



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

Unidade Curricular: Algoritmos e Programação

Relatório Relativo ao MiniProjecto I

Tema: Gestão de Exames

Realizado por: António Sousa – pv26160

Filipe Fonseca – pv22137

Inês Melo – pv22657

Pedro Rodrigues – pv24512

Viseu, 2023

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

Relatório relativo ao MiniProjecto I
Curso de Licenciatura em Engenharia Informática
Unidade Curricular de Algoritmos e Programação

Gestão de Exames

Ano Letivo 2022/23

Viseu, 2023

ÍNDICE

| | |
|--|-----------|
| 1. Introdução | 1 |
| 2. Início da implementação | 2 |
| 2.1. Abordagem inicial | 2 |
| 3. Implementação da aplicação | 3 |
| 3.1. Implementação do ficheiro ProjectoFinal.c | 4 |
| 3.2. Implementação do ficheiro alunos.c | 5 |
| 3.3. Implementação do ficheiro UnidadesCurriculares.c | 7 |
| 3.4. Implementação do ficheiro exames.c e inscricao_exames.c | 8 |
| 3.5. Implementação do ficheiro salas.c | 10 |
| 3.6. Implementação do header functions.h | 12 |
| 3.7. Estruturas de dados | 13 |
| 3.8. Validações | 14 |
| 3.8.1. Validações em alunos.c | 14 |
| 3.8.2. Validações em UnidadesCurriculares.c | 15 |
| 3.8.3. Validações em exames.c | 17 |
| 3.8.4. Validações em salas.c | 18 |
| 4. Conclusões | 19 |

Índice de Figuras

| | |
|--|----|
| Figura 1 - Esquema de desenvolvimento do projeto | 2 |
| Figura 2 - Menu principal..... | 3 |
| Figura 3 – Código Menu Principal | 3 |
| Figura 4 - Includes e defines..... | 4 |
| Figura 5 - Carregamento dos ficheiros de txt para a memória | 4 |
| Figura 6 - Importação do ficheiro de texto alunos..... | 5 |
| Figura 7 - Exportação para o ficheiro de texto alunos..... | 5 |
| Figura 8 - Função para inserir novo aluno..... | 6 |
| Figura 9 - Menu alunos..... | 6 |
| Figura 10 - Função para listar Unidades Curriculares | 7 |
| Figura 11 - Função para criar Unidade Curricular | 7 |
| Figura 12 - Função que verifica os dias entre exames..... | 8 |
| Figura 13 - Função para criar exame (excerto)..... | 9 |
| Figura 14 - Função para listar exames de um determinado aluno | 9 |
| Figura 15 - Função que edita informação sobre salas | 10 |
| Figura 16 - Função para apagar salas com validações..... | 11 |
| Figura 17 - Definição de estruturas | 12 |
| Figura 18 - Listagem de algumas funções | 12 |
| Figura 19 - Definição da estrutura UNIDADECURRICULAR e EXAMES..... | 13 |
| Figura 20 - Validações da existência do regime, ano de matrícula e aluno | 14 |
| Figura 21 - Validação da existência de inscrição em exame | 15 |
| Figura 22 - Validações Unidades Curriculares..... | 15 |
| Figura 23 - Validações para apagar Unidade Curricular | 16 |
| Figura 24 - Validação para apagar exame | 17 |
| Figura 25 - Validação da data inserida | 17 |
| Figura 26 - Validação que verifica se a sala existe..... | 18 |
| Figura 27 – Validação do número da sala e se pode ser apagada..... | 18 |

1. Introdução

Este relatório é realizado no âmbito da Unidade Curricular de Algoritmos e Programação, na qual foi solicitado o desenvolvimento de uma aplicação para a gestão de exames do Departamento de Informática, implementada em linguagem C. Nas próximas páginas pretende-se demonstrar o trabalho realizado ao longo do desenvolvimento bem como uma explicação das funcionalidades da aplicação.

Este documento está organizado em 4 capítulos que se seguem a esta introdução.

No segundo capítulo abordamos o início da implementação e uma breve explicação de como foi elaborada.

Segue-se o capítulo três, onde se detalha a implementação da aplicação.

Termina-se com o capítulo 4, onde se apresentam as conclusões deste trabalho.

2. Início da implementação

Este capítulo aborda o início da implementação da aplicação e o método da mesma.

Como referido anteriormente, no enunciado do Mini Projeto Prático, pretende-se a implementação de uma aplicação para a Gestão de Exames do Departamento de Informática da Escola de Gestão e Tecnologia de Viseu.

Na implementação da aplicação foi solicitado que fossem usados os conhecimentos obtidos durante as aulas de Algoritmos e Programação, por exemplo o uso de ficheiros de texto e uso de estruturas de dados, entre outros.

Para a implementação da aplicação foi usado o editor de código Visual Studio Code.

2.1. Abordagem inicial

Para implementar a aplicação foi decidido dividir as várias funcionalidades da mesma. Foram criados os ficheiros relativos às unidades curriculares (UnidadesCurriculares.c), exames (exames.c), alunos (alunos.c), salas (salas.c), épocas (epocas.c), feriados (feriados.c), um ficheiro inicial com o menu (ProjectoFinal.c) e um outro com listagem das funções usadas na aplicação bem como da definição das estruturas (funcions.h).

Para o armazenamento de dados foram criados os ficheiros de texto relativos aos alunos, cursos, épocas de exame, calendário de exames, inscrições nos exames, feriados, regimes de estudante, salas e unidades curriculares.

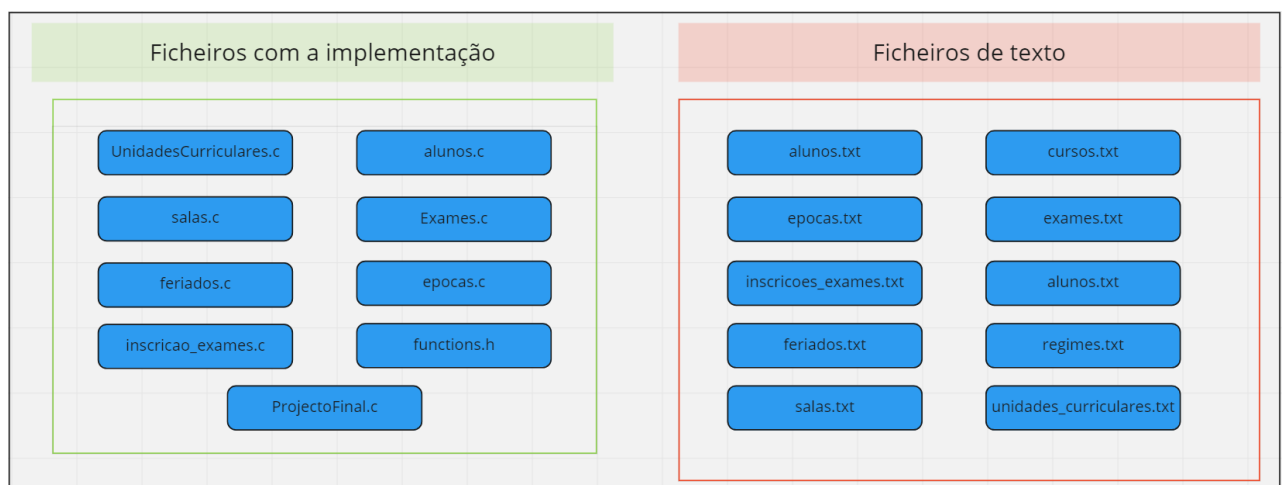


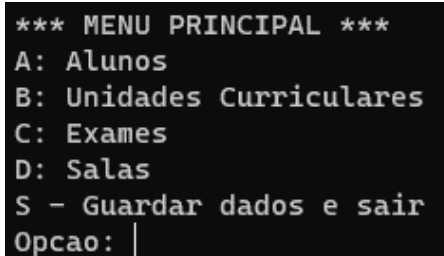
Figura 1 - Esquema de desenvolvimento do projeto

3. Implementação da aplicação

Como referido acima, foram criados os ficheiros com as implementações das várias funcionalidades.

No ficheiro “inicial” (ProjectoFinal.c) é implementado o menu principal da aplicação onde são solicitadas ao utilizador as opções relativas a:

- Informação relativa a alunos;
- Informação relativa às unidades curriculares;
- Informação relativa a exames;
- Informação relativa a salas;



```
*** MENU PRINCIPAL ***
A: Alunos
B: Unidades Curriculares
C: Exames
D: Salas
S - Guardar dados e sair
Opcao: |
```

Figura 2 - Menu principal



```
/* Termina o carregamento de variaveis em memoria */
while (1) {
    printf("\n\n");
    printf("*** MENU PRINCIPAL ***\n");
    printf("A: Alunos\n");
    printf("B: Unidades Curriculares\n");
    printf("C: Exames\n");
    printf("D: Salas\n");
    printf("S - Guardar dados e sair\n");
    printf("Opcao: ");
    scanf("%c", &escolha);
    system("cls");
    switch (escolha)
    {
        case 'A': // ALUNOS
            menu_alunos(alunos, regimes, cursos, inscricoes_exames);
            break;

        case 'B': // menu das unidades curriculares
            menu_uc(uc, cursos, exames);
            break;

        case 'C': // menu de exames
            menu_exames(exames, inscricoes_exames, alunos, salas, epocas, uc, cursos);
            break;
        case 'D': // menu salas
            menu_salas(salas, exames);
            break;
        case 'S': //grava tudo nos ficheiros
            export_salas(salas);
            export_exames(exames);
            export_UC(uc);
            export_feriados(feriado);
            export_alunos(alunos);

            break;
    }

    // Limpeza da memoria
    free(uc);
    free(cursos);
    free(feriado);
    free(exames);
    free(alunos);
    free(salas);
    free(epocas);
}
```

Figura 3 – Código Menu Principal

3.1. Implementação do ficheiro ProjectoFinal.c

Este ficheiro é o ponto de partida da aplicação. Nele estão as diretivas define e include dos ficheiros .c usados, bem como as funções que carregam em memória todos os ficheiros de texto necessários ao correto funcionamento da aplicação. No seu código possui ainda a função main com o menu inicial.

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "functions.h"
#include "feriados.c"
#include "UnidadesCurriculares.c"
#include "exames.c"
#include "salas.c"
#include "alunos.c"
#include "epocas.c"

#define MAX_EXAMES_FILE 100
#define STRING char *
#define MAX_FERIADOS 20
#define MAX_UNIDADES_CURRICULARES 100
#define MAX_CURSOS 20
#define MAX_ALUNOS 3000
#define MAX_SALAS 100
#define MAX_REGIMES 5
#define MAX_EPOCAS 10
#define MAX_INSCRICOES 3000
```

Figura 4 - Includes e defines

```
//Função Principal do programa
void main(int argc, char** argv) {
    STRING* V;
    char escolha;
    /* Carregamento em memoria de todos os ficheiros txt do projecto*/

    //carrega os dados das epocas
    EPOCAS* epocas = (EPOCAS*)malloc(MAX_SALAS * sizeof(EPOCAS));
    import_txt_epocas(epocas, V);

    //carrega o ficheiro dos feriados em memoria
    FERIADOS* feriado = (FERIADOS*)malloc(MAX_FERIADOS * sizeof(FERIADOS));
    import_feriados(feriado, V);

    //carrega o ficheiro das UC em memoria
    UNIDADECURRICULAR* uc = (UNIDADECURRICULAR*)malloc(MAX_UNIDADES_CURRICULARES * sizeof(UNIDADECURRICULAR));
    import_txt_uc(uc, V);

    //carrega o ficheiro dos cursos em memoria
    CURSO* cursos = (CURSO*)malloc(MAX_CURSOS * sizeof(CURSO));
    import_txt_cursos(cursos, V);

    //carrega os dados dos exames em memoria
    EXAMES* exames = (EXAMES*)malloc(MAX_EXAMES_FILE * sizeof(EXAMES));
    import_txt_exames(exames, V, uc);

    //carrega os dados das inscricoes dos exames em memoria
    INSCRICOESEXAMES* inscricoes_exames = (INSCRICOESEXAMES*)malloc(MAX_INSCRICOES * sizeof(INSCRICOESEXAMES));
    import_txt_inscricoes_exames(inscricoes_exames, V);

    //carrega os dados dos regimes em memoria
    REGIMES* regimes = (REGIMES*)malloc(MAX_REGIMES * sizeof(REGIMES));
    import_txt_regimes(regimes, V);

    //carrega os dados dos alunos em memoria
    ALUNOS* alunos = (ALUNOS*)malloc(MAX_ALUNOS * sizeof(ALUNOS));
    import_txt_alunos(alunos, V);

    //carrega os dados das salas
    SALAS* salas = (SALAS*)malloc(MAX_SALAS * sizeof(SALAS));
    import_txt_salas(salas, V);
}
```

Figura 5 - Carregamento dos ficheiros de txt para a memória

3.2. Implementação do ficheiro alunos.c

Esta implementação permite consultar informação, relativa aos alunos, bom como criar, apagar e editar essa informação.

Esta implementação recorre aos ficheiros de texto alunos.txt e cursos.txt.

```
//passa para o vecto a informacao do ficheiro alunos.txt
void import_txt_alunos(ALUNOS* alunos, STRING* V) {
    int i = 0, k, n_campos_lidos;

    //Abre o ficheiro
    FILE* f = fopen("alunos.txt", "r");
    if (f == NULL) {
        printf("Erro ao abrir ficheiro alunos.txt\n");
        exit(1);
    }

    do {
        V = Read_Split_Line_File(f, &n_campos_lidos);
        if (V != NULL) { // caso consigamos ler alguma informacao

            alunos[i].nome = (char*)malloc(sizeof(char) * (strlen(V[0]) + 1));
            strcpy(alunos[i].nome, V[0]);

            alunos[i].regime = (char*)malloc(sizeof(char) * (strlen(V[1]) + 1));
            strcpy(alunos[i].regime, V[1]);

            alunos[i].ano_matricula = atoi(V[2]);

            alunos[i].numero = atoi(V[3]);

            alunos[i].curso = (char*)malloc(sizeof(char) * (strlen(V[4]) + 1));
            strcpy(alunos[i].curso, V[4]);

            alunos[i].ocupado = 1;

            for (k = 0; k < n_campos_lidos; k++)
                free(V[k]);
            free(V);
            i++;
        }
    } while (!feof(f));

    fclose(f);
}
```

Figura 6 - Importação do ficheiro de texto alunos

```
//export
void export_alunos(ALUNOS* alunos) {
    int i, k = 0;
    FILE *f;

    f = fopen("alunos.txt", "w");
    if (f == NULL) {
        printf("Erro ao abrir o ficheiro alunos.txt");
        exit(1);
    }

    for (i = 0; i < MAX_UNIDADES_CURRICULARES; i++)
    {
        if (alunos[i].ocupado == 1) {
            fprintf(f, "%s|%s|%d|%d|%s",
                alunos[i].nome,
                alunos[i].regime,
                alunos[i].ano_matricula,
                alunos[i].numero,
                alunos[i].curso
            );
        }
    }

    fclose(f);
}
```

Figura 7 - Exportação para o ficheiro de texto alunos

```

//
int get_posicao_vect_alunos(ALUNOS* alunos) {
    int i;
    for (i = 0; i < MAX_ALUNOS; i++)
    {
        if ((alunos[i].ocupado == 0) && (i < MAX_ALUNOS)) {
            return i;
        }
    }
    return -1;
}

//Insere aluno
int insere_aluno(ALUNOS* alunos, char* nome, char* regime, int ano_matricula, int numero, char* curso) {
    int posicaoVazia = 0;
    posicaoVazia = get_posicao_vect_alunos(alunos);
    if (posicaoVazia == -1) {
        return -1;
    }

    //insere o novo aluno no vector
    alunos[posicaoVazia].nome = nome;
    alunos[posicaoVazia].regime = regime;
    alunos[posicaoVazia].ano_matricula = ano_matricula;
    alunos[posicaoVazia].numero = numero;
    alunos[posicaoVazia].curso = curso;
    alunos[posicaoVazia].ocupado = 1;

    return 1;
}

```

Figura 8 - Função para inserir novo aluno

```

//menu relativo aos alunos
void menu_alunos(ALUNOS* alunos, REGIMES* regimes, CURSO* cursos, INSCRICOESEXAMES* inscricoes_exames) {
    int opcao = -1;

    while (opcao != 0) {
        printf("\n");
        printf("Bem-vindo ao menu dos alunos!\n");
        printf("Escolha uma das opcoes:\n");
        printf("1 - Consultar Alunos\n");
        printf("2 - Adicionar Aluno\n");
        printf("3 - Editar Aluno\n");
        printf("4 - Apagar Aluno\n");
        printf("0 - Sair\n");
        printf("Introduza a Opcao: ");
        scanf("%d", &opcao);
        system("cls");

        switch (opcao) {
            case 1: //lista os alunos
                listar_alunos(alunos);
                break;
            case 2: //cria alunos
                criar_aluno(alunos, regimes, cursos);
                break;
            case 3: //Editar aluno
                editar_aluno(alunos, inscricoes_exames, regimes, cursos);
                break;
            case 4: //Apagar aluno
                apagar_aluno(alunos, inscricoes_exames);
                break;
            case 0:
                break;
        }
    }
}

```

Figura 9 - Menu alunos

3.3. Implementação do ficheiro UnidadesCurriculares.c

Este ponto permite, também, consultar informação relativa às várias unidades curriculares dos vários anos, semestres e cursos, bem como criar, apagar e editar essa mesma informação contida no ficheiro de texto unidades_curriculares.txt.

```
// funcao que lista as unidades curriculares
void listar_UC(UNIDADECURRICULAR* uc) {
    int i;
    printf("%-2s %s %40s %17s %7s %10s\n", "Cod", "Nome", "Docente", "Curso", "Ano", "Semestre");
    for (i = 0; i < MAX_UNIDADES_CURRICULARES; i++)
    {
        if (uc[i].ocupado == 1) {
            printf("%-3d %-37s %-19s %s %d %d\n", uc[i].codigo, uc[i].descricao, uc[i].docente, uc[i].curso, uc[i].ano, uc[i].semestre);
        }
    }
}
```

Figura 10 - Função para listar Unidades Curriculares

```
void criar_UC(UNIDADECURRICULAR* uc, CURSO* cursos) {
    //vamos pedir ao utilizador os dados para criar a nova unidade curricular
    //declaracao de variaveis necessarias
    int codigo = 0;
    char* descricao;
    descricao = (char *)malloc(sizeof(char) * 100);
    char* docente;
    docente = (char *)malloc(sizeof(char) * 100);
    int posicaoCurso;
    int ano;
    int semestre;
    //vamos pedir o nome da unidade curricular
    do
    {
        printf("Qual o nome da unidade curricular?\n");
        scanf("%s", descricao);
    } while (strlen(descricao) == 0);

    //vamos pedir o nome do docente
    do
    {
        printf("Qual o nome do docente?\n");
        scanf("%s", docente);
    } while (strlen(docente) == 0);

    //selecao do curso
    do {
        listar_cursos(cursos);
        printf("Indique o curso da lista acima\n");
        scanf("%d", &posicaoCurso);
    } while (valida_curso_escolhido(cursos, posicaoCurso) == 0);

    // introducao do ano e do semestre
    do{
        printf("Indique o ano (1,2,3) e o semestre (1,2) da unidade curricular\n");
        scanf("%d %d", &ano, &semestre);
    }while (valida_ano_semestre(ano, semestre) == 0);

    //valida se a unidade curricular já existe no nosso vector
    //No caso de retornar 1 a UC ja existe, caso contrario podemos inserir
    if (valida_UC_existe_vector(uc, descricao, cursos[posicaoCurso].codcurso) == 1) {
        printf("A unidade curricular já existe!\n");
    } else {
        if (insere_uc(uc, descricao, docente, cursos[posicaoCurso].codcurso, ano, semestre) == 1) {
            printf("Unidade curricular inserida com sucesso!\n");
            printf("\n");
        } else {
            printf("Ocorreu um erro ao guardar a unidade curricular em memoria\n");
            printf("Tente novamente!\n");
            printf("\n");
        }
    }
    free(descricao);
    free(docente);
}
```

Figura 11 - Função para criar Unidade Curricular

3.4. Implementação do ficheiro exames.c e inscricao_exames.c

Esta implementação é o foco principal da aplicação, pois permite efetuar a marcação de exames para as várias unidades curriculares, bem como inscrição dos alunos nos mesmos.

Decidimos dividir em dois ficheiros (exames.c e inscricao_exames.c)

Tendo em conta as diversas particularidades, como exames poderem ser de três tipos: época normal, de recurso ou época especial, sendo que os exames de época normal e recurso podem ser realizados por qualquer aluno e os exames de época especial estão acessíveis apenas a alunos com estatutos de trabalhador-estudante, atletas, dirigentes associativos ou alunos que frequentem o último ano do curso (corresponde ao 3ºano de matrícula). Não permite marcação de exames de um mesmo ano curricular com uma diferença de dias inferior a 3, verifica que a data marcada não é fim de semana ou feriado e a referida marcação terá de ser dentro do período definido para cada época.

Para esta implementação foram usados os ficheiros de texto exames.txt e inscricoes_exame.txt.

```
// funcao valida se existem exames marcados com 3 dias de intervalo
int numero_dias_entre_exames(EXAMES* exames_bv, DATA* datainserida, char* curso, int ano) {
    int i = 0;
    for (i = 0; i < MAX_EXAMES_FILE; i++)
    {
        if (exames_bv[i].ocupado == 1 && exames_bv[i].realizado == 0) {
            struct tm tm1 = { 0 };
            struct tm tm2 = { 0 };
            DATA* data_exame = (DATA*)malloc(sizeof(DATA));
            coloca_data_em_struct(exames_bv[i].data, data_exame);

            time_t start_standard, end_standard;
            struct tm start_date = { 0 };
            struct tm end_date = { 0 };
            double diff;

            start_date.tm_year = datainserida->ano;
            start_date.tm_mon = datainserida->mes;
            start_date.tm_mday = datainserida->dia;
            start_date.tm_hour = 10;
            start_date.tm_min = 00;
            start_date.tm_sec = 00;

            end_date.tm_mday = data_exame->dia;
            end_date.tm_mon = data_exame->mes;
            end_date.tm_year = data_exame->ano;
            end_date.tm_hour = 10;
            end_date.tm_min = 00;
            end_date.tm_sec = 00;

            /* first with standard time */
            start_date.tm_isdst = 0;
            end_date.tm_isdst = 0;
            start_standard = mktime(&start_date);
            end_standard = mktime(&end_date);
            diff = difftime(start_standard, end_standard);

            if (diff > -3 && diff < 3) {
                return 0;
            }
            free(data_exame);
        }
    }
    return 1;
}
```

Figura 12 - Função que verifica os dias entre exames

Na figura seguinte temos parte da implementação da função criar_Exame(...) que permite ao utilizador criar uma data para um determinado exame. Podemos verificar também que a função apresenta as validações necessárias para um funcionamento mais eficiente da

aplicação. Essas validações são a verificação se os dados relativos à época, curso, unidade curricular e a data foram inseridos corretamente.

```
// funcao que cria novos exames
void criar_Exame(EXAMES* exames_bv, ALUNOS* aluno, UNIDADECURRICULAR* uc, SALAS* salas, EPOCAS* epocas, CURSOS* cursos, FERIADOS* feriados_datas) {
    int opcaoepoca = -1;
    int duracao_exame = 0;
    int opcaoUC = -1;
    int posicaoCurso = 0;
    int semestre = 0;
    char* unidade_curricular_escolhida;
    char* curso;
    char* epoca = (char*)malloc(sizeof(char) * 10);
    curso = (char*)malloc(sizeof(char) * 10);
    DATA* data_inicio_epoca = (DATA*)malloc(sizeof(DATA));
    DATA* data_fim_epoca = (DATA*)malloc(sizeof(DATA));

    //nesta funcao vamos criar um novo exame
    do
    {
        printf("\n*** SELECAO DE EPOCA ***\n");
        lista_epocas(epocas);
        printf("Indique o codigo da epoca\n");
        opcaoepoca = 0;
        scanf("%i", &opcaoepoca);
        if (valida_epoca(epocas, opcaoepoca) == -1) {
            opcaoepoca = -1;
        }
        else {
            opcaoepoca = valida_epoca(epocas, opcaoepoca);
        }
    } while (opcaoepoca == -1);

    //coloca as datas de inicio e fim da epoca numa struct
    coloca_data_em_struct(epocas[opcaoepoca].dataInicio, data_inicio_epoca);
    coloca_data_em_struct(epocas[opcaoepoca].dataFim, data_fim_epoca);
    semestre = epocas[opcaoepoca].semestre;
    epoca = epocas[opcaoepoca].epoca;

    printf("\nSelecionou a epoca %s que vai de %s ate %s\n", epoca, epocas[opcaoepoca].dataInicio, epocas[opcaoepoca].dataFim);
    // if(strcmp(epoca, "Especial") == 0) {
    //     printf("es");
    // }
    // else {
    printf("Qual o curso para o qual quer marcar exame?\n");
    do {
        listar_cursos(cursos);
        printf("Indique o curso da lista acima\n");
        scanf("%d", &posicaoCurso);
    } while (valida_curso_escolhido(cursos, posicaoCurso) == 0);

    curso = cursos[posicaoCurso].codcurso;

    // Vamos escolher a UC
    do
    {
        listar_UC_curso_semestre(uc, curso, semestre);
        printf("\nIndiqu o codigo da unidade curricular\n");
        opcaoUC = -1;
        scanf("%d", &opcaoUC);
    } while (valida_UC_curso_semestre(uc, curso, semestre, opcaoUC) == -1);
}
```

Figura 13 - Função para criar exame (excerto)

```
//lista as inscricoes por aluno
void lista_inscricoes_aluno(INSCRICOESEXAMES* inscricoes_exames){
    int i;
    int opcaoAluno = 0;

    printf("Insira o numero do aluno: ");
    scanf("%d", &opcaoAluno);

    if(valida_aluno_inscrito(inscricoes_exames, opcaoAluno)==1){
        printf("%s %s %s %s\n", "ID", "Unidade Curricular", "Numero Aluno", "Ano Matricula", "Epoca", "Regime\n");
        for (i = 0; i < MAX_INSCRICOES; i++)
        {
            if (inscricoes_exames[i].numero_aluno == opcaoAluno) {
                printf("%d %s %d %d %s %s\n", inscricoes_exames[i].codigo, inscricoes_exames[i].unidade_curricular, inscricoes_exames[i].numero_aluno,
                inscricoes_exames[i].ano_matricula, inscricoes_exames[i].epoca, inscricoes_exames[i].regime);
            }
        }
    }
}
```

Figura 14 - Função para listar exames de um determinado aluno

3.5. Implementação do ficheiro salas.c

Esta implementação permite ao utilizador consultar as salas existentes, bem como editar, apagar e criar salas. Esta informação está contida no ficheiro de texto salas.txt.

```
//funcao que edita dados das salas
void editar_sala(SALAS* salas, EXAMES* exames) {
    int opcaoSala;
    int IDSala;

    //lista as salas
    listar_salas(salas);
    //pede ao utilizador o id da sala
    do
    {
        printf("Introduza o ID da sala (dentro das opcoes acima)\n");
        printf("Sala:");
        scanf("%d", &opcaoSala);
        IDSala = valida_cod_sala(salas, opcaoSala);
        if (IDSala == -1) {
            printf("O numero %d nao se encontra na lista!\n", opcaoSala);
        }
    } while (IDSala == -1);

    //Valida de existe algum exame para a unidade curricular
    if (valida_delete_sala(exames, salas[opcaoSala].codigo) == 0) {
        printf("A sala %s nao pode ser editada, pois ja tem exames marcados\n", salas[IDSala].nome_sala);
    }
    else {
        fflush(stdin);
        char* codigo;
        codigo = (char *)malloc(sizeof(char) * 100);
        char* nome_sala;
        nome_sala = (char *)malloc(sizeof(char) * 100);
        int lotacao;
        do
        {
            printf("Indique o novo numero da sala\n");
            scanf("%s", codigo);
        } while (strlen(codigo) == 0);

        //vamos pedir o nome da sala
        do
        {
            printf("Qual o nome da sala?\n");
            scanf("%s", nome_sala);
        } while (strlen(nome_sala) == 0);

        //lotacao da sala
        do {
            printf("Indique lotacao\n");
            scanf("%d", &lotacao);
        } while (lotacao == 0);

        //valida se a sala ja existe no nosso vector
        //No caso de retornar 1 a sala ja existe, caso contrario podemos inserir
        if (valida_sala_existe_vector(salas, nome_sala) == 1) {
            printf("A sala ja existe!\n");
        } else {
            salas[IDSala].codigo = codigo;
            salas[IDSala].nome_sala = nome_sala;
            salas[IDSala].lotacao = lotacao;
            salas[IDSala].ocupado = 1;

            printf("Sala editada com sucesso!\n\n");
            free(codigo);
            free(nome_sala);
        }
    }
}
```

Figura 15 - Função que edita informação sobre salas

```

//valida se a sala não está definida para algum exame
int valida_delete_sala(EXAMES* exames, char* codigo) {
    int i = 0;
    for ( i = 0; i < MAX_EXAMES_FILE; i++)
    {
        if (exames[i].ocupado == 1) {
            if ((exames[i].sala == codigo)) {
                return 0;
            }
        }
    }
    return 1;
}

//Funcao que elimina a sala com validacoes
void apagar_salas(SALAS* salas, EXAMES* exames){
    int opcaoSala;
    int IDSala = 0;
    //lista as salas
    listar_salas(salas);
    //pede ao utilizador o numero da sala
    do
    {
        printf("Introduza o ID da sala (dentro das opcoes acima): \n");
        printf("ID Sala:");
        scanf("%d", &opcaoSala);
        IDSala = valida_cod_sala(salas, opcaoSala);
        if (IDSala == -1) {
            printf("O ID %d nao se encontra na lista!\n\n", opcaoSala);
        }
    } while (IDSala == -1);

    //Valida de existe algum exame para a sala
    if (valida_delete_sala(exames, salas[IDSala].codigo) == 0) {
        printf("A sala %s nao pode ser eliminada, pois ja tem exames marcados\n", salas[IDSala].nome_sala);
    }
    else {
        salas[IDSala].ocupado = 0;
        salas[IDSala].id = 0;
        salas[IDSala].codigo = 0;
        salas[IDSala].nome_sala = "";
        salas[IDSala].lotacao = 0;
        printf("Registo eliminado com sucesso!\n\n");
    }
}

```

Figura 16 - Função para apagar salas com validações

3.6. Implementação do header functions.h

Neste ficheiro estão definidas as estruturas usadas no desenvolvimento da aplicação, bem como as funções utilizadas. Foi usada esta implementação pois é necessário usar a mesma função em mais do que um ficheiro e assim não há necessidade de a repetir tornando-se o código mais eficiente funcionando este ficheiro como uma biblioteca de funções.

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H
#define STRING char *

#include <locale.h>

/**INICIO Da declaracao de estruturas**/
//Definicao de estrutura de feriados
typedef struct { ... } FERIADOS;

typedef struct { ... } EPOCAS;

typedef struct { ... } CURSO;

//definicao da estrutura de unidade curricular
typedef struct
{
    int codigo;
    char* descricao;
    char* curso;
    char* docente;
    int ano;
    int semestre;
    int ocupado;
} UNIDADECURRICULAR;

//definicao da estrutura de exames
typedef struct
{
    int codigo;
    char* unidade_curricular;
    int ano;
    char* curso;
    char* epoca;
    char* data;
    char* hora;
    int duracao;
    char* sala;
    int alunos_inscritos;
    int realizado;
    int ocupado;
} EXAMES;

//definicao da estrutura regimes
typedef struct { ... } REGIMES;

//definicao da estrutura alunos
typedef struct {
    char* nome;
    char* regime; //ID dos regimes = 1 (normal), 2 (trabalhador estudante), 3 (atleta), 4 (dirigente associativo), 5 (Erasmus)
    int ano_matricula;
    int numero;
    char* curso;
    int ocupado;
} ALUNOS;
```

Figura 17 - Definição de estruturas

```
void listar_exames(EXAMES* exames, int jarealizados);
void import_txt_alunos(ALUNOS* alunos, STRING* V);
void import_txt_regimes(REGIMES* regimes, STRING* V);
void listar_regimes(REGIMES* regimes);
void listar_alunos(ALUNOS* alunos);
int valida_regime_escolhido(REGIMES* regimes, int posicaoRegime);
int valida_ano_matricula(int ano_matricula);
int valida_aluno_existe(ALUNOS* alunos, int numero);
int get_posicao_vect_alunos(ALUNOS* alunos);
int insere_aluno(ALUNOS* alunos, char* nome, char* regime, int ano_matricula, int numero, char* curso);
void criar_aluno(ALUNOS* alunos, REGIMES* regimes, CURSO* cursos);
int valida_delete_aluno(INSCRICOSEXAMES* inscricoes_exames, ALUNOS* alunos, int numero_aluno, int opcao_aluno);
void apagar_aluno(ALUNOS* alunos, INSCRICOSEXAMES* inscricoes_exames);
void editar_aluno(ALUNOS* alunos, INSCRICOSEXAMES* inscricoes_exames, REGIMES* regimes, CURSO* cursos);
void export_alunos(ALUNOS* alunos);
void menu_alunos(ALUNOS* alunos, REGIMES* regimes, CURSO* cursos, INSCRICOSEXAMES* inscricoes_exames);
void import_txt_salas(SALAS* salas, STRING* V);
void listar_salas(SALAS* salas);
int valida_sala_existe_vector(SALAS* salas, char* nome_sala);
int get_newID_sala(SALAS* salas);
```

Figura 18 - Listagem de algumas funções

3.7. Estruturas de dados

Como referido no ponto anterior, foram usadas várias estruturas no desenvolvimento da aplicação e que se tornam necessárias para o bom funcionamento da mesma.

Foram usadas as seguintes estruturas de dados:

- FERIADOS: Onde estão definidos o dia e mês de cada feriado para auxiliar nas validações ao efetuar uma marcação de um exame;
- EPOCAS: Onde estão definidos os nomes de cada época (normal, recurso e especial), as datas de início e de fim e a que semestre pertencem;
- CURSO: Onde estão definidos os cursos e as iniciais dos mesmos;
- UNIDADECURRICULAR: Onde estão definidos os nomes da cada unidade curricular, a que curso pertencem, o docente responsável e o ano e semestre a que pertencem;
- EXAMES: Onde estão definidas as unidades curriculares, ano, curso, época, data e hora, duração, sala em que se realizam e o numero de alunos inscritos;
- REGIMES: Onde estão definidos os nomes dos regimes (Normal, trabalhador-estudante, dirigente associativo, atleta e Erasmus);
- ALUNOS: Onde estão definidos os nomes, regimes, ano de matrícula, número e curso dos alunos;
- SALAS: Onde estão definidos os números de salas, nomes, e respetiva lotação;
- INSCRICOESEXAMES: Onde estão definidas as unidades curriculares, o número de aluno inscrito, o ano de matrícula e a época para a qual está inscrito.
- DATA: Onde estão definidas as datas para a realização dos exames.

```
//definicao da estrutura de unidade curricular
typedef struct
{
    int codigo;
    char* descricao;
    char* curso;
    char* docente;
    int ano;
    int semestre;
    int ocupado;
} UNIDADECURRICULAR;

//definicao da estrutura de exames
typedef struct
{
    int codigo;
    char* unidade_curricular;
    int ano;
    char* curso;
    char* epoca;
    char* data;
    char* hora;
    int duracao;
    char* sala;
    int alunos_inscritos;
    int realizado;
    int ocupado;
} EXAMES;
```

Figura 19 - Definição da estrutura UNIDADECURRICULAR e EXAMES

3.8. Validações

Durante a implementação da aplicação tornou-se necessário usar validações com o intuito de deixar a mesma mais eficiente, pois sem validações poderia, por exemplo, ocorrer a criação de uma sala já existente, eliminação da mesma quando está definida para um exame, marcação de um exame em época especial de um aluno do primeiro ano ou regime normal.

Caso os dados introduzidos não estejam corretos, aparecerá uma mensagem de erro e informar que dados foram inseridos incorretamente e solicita que sejam inseridos novamente.

De seguida mostram-se algumas das validações usadas.

3.8.1. Validações em alunos.c

No caso do ficheiro alunos.c foi necessário usar validações para verificar se o regime e ano de matrícula existem, se o aluno em causa já está criado e se um determinado aluno pode ser editado ou apagado (caso esteja inscrito nalgum exame não poderá ser).

```
//funcao que valida se o regime existe
int valida_regime_escolhido(REGIMES* regimes, int posicaoRegime){
    int i = 0;

    for (i = 0; i < MAX_REGIMES; i++)
    {
        if (regimes[i].ocupado == 1 && i == posicaoRegime) {
            return 1;
        }
    }

    return 0;
}

//funcao que valida o ano de matricula
int valida_ano_matricula(int ano_matricula) {
    if (ano_matricula < 1 || ano_matricula > 3) {
        return 0;
    }
    return 1;
}

//funcao que valida o numero do aluno
int valida_aluno_existe(ALUNOS* alunos, int numero) {
    int i = 0;
    for (i = 0; i < MAX_ALUNOS; i++)
    {
        if (alunos[i].ocupado == 1) {
            if (alunos[i].numero == numero) {
                return 1;
            }
        }
    }

    return 0;
}
```

Figura 20 - Validações da existência do regime, ano de matrícula e aluno

```

//valida exame
int valida_delete_aluno(INSCRICOESEXAMES* inscricoes_exames, ALUNOS* alunos, int numero_aluno, int opcao_aluno) {
    int i = 0;
    for ( i = 0; i < MAX_INSCRICOES; i++)
    {
        if (inscricoes_exames[i].ocupado == 1) {
            if (inscricoes_exames[i].numero_aluno == opcao_aluno) {
                return 0;
            }
        }
    }
    return 1;
}

```

Figura 21 - Validação da existência de inscrição em exame

3.8.2. Validações em UnidadesCurriculares.c

Neste ficheiro foram necessárias usar validações para verificar se o utilizador introduziu corretamente os dados.

Alguns exemplos, verificar se ao criar uma Unidade Curricular para um determinado curso, a mesma não está já criada, se o curso para o qual a Unidade Curricular está a ser criada existe e se o ano e semestre são inseridos corretamente.

```

// valida se o utilizador introduziu um dos cursos da lista
int valida_curso_escolhido(CURSO* cursos, int escolhido) {
    int i = 0;

    for ( i = 0; i < MAX_CURSOS; i++)
    {
        if (cursos[i].ocupado == 1 && i == escolhido) {
            return 1;
        }
    }
    return 0;
}

//funcao que valida se o utilizador introduziu bem os dados do ano e do semestre
int valida_ano_semestre(int ano, int semestre) {
    if(ano < 1 || ano > 3 ) {
        return 0;
    }
    if(semestre < 1 || semestre > 2 ) {
        return 0;
    }
    return 1;
}

//valida se a Unidade Curricular já existe
int valida_UC_existe_vector(UNIDADECURRICULAR* uc, char* descricao, char* curso) {
    int i = 0;
    for ( i = 0; i < MAX_UNIDADES_CURRICULARES; i++)
    {
        if (uc[i].ocupado == 1) {
            if ((strcmp(uc[i].descricao, descricao) == 0) && strcmp(uc[i].curso, curso) == 0) {
                return 1;
            }
        }
    }
    return 0;
}

```

Figura 22 - Validações Unidades Curriculares

```

//valida se o ID da Unidade Curricular foi inserido corretamente
int valida_cod_UC(UNIDADECURRICULAR* uc, int codigo) {
    int i = 0;
    for ( i = 0; i < MAX_UNIDADES_CURRICULARES; i++)
    {
        if ((uc[i].ocupado == 1 ) && (uc[i].codigo == codigo)) {
            return i;
        }
    }
    return -1;
}

//valida se existe algum exame para a Unidade Curricular
int valida_delete_UC(EXAMES* exames_bv, char* curso, char* unidadecurricular) {
    int i = 0;
    for ( i = 0; i < MAX_EXAMES_FILE; i++)
    {
        if (exames_bv[i].ocupado == 1) {
            if ((exames_bv[i].curso == curso) && (exames_bv[i].unidade_curricular == unidadecurricular) ) {
                return 0;
            }
        }
    }
    return 1;
}

```

Figura 23 - Validações para apagar Unidade Curricular

3.8.3. Validações em exames.c

Neste caso usaram-se várias validações. Alguns exemplos são a verificação se um determinado aluno se pode inscrever a um exame em época especial, se ao marcar um exame o mesmo não é em dia de fim de semana ou feriado e se ao apagar um exame o mesmo já foi realizado.

```
//funcao que valida se o exame que introduziu pode ser apagado
int valida_codigo_exame_apagar(EXAMES* exames_bv, int codigo_exame) {
    int i = 0;
    for ( i = 0; i < MAX_EXAMES_FILE; i++)
    {
        if ((exames_bv[i].ocupado == 1) && (exames_bv[i].realizado == 1)) {
            if (exames_bv[i].codigo == codigo_exame) {
                return i;
            }
        }
    }
    return -1;
}
```

Figura 24 - Validação para apagar exame

```
// validacao data
DATA* validacao_data(DATA* datainserida, DATA* datainicio, DATA* datafim, FERIADOS* feriado, EXAMES* exames_bv) {
    int ano, mes, dia = 0;
    int data_valida = 1;
    DATA* hoje = (DATA*)malloc(sizeof(DATA));

    hoje = data_atual(hoje);
    do
    {
        printf("Qual a data pretendida? O formato e dd mm aaaa\n");
        scanf("%d %d %d", &dia, &mes, &ano);
        if (ano < hoje->ano) {
            data_valida = 0;
        }
        else {
            if (mes < 1 || mes > 12) {
                data_valida = 0;
            }
            else {
                if (dia < 1 || dia > 31) {
                    data_valida = 0;
                }
                else {
                    // vamos verificar se a data esta dentro da epoca escolhida
                    datainserida->ano = ano;
                    datainserida->mes = mes;
                    datainserida->dia = dia;
                    int dentro_dadas = valida_data_dentro_epoca(datainserida, datainicio, datafim);
                    if (dentro_dadas == 0) {
                        data_valida = 0;
                        printf("A data inserida esta fora do intervalo de dados da epoca\n");
                    }
                    else {
                        //valida se e feriado
                        int data_de_feriado = e_feriado(datainserida, feriado);
                        if (data_de_feriado == 1) {
                            printf("A data introduzida e um feriado\n");
                            data_valida = 0;
                        }
                        else {
                            // vamos validar se e fim de semana
                            int weekday = (datainserida->dia += datainserida->mes < 3 ? datainserida->ano - 1 : datainserida->ano - 2, 23*datainserida->mes/9 + datainserida->dia + 4 + datainserida->ano/4 - datainserida->ano/100 + datainserida->ano/400)%7;
                            if (weekday == 0 || weekday == 6) {
                                printf("Nao e permitido marcar exames ao fim de semana\n");
                                data_valida = 0;
                            }
                            else {
                                //vamos verificar se existe algum exame do mesmo ano curricular ate 3 dias antes ou depois
                                numero_dias_entre_exames(exames_bv, datainserida);
                            }
                        }
                    }
                }
            }
        } while (data_valida == 0);

    free(hoje);
    return datainserida;
}
```

Figura 25 - Validação da data inserida

3.8.4. Validações em salas.c

No ficheiro salas.c foram necessárias usar validações, entre as quais, verificar se ao criar uma sala a mesma existe, se ao apagar uma sala a mesma foi escolhida dentro das que estão disponíveis ou se não existem exames marcados para essa mesma sala.

```
//verifica se a sala ja existe
int valida_sala_existe_vector(SALAS* salas, char* codigo_sala) {
    int i = 0;
    for ( i = 0; i < MAX_SALAS; i++)
    {
        if (salas[i].ocupado == 1) {
            if (strcmp(salas[i].codigo, codigo_sala) == 0) {
                return 1;
            }
        }
    }
    return 0;
}
```

Figura 26 - Validação que verifica se a sala existe

```
//valida se o numero da sala existe
int valida_cod_sala(SALAS* salas, int opcaoSala) {
    int i = 0;
    for ( i = 0; i < MAX_SALAS; i++)
    {
        if ((salas[i].ocupado == 1 ) && (salas[i].id == opcaoSala)) {
            return i;
        }
    }
    return -1;
}

//valida se a sala não está definida para algum exame
int valida_delete_sala(EXAMES* exames, char* codigo) {
    int i = 0;
    for ( i = 0; i < MAX_EXAMES_FILE; i++)
    {
        if (exames[i].ocupado == 1) {
            if ((exames[i].sala == codigo)) {
                return 0;
            }
        }
    }
    return 1;
}
```

Figura 27 – Validação do número da sala e se pode ser apagada

4. Conclusões

Com este trabalho concluímos que a linguagem C, embora tenha vindo a ser substituída por outras mais recentes, é bastante poderosa. É utilizada nas mais diversas plataformas, no desenvolvimento de sistemas operacionais e no desenvolvimento de aplicações, como o caso da que vimos neste relatório.

Durante a implementação da aplicação aplicámos os conhecimentos adquiridos na Unidade Curricular, nomeadamente, uso de estruturas, uso de funções, alocação de memória e uso de ficheiros.

Com a realização deste projeto foi possível também verificar as dificuldades que surgem na construção e implementação de um programa, pelo que, após o tempo investido no projeto, torna-se gratificante ver o mesmo em funcionamento.