua chamada, já que se estava a passar o tamanho dos arrays e não os seus índices, o que resultava num primeiro elemento inicializado.

Claro que os erros não se ficaram por aqui. Houve muitos mais, e provavelmente ainda há, apenas à espera de serem descobertos. E foi isso que aconteceu hoje dia 10 de janeiro, a poucas horas da entrega do trabalho, foi descoberto um pequeno bug na função de procurar_nacionalidades(). Não é algo que faça o programa crashar ou perto disso, mas certamente não o faz funcionar da maneira pretendida. Trata-se de um erro no sistema de sugestões, onde caso existam duas ou mais nacionalidades no array como "Portuguesa" e "PORTUGUESA", e se introduza "Port" na procura, o programa irá sugerir uma das duas (no caso a que se encontrar no aluno de código mais baixo), e caso se recuse, não irá sugerir a outra,

uma vez que na comparação de usam as strings normalizadas. Possivelmente a solução mais viável (e realista) seria capitalizar as nacionalidades sempre que forem introduzidas, e guardá-las como tal, mas caso contrário, seria necessário guardar a nacionalidade não normalizada no array de sugestões, e depois comparar a nova sugestão com essa string. De qualquer dos modos, não tínhamos tempo para implementar e testar o programa, pelo que fica assim detalhado a existência desse erro.

5.2. Considerações Adicionais

Esta secção funciona como uma espécie de autorreflexão. Nós temos plena noção de que há muitas coisas que são meros "extras" no trabalho e outras que poderiam ter sido mais bem abordados. Isto servirá então também para explicar muitas das escolhas e abordagens tomadas.

Nunca foi objetivo fazer uso de diversas bibliotecas, assim como diz no enunciado, que se "deve apenas usar os conhecimentos lecionados nas aulas de Algoritmos e Programação", e como tal, estávamos a pensar em fazer tudo "à unha", daí o uso de separar_parametros(). Com o tempo fomos nos apercebendo que afinal podíamos utilizar mais bibliotecas e explorar um pouco mais o trabalho, e fomos utilizando mais bibliotecas externas como o <time.h> ou o <wctype.h> e funções como o strstr, strchr, etc.

Tentámos ao máximo tornar o projeto o mais completo possível, no entanto, houve alguns aspetos que ficaram por tratar, em parte por constrangimentos de tempo.

Por exemplo, no que toca às nacionalidades, podia ter sido criado uma função para manter um array com todas as nacionalidades existentes no programa, ou até mesmo guardar num ficheiro, e ir atualizando a cada inserção, de modo a facilitar a procura das mesmas. Isto inclusivamente tornaria o programa mais eficiente, especialmente caso fossem acrescentadas mais funcionalidades envolvendo nacionalidades.

Outro dos aspetos que podia ser melhorado é a repetição de código. Muito provavelmente seria possível diminuir o tamanho de algumas secções, principalmente em carregar_dados_txt() e em inserir_estudante(), onde temos muitos ciclos do {}while com parâmetros iguais mas mensagens de erro diferentes. Seria possível agrupar isso numa função, mas como este caso há mais, e agora que se terminou o programa e o observamos, cada vez sobressai mais certas partes repetidas, que certamente, se o refizéssemos, não seriam feitas daquela forma.

Tocando agora nos pontos positivos, o programa consegue entregar as funcionalidades pedidas de forma bonita e prática. Conseguimos o essencial e depois deixámos a criatividade assumir e desenvolver ainda mais o programa, como pela adição do checksum ou das sugestões na nacionalidade. Também achamos estar bastante completo, com a sua portabilidade para

Windows e sistemas Unix (apesar de não ter sido testado em macOS, espera-se que funcion nesse sistema) e verificações rigorosas e oportunas como se um ficheiro já existe ou não ao pedir uma listagem.

Apesar disto, ainda gostávamos de ter acrescentado mais algumas funcionalidades, como a possibilidade de efetuar a listagem em formato .md (markdown) ou a de possuir um sistema de paginação com opções de retroceder, avançar, ou ir para página "X".

Fica também a preocupação relativa aos menus. Estão desenvolvidos de forma a serem bonitos, mas há sempre a possibilidade de não funcionarem em certos casos. No Windows, será difícil, mas em macOS ou Linux, talvez aconteça, no entanto, mesmo que o menu funcionasse, o programa em geral iria mal funcionar por apresentar caracteres com acentos ou 'ç' nas mensagens de erros, menus, etc. Caso os menus não funcionem de todo, foi pensado na possibilidade de criar outro menu com caracteres diferentes dentro de cada função, que seria ativado por uma condição if de uma variável global que iria buscar o resultado da normalização UTF-8.

Ainda na questão dos acentos e do 'ç', temos imensa pena de não ter conseguido colocar a função normalizar_string() a funcionar corretamente (uma vez que não consegue retirar acentos e 'ç' das strings). A possível solução para esse problema, muito provavelmente passaria por usar uma biblioteca externa, como o <iconv.h> (para evitar refazer o programa), uma vez que o melhor seria utilizar wchar_t para armazenar este tipo de caracteres (melhor solução).

Por fim, fica a informação que o programa foi desenvolvido e compilado utilizando a versão de C23, com o compilador GCC (gcc -Wall -Wextra -g -O0 -std=c23 -o programa main.c funcoes.c).

Por muito pena nossa, o programa não funcionou em Linux, porém o erro já está identificado, apenas não temos tempo para o resolver. Acontece que o programa está a funcionar perfeitamente, no entanto, ao carregar os dados de ficheiros .TXT, todos são tratados como inválidos, uma vez que no Windows, os ficheiros terminam com "\r\n" e no Linux apenas "\n", o que leva a leitura a ser considerada inválida. Será posteriormente colocado o fix no repositório do GitHub, que estará público até à entrega das notas do trabalho.

5.3. Considerações finais

O desenvolvimento do presente programa constituiu uma oportunidade única para aplicar, de forma prática, conceitos fundamentais de algoritmia e programação, reforçando não apenas as competências técnicas, mas também a capacidade de análise e resolução de problemas.

Durante o processo de implementação, foram enfrentados diversos desafios, desde a resolução de erros até à tomada de decisões estruturais que influenciaram diretamente a funcionalidade e a robustez do sistema, como já foi mencionado.

Um dos principais conhecimentos adquiridos foi a compreensão da importância de uma validação boa e rigorosa e de um tratamento meticuloso dos dados, especialmente no que concerne à interação com o utilizador e à manipulação de ficheiros. Adicionalmente, a adoção de abordagens como a cópia de dados, para salvaguardar a integridade das operações realizadas, revelou-se fundamental para resolver problemas mais complexos.

Embora o programa cumpra os requisitos estabelecidos e demonstre um desempenho satisfatório, reconhecemos que existem oportunidades de melhoria. Áreas como a otimização do código, uma maior modularidade e a expansão das funcionalidades poderiam ser exploradas em versões futuras. De igual modo, algumas decisões tomadas durante o desenvolvimento visaram simplificar a implementação, em alinhamento com os conhecimentos adquiridos até ao momento, mas deixam em aberto possibilidades de aperfeiçoamento.

Em síntese, este projeto constituiu um marco relevante para a consolidação dos conhecimentos adquiridos em algoritmia e programação, ao mesmo tempo que evidenciou as complexidades inerentes ao desenvolvimento de soluções computacionais completas. Apesar das limitações identificadas, considera-se que os objetivos principais foram alcançados, sendo o resultado final reflexo do esforço e dedicação aplicados em todas as fases de desenvolvimento.

6. Referências

(GeeksforGeek, s.d.) (consultado a 10-12-2024)

(WordPress, s.d.) (consultado a 19-12-2024)

(GeeksforGeeks, s.d.) (consultado a 26-12-2024)

(GeeksforGeeks, s.d.) (consultado a 23-12-2024)

(Matemática.PT, s.d.) (consultado a 29-12-2024)

(GeeksforGeeks, s.d.) (consultado a 2-1-2025)