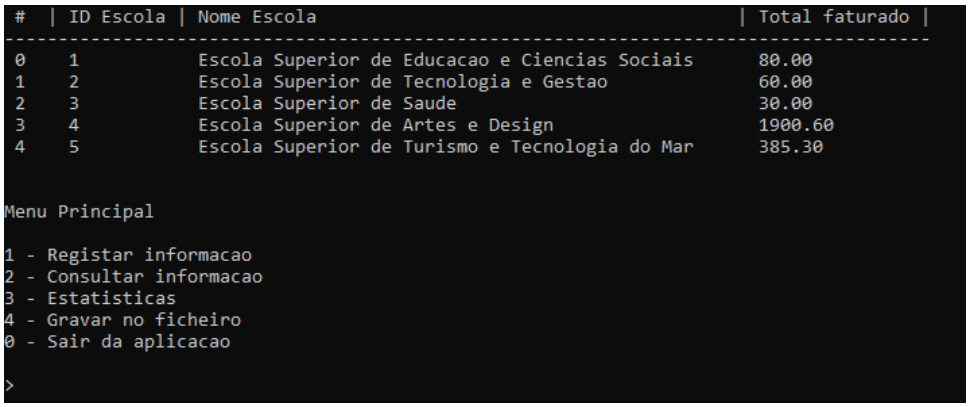
POLITÉCNICO DE LEIRIA

Curso de TeSP em
Programação de Sistemas de Informação
Fundamentos de Programação

Ano Letivo 2021/2022 **1º Ano, 1º Semestre**

Projeto de Fundamentos de Programação



Grupo: PL2-7	**Docente:** José Ramos
Nº 2211868	Marco Padeiro
Nº 2211866	Tomás Moura

Cofinanciado por:

CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

Cofinanciado por:

CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

ÍNDICE

Cofinanciado por:

CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

ÍNDICE DE FIGURAS

Cofinanciado por:
CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

1. INTRODUÇÃO

Este projeto foi realizado no âmbito da Unidade Curricular de Fundamentos de Programação e tem como objetivo o desenvolvimento de um protótipo de uma aplicação que permita efetuar a gestão das operações de carregamento e pagamento (transações) do sistema SIGA nos bares e cantinas do Politécnico de Leiria.

No desenvolvimento deste projeto foi utilizada a linguagem de programação C, que auxiliou a manipulação de ficheiros binários, estruturas de dados complexas e desenvolvimento do aspeto gráfico em geral.

A aplicação desenvolvida situa-se num ambiente de consola, ou seja, toda a entrada de informação feita pelo utilizador tem de ser através do teclado e a saída de informação só suporta caracteres.

Este documento tem como objetivo auxiliar a compreensão das decisões e escolhas tomadas pelo grupo responsável pela criação do projeto.

Cofinanciado por:
CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

2. ESPECIFICAÇÃO DO SISTEMA

2.1. Estruturas de dados utilizadas

Na criação do projeto foram utilizadas as seguintes estruturas de dados…

2.1.1. "Struct" de Escolas

A criação deste "struct" serve para armazenar todas as escolas, onde o grupo utilizou o typedef para mais tarde ser mais fácil o chamamento deste tipo de variável pelo nome "t_escola". Constitui uma variável de tipo inteiro, chamada "id_escola" que serve para armazenar o identificador de escola, de seguida existem quatro vetores do tipo char que servem para armazenar o nome da escola, a abreviatura, a que campus pertence e a sua localização. Cada um destes vetores tem um tamanho diferente, tendo o nome da escola 60 elementos, a abreviatura 10, o campus 20 e a localização 30.

```
34  typedef struct {
35      int id_escola;
36      char nome_escola[TAMANHO_NOME_ESCOLA], abreviatura[TAMANHO_ABREVIATURA], campus[TAMANHO_CAMPUS], localizacao[TAMANHO_LOCALIZACAO];
37  } t_escola;
```

Fig.1- Struct de escolas.

2.1.2. "Struct" de Utilizadores

A criação deste "struct" serve para armazenar todos os utilizadores, onde o grupo utilizou o typedef para mais tarde ser mais fácil o chamamento deste tipo de variável pelo nome "t_utilizador". Constitui três variáveis de tipo inteiro, chamadas "id_utilizador", "id_escola", "NIF" que servem para armazenar o identificador do utilizador, o identificador de escola e o NIF respetivamente, de seguida existe uma variável do tipo float chamada "saldo" que armazena o saldo, por fim existem três vetores do tipo char que servem para armazenar o nome do utilizador, o tipo de utilizador, e o seu email. Cada um destes vetores tem um tamanho diferente, tendo o nome do utilizador 30 elementos, o tipo de utilizador 20 e o email 30.

```
typedef struct {
    int id_utilizador, id_escola, NIF;
    float saldo;
    char nome_utilizador[TAMANHO_NOME_UTILIZADOR], tipo_utilizador[TAMANHO_TIPO_UTILIZADOR], email[TAMANHO_EMAIL];
} t_utilizador;
```

Fig.2- Struct de utilizadores.

2.1.3. "Struct" de Transações

A criação deste "struct" serve para armazenar todas as trasações, onde o grupo utilizou o typedef para mais tarde ser mais fácil o chamamento deste tipo de variável pelo nome "t_transacao". Constitui duas variáveis de tipo inteiro, chamadas "id_transacao", "id_utilizador" que servem para armazenar o identificador de transação e o identificador do utilizador respetivamente, existe uma variável do tipo float chamada "valor" que armazena o valor, existe também um vetor do tipo char que serve para armazenar o tipo de transação com o tamanho de 30 elementos. Neste struct fomos também buscar as variáveis da hora e data a outros structs.

```
typedef struct {
    int id_transacao, id_utilizador;
    char tipo_transacao[TAMANHO_TIPO_TRANSACAO];
    float valor;
    t_data data;
    t_hora hora;
} t_transacao;
```

Fig.3- Struct de transações.

2.1.4. "Struct" da data

A criação deste "struct" serve para armazenar as datas, onde o grupo utilizou o typedef para mais tarde ser mais fácil o chamamento deste tipo de variável pelo nome "t_data". Constitui três variáveis de tipo inteiro, chamadas "dia", "mes", "ano" que servem para armazenar o dia, mês e ano respetivamente.

```
typedef struct {
    int dia, mes, ano;
} t_data;
```

Fig.4- Struct da data.

2.1.5. "Struct" da hora

A criação deste "struct" serve para armazenar as horas, onde o grupo utilizou o typedef para mais tarde ser mais fácil o chamamento deste tipo de variável pelo nome "t_horas". Constitui três variáveis de tipo inteiro, chamadas "hora", "minuto", "segundo" que servem para armazenar a hora, o minuto e os segundos respetivamente.

```
typedef struct {
    int hora, minuto, segundo;
} t_hora;
```

Fig.5- Struct da hora.

2.1.6. "Struct" principal

A criação deste "struct" serve para armazenar as escolas, utilizadores e transações, onde o grupo utilizou o typedef para mais tarde ser mais fácil o chamamento deste tipo de variável pelo nome "t_principal". Constitui três vetores de structs do tipo "t_escola", "t_utilizador" e "t_transacao", chamadas "v_escola", "v_utilizador" e "v_transacao" respetivamente que armazenam todas as escolas, utilizadores e transações com o número máximo de 5 elementos para as escolas, 200 para os utilizadores e 5000 para as transações.

```
typedef struct {
    t_escola v_escola[MAX_ESCOLA];
    t_utilizador v_utilizador[MAX_UTILIZADOR];
    t_transacao v_transacao[MAX_TRANSACAO];
} t_principal;
```

Fig.6- Struct principal.

2.2. Constantes e variáveis principais

Na criação do projeto foram criadas as seguintes constantes...

```
#define MAX_ESCOLA 5
#define MAX_UTILIZADOR 200
#define MAX_TRANSACAO 5000
```

Fig.7- Constantes de limites na base de dados.

Estas constantes representam o limite de elementos na base de dados.

```
#define SUCESSO 0
#define INSUCESSO 1

#define ESCOLA 1
#define UTILIZADOR 2
#define TRANSACAO 3

#define BASE_DADOS_CHEIA -1

#define IGUAL 0
#define MAIOR 1
#define MENOR 2
```

Fig.8- Constantes para facilitar leitura.

Estas constantes foram criadas para facilitar a leitura do código.

```
#define TAMANHO_NOME_ESCOLA 60
#define TAMANHO_TIPO_TRANSACAO 30
#define TAMANHO_ABREVIATURA 10
#define TAMANHO_CAMPUS 20
#define TAMANHO_LOCALIZACAO 30
#define TAMANHO_NOME_UTILIZADOR 30
#define TAMANHO_TIPO_UTILIZADOR 20
#define TAMANHO_EMAIL 30
```

Fig.9- Constantes máximo de caracteres por string.

E por último existem as constantes com o número máximo de caracteres por string.

Variáveis:

```
void main() { // 4 linhas
    t_principal s_principal;                    // cria uma variavel responsavel por armazenar a base de dados na memoria
```

Fig.10-Variável principal.

A única variável que consideramos sendo a principal é a s_principal do tipo t_principal que armazena todas as outras estruturas de dados na memoria e foi declarada na função main, mas tendo o seu valor sempre alterado através de outras funções.

2.3. Funcionalidades implementadas e respetivas funções

Este projeto consiste num protótipo de uma aplicação que permita efetuar a gestão das operações de carregamento e pagamento (transações) do sistema SIGA nos bares e cantinas do Politécnico de Leiria e para tal foi necessário criar algumas funcionalidades. As funcionalidades que o nosso grupo implementou foram as seguintes:

1. Registar dados:

Para registar informação o grupo criou um menu para escolher a base de dados que o utilizador pretende inserir dados. Mas antes disso no menu principal, aparecem várias funções para o utilizador escolher o que pretender fazer e ao selecionar que quer registar informação vai ser levado para um menu onde vai ter de escolher qual é a base de dados que quer manipular.

```c
int selecinar_base_dados() { // 7 linhas
    int opcao;
    // obtem uma base de dados          Escolas || Utilizadores || Transações
    do {
        system("cls");
        opcao = obter_input(0, 3, "Selecione a base de dados que deseja manipular:\n\n", "1 - Escolas\n", "2 - Utilizadores\n", "3 - Transacoes\n", "0 - Voltar atras", "\0");

    } while (opcao < 0 && opcao > 3);
    return opcao;
}
```

Fig.11-Menu para escolher a base de dados.

Depois de escolher a base de dados pretendida, irá ser levado para o menu para registar a informação na base de dados que foi escolhida anteriormente.

Para isso acontecer corretamente o grupo criou uma função que verifica se a base de dados existe e se não está cheia, depois dependendo da base de dados escolhida esta nova função redireciona para a função "registar_...", caso a base de dados esteja cheia, irá para o menu onde diz que a base de dados está cheia.

```c
int registar_informacao(t_principal* s_principal, int index, int baseDados) { // 9 linhas
    int sucesso = INSUCESSO;
    if(index != BASE_DADOS_CHEIA){ // caso a base de dados nao esteja cheia
        switch (baseDados) {
            case ESCOLA: sucesso = registar_escola(s_principal, index); break;            // insere escolas
            case UTILIZADOR: sucesso = registar_utilizador(s_principal, index); break;    // insere utilizadores
            case TRANSACAO: sucesso = registar_transacao(s_principal, index); break;      // insere transacoes
        }
    } else { // caso a base de dados esteja cheia
        limite_base_dados(s_principal, baseDados);  // chama o menu que diz que a base de dados está cheia
    }
    return sucesso;
}
```

Fig.12-Função para verificar se a base de dados está cheia e se existe.

```c
int registar_escola(t_principal* s_principal, int index) { // 19 linhas
    // obtem o input do utilizador, valida-o e caso esteja tudo valido, insere na base de dados na memoria
    t_escola v_aux_escola[MAX_ESCOLA];
    printf("Identificador Escola: ");
    obter_identificador(s_principal, &v_aux_escola[index].id_escola, ESCOLA, 0);
    fflush(stdin);
    printf("Nome Escola: ");
    scanf("%[^\n]s", v_aux_escola[index].nome_escola);
    fflush(stdin);
    printf("Abreviatura: ");
    scanf("%[^\n]s", v_aux_escola[index].abreviatura);
    printf("Campus: ");
    obter_string(v_aux_escola[index].campus, "Campus 1", "\0", "\0", 2, 0);
    fflush(stdin);
    printf("Localizacao: ");
    scanf("%[^\n]s", v_aux_escola[index].localizacao);
    s_principal->v_escola[index].id_escola = v_aux_escola[index].id_escola;
    strcpy(s_principal->v_escola[index].nome_escola, v_aux_escola[index].nome_escola);
    strcpy(s_principal->v_escola[index].abreviatura, v_aux_escola[index].abreviatura);
    strcpy(s_principal->v_escola[index].campus, v_aux_escola[index].campus);
    strcpy(s_principal->v_escola[index].localizacao, v_aux_escola[index].localizacao);
    return SUCESSO;
}
```

Fig.13-Função para registar escolas.

```c
int registar_utilizador(t_principal* s_principal, int index) { // 25 linhas
    // obtem o input do utilizador, valida-o e caso esteja tudo valido, insere na base de dados na memoria
    t_utilizador v_aux_utilizador[MAX_UTILIZADOR];
    fflush(stdin);
    printf("Identificador Utilizador: ");
    obter_identificador(s_principal, &v_aux_utilizador[index].id_utilizador, UTILIZADOR, 0);
    printf("Identificador Escola: ");
    obter_identificador(s_principal, &v_aux_utilizador[index].id_escola, ESCOLA, 1);
    fflush(stdin);
    printf("Nome Utilizador: ");
    scanf("%[^\n]s", v_aux_utilizador[index].nome_utilizador);
    printf("NIF: ");
    obter_int(&v_aux_utilizador[index].NIF, 10000000, 999999999);
    printf("Tipo Utilizador: ");
    obter_string(v_aux_utilizador[index].tipo_utilizador, "Docente", "Estudante", "Funcionario", 3, 1);
    fflush(stdin);
    printf("Email: ");
    obter_email(v_aux_utilizador[index].email);
    printf("Saldo: ");
    scanf("%f", &v_aux_utilizador[index].saldo);
    s_principal->v_utilizador[index].id_utilizador = v_aux_utilizador[index].id_utilizador;
    s_principal->v_utilizador[index].id_escola = v_aux_utilizador[index].id_escola;
    strcpy(s_principal->v_utilizador[index].nome_utilizador, v_aux_utilizador[index].nome_utilizador);
    s_principal->v_utilizador[index].NIF = v_aux_utilizador[index].NIF;
    strcpy(s_principal->v_utilizador[index].tipo_utilizador, v_aux_utilizador[index].tipo_utilizador);
    strcpy(s_principal->v_utilizador[index].email, v_aux_utilizador[index].email);
    s_principal->v_utilizador[index].saldo = v_aux_utilizador[index].saldo; return SUCESSO;
}
```

Fig.14-Função para registar utilizadores.

FALTA REGISTAR TRANSACAO

2. Consultar dados:

Para consultar os dados o grupo criou também uma função adaptável à base de dados escolhida.

O sistema funciona igual ao de registar informação, depois do utilizador escolher no menu principal que quer consultar dados, irá aparecer o menu a perguntar qual é a base de dados que ele pretende consultar e depois há uma função que verifica qual foi a base de dados escolhida e mostra os respetivos dados.

```c
void menu_consultar(t_principal* s_principal, int baseDados) { // 21 linhas
    system("cls");
    int opcao = 0, max_elementos = 0;

    // para definir o titulo da base de dados que está a inserir
    char titulo[13];
    switch(baseDados){
        case ESCOLA: strcpy(titulo, "Escolas"); max_elementos = MAX_ESCOLA; break;
        case UTILIZADOR: strcpy(titulo, "Utilizadores"); max_elementos = MAX_UTILIZADOR; break;
        case TRANSACAO: strcpy(titulo, "Transacoes"); max_elementos = MAX_TRANSACAO; break;
    }
    printf("Menu Consultar - %s\n\n", titulo);
    if(baseDados == ESCOLA || baseDados == UTILIZADOR || baseDados == TRANSACAO){ // caso a base de dados inserida no parametro seja valida
        cabecalho_apresentar_dados(baseDados);
        consultar_informacao(s_principal, baseDados); // apresenta a todos os elementos da respetiva base de dados

        // ofecere varias opções ao utilizador e espera pelo seu input
        opcao = obter_input(0, 4, "\n1 - Apagar elemento", "\n2 - Alterar elemento", "\n3 - Mudar base de dados", "\n4 - Limpar base de dados", "\n0 - Voltar Atras", "\0");
        switch (opcao){
            case 1: apagar_elemento(s_principal, receber_index(), baseDados); break;
            case 2: alterar_elemento(s_principal, receber_index(), baseDados); break;
            case 3: menu_consultar(s_principal, selecinar_base_dados()); break;
            case 4: limpa_array(s_principal, baseDados);  menu_consultar(s_principal, baseDados);break;
            case 0: menu_principal(s_principal); break;
        }
    } else { menu_principal(s_principal); }
}
```

Fig.15-Função para consultar dados.

```c
void consultar_informacao(t_principal* s_principal, int baseDados) { // 24 linhas
    int index = 0;
    // verifica qual a base de dados inserida por parametro
    switch (baseDados) {
        // cria um ciclo que apresenta todos os elementos que NAO estejam vazios
        case ESCOLA:
            for(index = 0; index < MAX_ESCOLA; index++){
                if(s_principal->v_escola[index].id_escola != 0){
                    apresentar_dados(s_principal, index, ESCOLA);
                }
            }
            break;
        case UTILIZADOR:
            for(index = 0; index < MAX_UTILIZADOR; index++){
                if(s_principal->v_utilizador[index].id_utilizador != 0){
                    apresentar_dados(s_principal, index, UTILIZADOR);
                }
            }
            break;
        case TRANSACAO:
            for(index = 0; index < MAX_TRANSACAO; index++){
                if(s_principal->v_transacao[index].id_transacao != 0){
                    apresentar_dados(s_principal, index, TRANSACAO);
                }
            }
            break;
    }
}
```

Fig.15-Função para consultar toda a informação que não esteja vazia.

```c
void apresentar_dados(t_principal* s_principal, int index, int baseDados){ // 12 linhas
    char espacos[100];
    switch(baseDados){
        case ESCOLA: ;
            apresentar_escolas(s_principal, index, baseDados);
            break;
        case UTILIZADOR: ;
            apresentar_utilizadores(s_principal, index, baseDados);
            break;
        case TRANSACAO: ;
            apresentar_transacoes(s_principal, index,  baseDados);
            break;
    }
}
```

Fig.16-Função que recebe os dados não vazios e dependendo da base de dados inserida apresenta os dados.

```c
void apresentar_escolas(t_principal* s_principal, int index, int baseDados){ // 13 linhas
    char espacos[100];
    t_escola v_aux_escola = s_principal->v_escola[index];
    calcula_numero_de_espacos(espacos, 5, conta_caracteres_numero(index));
    printf(" %d%s", index, espacos);
    calcula_numero_de_espacos(espacos, 12, conta_caracteres_numero(v_aux_escola.id_escola));
    printf("%d%s", v_aux_escola.id_escola, espacos);
    calcula_numero_de_espacos(espacos, 52, strlen(v_aux_escola.nome_escola));
    printf("%s%s", v_aux_escola.nome_escola, espacos);
    calcula_numero_de_espacos(espacos, 15, strlen(v_aux_escola.abreviatura));
    printf("%s%s", v_aux_escola.abreviatura, espacos);
    calcula_numero_de_espacos(espacos, 13, strlen(v_aux_escola.campus));
    printf("%s%s", v_aux_escola.campus, espacos);
    printf("%s\n", v_aux_escola.localizacao);
}
```

Fig.17-Função para apresentar os dados das escolas.

```c
void apresentar_utilizadores(t_principal* s_principal, int index, int baseDados){ // 17 linhas
    char espacos[100];
    t_utilizador v_aux_utilizador = s_principal->v_utilizador[index];
    calcula_numero_de_espacos(espacos, 4, conta_caracteres_numero(index));
    printf(" %d%s", index, espacos);
    calcula_numero_de_espacos(espacos, 16, conta_caracteres_numero(v_aux_utilizador.id_utilizador));
    printf("%d%s", v_aux_utilizador.id_utilizador, espacos);
    calcula_numero_de_espacos(espacos, 12, conta_caracteres_numero(v_aux_utilizador.id_escola));
    printf("%d%s", v_aux_utilizador.id_escola, espacos);
    calcula_numero_de_espacos(espacos, 22, strlen(v_aux_utilizador.nome_utilizador));
    printf("%s%s", v_aux_utilizador.nome_utilizador, espacos);
    calcula_numero_de_espacos(espacos, 12, conta_caracteres_numero(v_aux_utilizador.NIF));
    printf("%d%s", v_aux_utilizador.NIF, espacos);
    calcula_numero_de_espacos(espacos, 18, strlen(v_aux_utilizador.tipo_utilizador));
    printf("%s%s", v_aux_utilizador.tipo_utilizador, espacos);
    calcula_numero_de_espacos(espacos, 24, strlen(v_aux_utilizador.email));
    printf("%s%s", v_aux_utilizador.email, espacos);
    printf("%.2f\n", v_aux_utilizador.saldo);
}
```

Fig.18-Função para apresentar os dados dos utilizadores.

```c
void apresentar_transacoes(t_principal* s_principal, int index, int baseDados){ // 17 linhas
    char espacos[100];
    t_transacao v_aux_transacao = s_principal->v_transacao[index];
    calcula_numero_de_espacos(espacos, 6, conta_caracteres_numero(index));
    printf(" %d%s", index, espacos);
    calcula_numero_de_espacos(espacos, 17, conta_caracteres_numero(v_aux_transacao.id_transacao));
    printf("%d%s", v_aux_transacao.id_transacao, espacos);
    calcula_numero_de_espacos(espacos, 19, conta_caracteres_numero(v_aux_transacao.id_utilizador));
    printf("%d%s", v_aux_transacao.id_utilizador, espacos);
    calcula_numero_de_espacos(espacos, 19, strlen(v_aux_transacao.tipo_transacao));
    printf("%s%s", v_aux_transacao.tipo_transacao, espacos);
    calcula_numero_de_espacos(espacos, 21, conta_caracteres_numero(v_aux_transacao.valor) + 3);
    printf("%.2f%s", v_aux_transacao.valor, espacos);
    calcula_numero_de_espacos(espacos, 16, 10);
    printf("%02d/%02d/%04d%s", v_aux_transacao.data.dia, v_aux_transacao.data.mes, v_aux_transacao.data.ano, espacos);
    printf("%02d:%02d:%02d\n", v_aux_transacao.hora.hora, v_aux_transacao.hora.minuto, v_aux_transacao.hora.segundo);
}
```

Fig.19-Função para apresentar os dados das transações.

3. Estatísticas:

Quando o utilizador escolhe ir para as estatísticas vai-lhe ser apresentado um menu com dados da escola, o total faturado, o número de pagamentos efetuados e a percentagem de pagamentos por escola.

```c
void menu_estatisticas(t_principal* s_principal, t_data dataInicial, t_data dataFinal, char tipoUtilizador[]){ // 23 linhas
    system("cls");
    float total_faturado = 0;
    int index = 0, opcao = 0;
    char espacos[100];
    // apresenta o menu
    printf("Visualizar Estatisticas\n\n");
    if(dataInicial.dia != 0){ printf("Data Inicial: %d/%d/%d\n", dataInicial.dia, dataInicial.mes, dataInicial.ano); } // apresenta a data inicial do filtro (caso seja 0 nao a
    else { printf("Data Inicial: \n"); }
    if(dataFinal.dia != 0){ printf("Data Final: %d/%d/%d\n", dataFinal.dia, dataFinal.mes, dataFinal.ano); } // apresenta a data inicial do filtro (caso seja 0 nao apresenta)
    else { printf("Data Final: \n"); }

    // apresenta o menu das estatisticas
    printf("Tipo de utilizador: %s\n\n", tipoUtilizador);
    printf(" #  | ID Escola | Nome Escola                                    | Total faturado |  N%c pagamentos  |  %% pagamentos |\n", 248);
    printf("-------------------------------------------------------------------------------------------------------------\n");
    // apresenta todas as escolas
    for(index = 0; index < MAX_ESCOLA; index++){
        if(s_principal->v_escola[index].id_escola != 0){
            escrever_elemento_tabela_estatisticas(s_principal, dataInicial, dataFinal, tipoUtilizador, index);
        }
    }
    // obtem input do utilizador
    opcao = obter_input(0, 1, "\n\n\n1 - Filtros","\n0 - Voltar atras", "\0", "\0", "\0", "\0");
    switch(opcao){
        case 1: escrever_filtos(s_principal); break;
        case 0: menu_principal(s_principal); break;
        default: menu_principal(s_principal);
    }
}
```

Fig.20-Função para apresentar o menu das estatisticas.

4. Total faturado por escola:

```c
float calcula_total_faturado_por_escola(t_principal * s_principal, int id_escola, t_data dataInicial, t_data dataFinal, char tipoUtilizador[]){ // 22 linhas
    int index = 0;
    float total_faturado = 0;
    for(index = 0; index < MAX_TRANSACAO; index++) { // vai por todas as transacoes
        if(s_principal->v_transacao[index].id_transacao != 0) { // se nao estiver vazio
            if(strcmp(s_principal->v_transacao[index].tipo_transacao, "Pagamento") == 0){
                if(verifica_se_utilizador_pertence_escola(s_principal, s_principal->v_transacao[index].id_utilizador, id_escola) == SUCESSO){
                    if(dataInicial.dia != 0){
                        if(verifica_se_data_esta_dentro(s_principal->v_transacao[index].data, dataInicial) == MAIOR){
                            if(verifica_se_data_esta_dentro(s_principal->v_transacao[index].data, dataFinal) == MENOR){
                                for(int i = 0; i < MAX_UTILIZADOR; i++){
                                    if(s_principal->v_utilizador[i].id_utilizador == s_principal->v_transacao[index].id_utilizador){
                                        if(strcmp(s_principal->v_utilizador[index].tipo_utilizador, tipoUtilizador) == 0){ total_faturado += s_principal->v_transacao[index]
                                        }
                                    }
                                }
                            }
                        }
                    } else { total_faturado += s_principal->v_transacao[index].valor; }
                }
            }
        }
    }
    return total_faturado;
}
```

Fig.21-Função para calcular o total faturado por escola.

5. Percentagem de transações por escola:

```c
int calcula_percentagem_transacoes(t_principal * s_principal, int id_escola, t_data dataInicial, t_data dataFinal, char tipoUtilizador[]){  // 23 linhas
    int percentagem_transacoes = 0, index = 0, total_transacoes = 0, transacoes_escola = 0;
    for(index = 0; index < MAX_TRANSACAO; index++) { // vai por todas as transacoes
        if(s_principal->v_transacao[index].id_transacao != 0) { // se nao estiver vazio
            if(strcmp(s_principal->v_transacao[index].tipo_transacao, "Pagamento") == 0){
                total_transacoes++;
                if(verifica_se_utilizador_pertence_escola(s_principal, s_principal->v_transacao[index].id_utilizador, id_escola) == SUCESSO){
                    if(dataInicial.dia != 0){
                        if(verifica_se_data_esta_dentro(s_principal->v_transacao[index].data, dataInicial) == MAIOR){
                            if(verifica_se_data_esta_dentro(s_principal->v_transacao[index].data, dataFinal) == MENOR){
                                for(int i = 0; i < MAX_UTILIZADOR; i++){
                                    if(s_principal->v_utilizador[i].id_utilizador == s_principal->v_transacao[index].id_utilizador){
                                        if(strcmp(s_principal->v_utilizador[index].tipo_utilizador, tipoUtilizador) == 0){ transacoes_escola++; }
                                    }
                                }
                            }
                        }
                    } else { transacoes_escola++; }
                }
            }
        }
    }
    percentagem_transacoes = (transacoes_escola * 100) / total_transacoes;
    return percentagem_transacoes;
}
```

Fig.22-Função para calcular a percentagem de transações por escola.

6. Total de transações entre duas datas:

```c
int verifica_se_data_esta_dentro(t_data dataInserida,t_data dataComparar){ // 17 linhas
    int verificador = IGUAL;
    if(dataInserida.ano < dataComparar.ano){
        verificador = MENOR;
    } else if(dataInserida.ano > dataComparar.ano){
        verificador = MAIOR;
        } else if(dataInserida.mes < dataComparar.mes){
            verificador = MENOR;
            } else if (dataInserida.mes > dataComparar.mes) {
                verificador = MAIOR;
                } else if(dataInserida.dia < dataComparar.dia){
                    verificador = MENOR;
                    } else if(dataInserida.dia > dataComparar.dia){
                        verificador = MAIOR;
                        } else {
                            verificador = IGUAL;
                        }


    return verificador;
}
```

Fig.23-Função para calcular o total de transações entre duas datas.

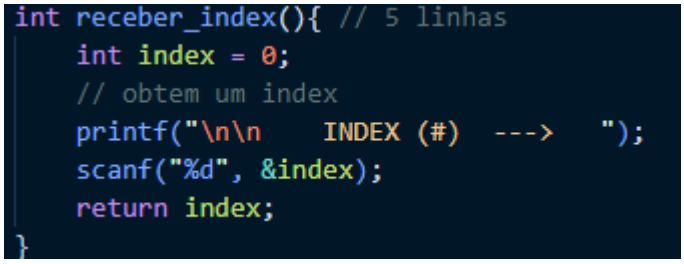
2.4. Funcionalidades por implementar

O grupo conseguiu implementar todas a funcionalidades pedidas pelo docente.

2.5. Validação de dados de entrada

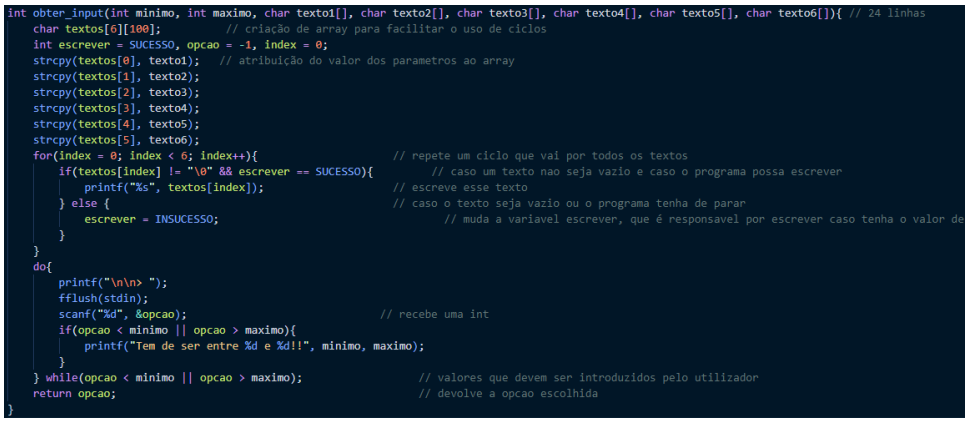
```c
int selecinar_base_dados() { // 7 linhas
    int opcao;
    // obtem uma base de dados          Escolas || Utilizadores || Transações
    do {
        system("cls");
        opcao = obter_input(0, 3, "Selecione a base de dados que deseja manipular:\n\n", "1 - Escolas\n", "2 - Utilizadores\n", "3 - Transacoes\n", "0 - Voltar atras", "\0");

    } while (opcao < 0 && opcao > 3);
    return opcao;
}
```

Fig.24-Função para pedir o input para selecionar uma base de dados.



```c
int receber_index(){ // 5 linhas
    int index = 0;
    // obtem um index
    printf("\n\n    INDEX (#)  --->   ");
    scanf("%d", &index);
    return index;
}
```

Fig.25-Função para pedir o input do index.



```c
int obter_input(int minimo, int maximo, char texto1[], char texto2[], char texto3[], char texto4[], char texto5[], char texto6[]){ // 24 linhas
    char textos[6][100];          // criação de array para facilitar o uso de ciclos
    int escrever = SUCESSO, opcao = -1, index = 0;
    strcpy(textos[0], texto1);   // atribuição do valor dos parametros ao array
    strcpy(textos[1], texto2);
    strcpy(textos[2], texto3);
    strcpy(textos[3], texto4);
    strcpy(textos[4], texto5);
    strcpy(textos[5], texto6);
    for(index = 0; index < 6; index++){                  // repete um ciclo que vai por todos os textos
        if(textos[index] != "\0" && escrever == SUCESSO){       // caso um texto nao seja vazio e caso o programa possa escrever
            printf("%s", textos[index]);                 // escreve esse texto
        } else {                                         // caso o texto seja vazio ou o programa tenha de parar
            escrever = INSUCESSO;                               // muda a variavel escrever, que é responsavel por escrever caso tenha o valor de
        }
    }
    do{
        printf("\n\n> ");
        fflush(stdin);
        scanf("%d", &opcao);                            // recebe uma int
        if(opcao < minimo || opcao > maximo){
            printf("Tem de ser entre %d e %d!!", minimo, maximo);
        }
    } while(opcao < minimo || opcao > maximo);               // valores que devem ser introduzidos pelo utilizador
    return opcao;                                        // devolve a opcao escolhida
}
```

Fig.26-Função para pedir o input para mudar de menus.

Fig.27-Função para obter o input de uma data.

Fig.28-Função para obter o input de uma hora.

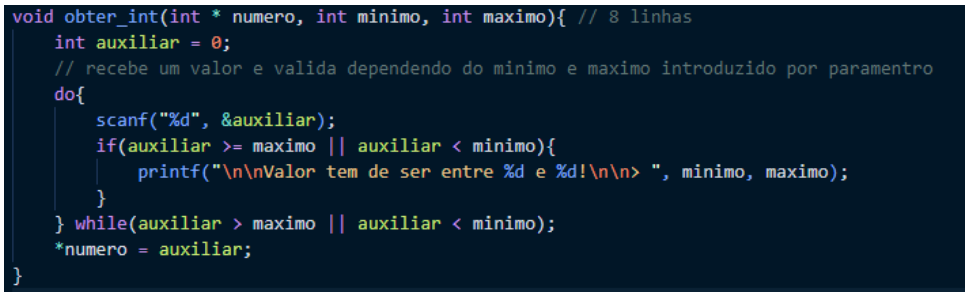


Fig.29-Função para obter um número inteiro.

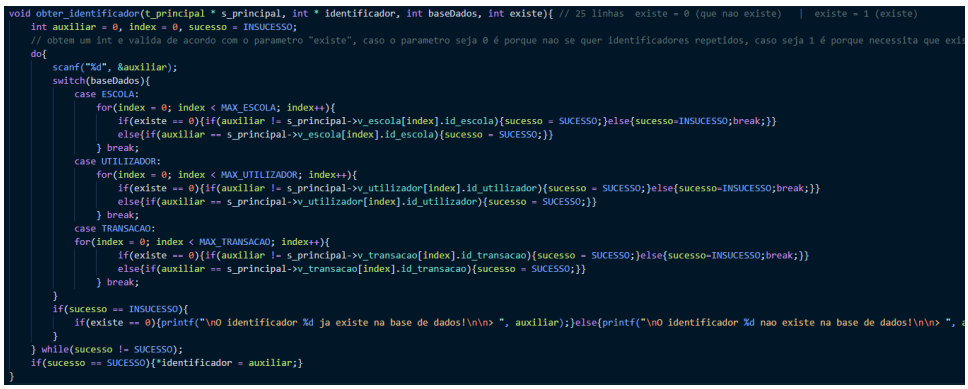


Fig.30-Função para obter um identificador e verificar se já existe na base de dados ou não.

Fig.31-Função para obter um uma string e comparar com outras até ter o pretendido.

```c
void obter_email(char email[]){ // 15 linhas
    char auxiliar[100];
    int index = 0, sucesso = INSUCESSO;
    // obtem uma string e valida a ver se tem um @
    do{
        fflush(stdin);
        scanf("%s", auxiliar);
        for(index = 0; index < 100; index++){
            if(auxiliar[index] == '@'){
                sucesso = SUCESSO;
            }
        }
        if(sucesso != SUCESSO){
            printf("\nO email precisa de ser valido!\n\n> ");
        }
    } while(sucesso != SUCESSO);
    strcpy(email, auxiliar);
}
```

Fig.32-Função para obter um email.

```c
void escrever_filtos(t_principal *s_principal){ // 9 linhas
    // obtem os filtros para o menu de estatisticas
    t_data dataInicial, dataFinal;
    char tipoUtilizador[20];
    printf("Data Inicial (FORMATO dd/mm/yyyy): ");
    obter_data(&dataInicial.dia, &dataInicial.mes, &dataInicial.ano);
    printf("Data Final (FORMATO dd/mm/yyyy): ");
    obter_data(&dataFinal.dia, &dataFinal.mes, &dataFinal.ano);
    printf("Tipo de Utilizador: ");
    scanf("%s", tipoUtilizador);
    menu_estatisticas(s_principal, dataInicial, dataFinal, tipoUtilizador);
}
```

Fig.33-Função para obter os filtros das estatísticas.

2.6. Realização de testes na aplicação

No decorrer deste projeto o grupo teve de realizar vários testes para certificar que as funções estavam todas a funcionar corretamente. À medida que se ia fazendo uma função testava-se logo para ver se funcionava e caso não funcionasse tentava-se corrigir de imediato o problema, podendo assim ter um

controlo de desenvolvimento, isto é, o grupo só avançava para a próxima funcionalidade quando tivesse terminado a anterior, isto não só ajudava no ponto de perceber os problemas e adaptar o código para resolve-los mas também no facto de não perder orientação no projeto porque se o grupo avançasse no desenvolvimento sempre que uma função não ficava a funcionar a cem por cento depois iria ter problemas com o desenvolvimento de outras funções que dependessem da anterior.

3. CONCLUSÕES

Com este projeto, o grupo consegue concluir que atingiu todos os objetivos definidos no começo do projeto e em geral ficou satisfeito com o resultado obtido, apesar de todas as dificuldades que teve de ultrapassar durante a fase de desenvolvimento.

O grupo pode também concluir que este projeto conseguiu ajudar todos os elementos individualmente a compreender melhor a lógica por detrás da programação, especificamente, a utilização de estruturas de dados, o uso de funções, a utilização de variáveis e constantes, validação de dados de entrada e a manipulação de ficheiros, principalmente binários.

Cofinanciado por:

CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu