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: Parque de Estacionamento

Realizado por: Guilherme Félix – 25172

João Cruz – 25178

Rodrigo Pereira – 27450

Viseu, 2024

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

Parque de Estacionamento

Ano Letivo 2023/24

Viseu, 2024

ÍNDICE

Introdução

Este projeto em C aborda a gestão de um parque de estacionamento de um supermercado, composto por três andares, cada um com 20 linhas e 25 colunas, totalizando 1500 lugares. O programa registra entradas e saídas de veículos por meio de matrículas, calculando o custo de estacionamento na saída. Além disso, o sistema gera um mapa que indica os lugares ocupados e a quantidade total de lugares disponíveis. A implementação eficiente em linguagem C visa garantir um funcionamento direto e ágil, abordando de forma prática as necessidades de gestão de estacionamento.

O trabalho explora detalhes da lógica do sistema, destacando a alocação de lugares, o registo de movimentação e o cálculo de custos, com reflexões sobre possíveis melhorias para aprimorar a eficiência global.

No capítulo 2, haverá uma explicação geral na organização do código, através da navegação de menus.

No seguinte, vamos explicar o que cada opção no menu principal faz, seguido de exemplos.

1. Estruturas e acesso a ficheiros

Para este trabalho recorremos a diversas estruturas para separar variáveis relevantes.
Através de 3 funções, abrimos ficheiros distintos, retiramos dados de cada coluna e associamos aos mesmos a uma variável na devida estrutura

1.1. Estruturas

```c
struct Data {
    int ano;
    int mes;
    int dia;
};

struct Cliente {
    int id_cliente;
    char matricula[MAX_MATRICULA_LENGTH];
    char nome[50];
    char cod_postal[9];
    int NIF;
    struct Data data_inscricao;
    struct Data fim_inscricao;
};
```

```c
struct Entradas {
    char matriculas[MAX_MATRICULA_LENGTH];
    struct Data data_entrada;
    int hora;
    int min;

};

struct EntradaseSaidas {
    char matriculas[MAX_MATRICULA_LENGTH];
    struct Data data_entrada;
    int horaent;
    int minent;
    struct Data data_saida;
    int horasaida;
    int minsaida;
};

struct LugarEstacionamento {
    int piso;
    char linha;
    int coluna;
};
```

Na estrutura “Data” é registado ano, mês e dia, sendo esta usada na data de subscrição de um cliente, entrada e saída de veículos.

Na estrutura “Cliente”, são registadas informações relevantes em relação ao cliente, tais como, Nome, NIF, cod_postal, e matrícula do carro associado ao mesmo.

Na estrutura “Entradas” é registado as matrículas dos carros que entraram no parque, e é usada a estrutura Data para registar o ano, mês, dia e o seu devido horário de entrada.

Na estrutura “EntradaseSaidas” é registado as matrículas dos carros que entraram e saíram do parque, e é usada a estrutura Data para registar o ano, mês, dia e o seu devido horário de entrada e saída.

Na estrutura “LugarEstacionamento” é registado o piso, a linha e a coluna de um lugar ocupado por um veículo.

1.2. Ler entradas

Para esta função, abrimos o ficheiro “entradas.txt” e registamos o dado relevante de cada coluna, nomeadamente, a matrícula, data de entrada e o lugar no qual o veículos ficou estacionado, sendo este lugar catalogado por piso, linha e coluna.

Cada coluna lhe será associada uma variável,e os dados serão guardados em memória para alteração, e leitura em iterações futuras.

1.3. Carregar Clientes

Para a função “carregarClientes” ocorre o mesmo processo, é aberto o ficheiro “clientes.txt”, que serve para mostrar os clientes que possuem subscrição no parque.

```
void carregarClientes(struct Cliente clientes[], int *num_clientes) {
    FILE *file;
    file = fopen( Filename: "clientes.txt", Mode: "r");

    if (file == NULL) {
        printf( format: "Erro ao abrir o arquivo clientes.txt\n");
        return;
    }

    while (*num_clientes < MAX_CLIENTES &&
           fscanf( stream: file, format: "%s %49[^\t] %s %d %d %d %d",
                   clientes[*num_clientes].matricula,
                   clientes[*num_clientes].nome,
                   clientes[*num_clientes].cod_postal,
                   &clientes[*num_clientes].NIF,
                   &clientes[*num_clientes].data_inscricao.ano,
                   &clientes[*num_clientes].data_inscricao.mes,
                   &clientes[*num_clientes].data_inscricao.dia) == 7) {
        clientes[*num_clientes].id_cliente = *num_clientes + 1; // Adicionado ID do cliente a começar no 1
        (*num_clientes)++;
    }
```

1.4. Ler Entradas e Saídas

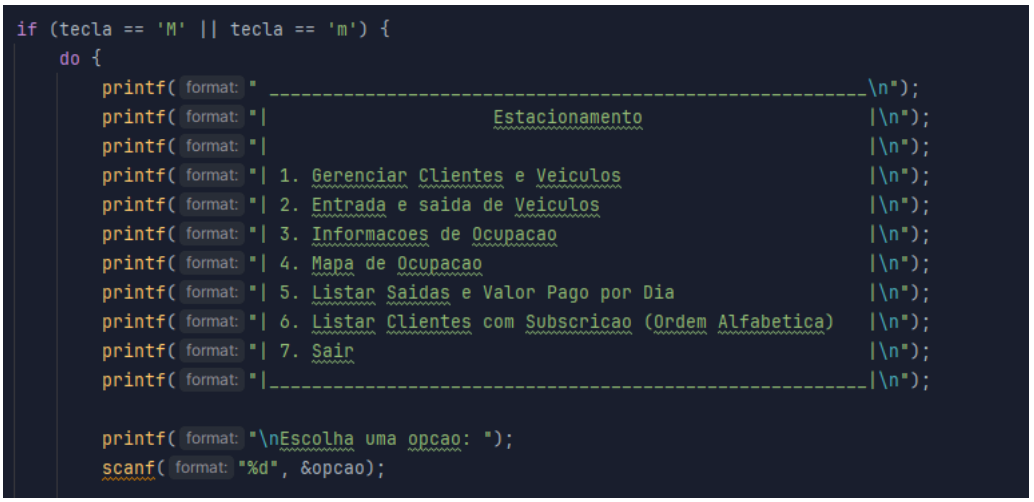
Uma vez mais, a função lerEntradasSaidasa abre o ficheiro entradas_e_saidas.txt, e, associa às estruturas a matrícula, e a sua respectivas data de entrada e saída do estacionamento.

```
void lerEntradaseSaidas(struct EntradaseSaidas entsaida[], int *num_EntradaseSaidas){
    FILE *file = fopen( Filename: "entradas_e_saidas.txt",  Mode: "r");
    if (file == NULL) {
        printf( format: "Erro ao abrir o arquivo.\n");
        return;
    }

    *num_EntradaseSaidas = 0;

    while(*num_EntradaseSaidas < MAX_ENTRIES &&
          fscanf( stream: file, format: "%s   %d %d %d %d %d %d %d %d %d %d ",
                &entsaida[*num_EntradaseSaidas].matriculas,
                &entsaida[*num_EntradaseSaidas].data_entrada.mes,
                &entsaida[*num_EntradaseSaidas].data_entrada.dia,
                &entsaida[*num_EntradaseSaidas].horaent,
                &entsaida[*num_EntradaseSaidas].minent,
                &entsaida[*num_EntradaseSaidas].data_entrada.ano,
                &entsaida[*num_EntradaseSaidas].data_entrada.mes,
                &entsaida[*num_EntradaseSaidas].data_entrada.dia,
                &entsaida[*num_EntradaseSaidas].horasaida,
                &entsaida[*num_EntradaseSaidas].minsaida) ==10) {
        (*num_EntradaseSaidas)++;
```

2. Navegação por menus

Algumas das principais funcionalidades incluem a gestão de clientes e veículos, registo de pagamentos de subscrições, entrada e saída de veículos, exibição de informações de ocupação, visualização de um mapa de ocupação, listagem de saídas e valores pagos por dia, listagem de clientes com subscrição e a opção de sair do programa.

Através do main(), o utilizador pode executar cada uma destas tarefas através das opções no menu, sendo ExibirMenuClientes() e ExibirMovimentoVeiculos() submenus para tarefas mais específicas.



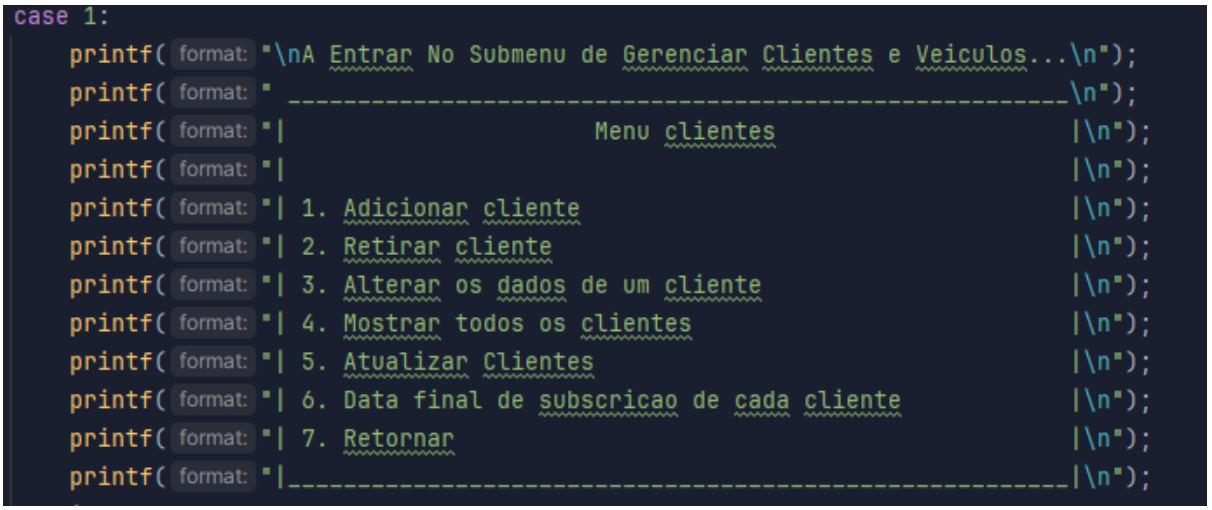
```
if (tecla == 'M' || tecla == 'm') {
    do {
        printf( format: " ____________________________________________________\n");
        printf( format: "|                    Estacionamento                  |\n");
        printf( format: "|                                                    |\n");
        printf( format: "| 1. Gerenciar Clientes e Veiculos                   |\n");
        printf( format: "| 2. Entrada e saida de Veiculos                     |\n");
        printf( format: "| 3. Informacoes de Ocupacao                         |\n");
        printf( format: "| 4. Mapa de Ocupacao                                |\n");
        printf( format: "| 5. Listar Saidas e Valor Pago por Dia              |\n");
        printf( format: "| 6. Listar Clientes com Subscricao (Ordem Alfabetica)|\n");
        printf( format: "| 7. Sair                                            |\n");
        printf( format: "|____________________________________________________|\n");

        printf( format: "\nEscolha uma opcao: ");
        scanf( format: "%d", &opcao);
```

Através do switch case, acedemos aos submenus e às suas devidas funções.

No case 1 acede-se ao submenu “Menu Clientes”.
No case 2 acede-se ao submenu “Menu Veículos”

2.1. ExibirMenuClientes()

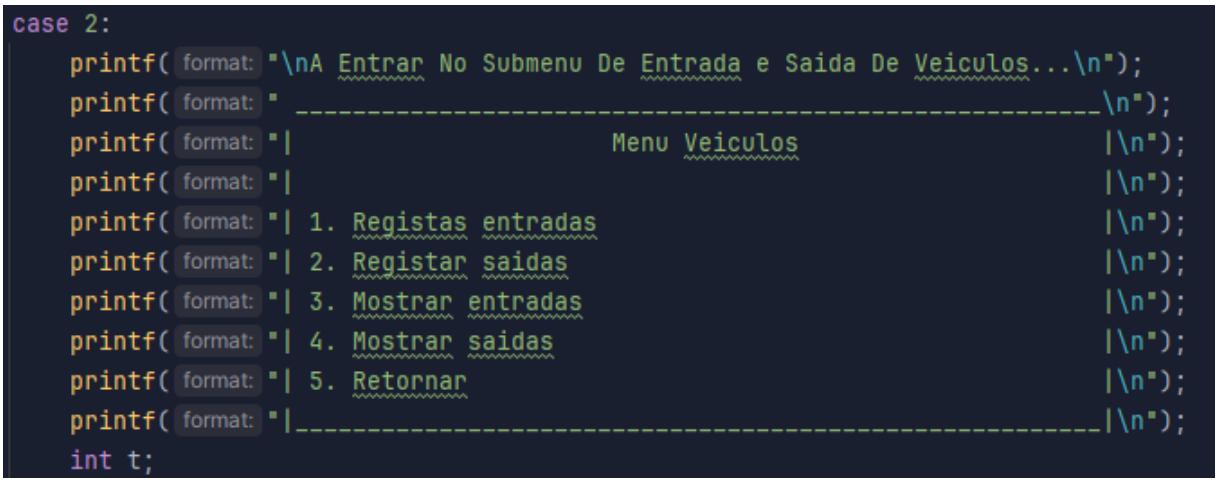
Este menu permite ao utilizar adicionar, remover, alterar um cliente com subscrição, respetivamente, do ficheiro.



```
case 1:
    printf( format: "\nA Entrar No Submenu de Gerenciar Clientes e Veiculos...\n");
    printf( format: " _____________________________________________________\n");
    printf( format: "|                    Menu clientes                    |\n");
    printf( format: "|                                                     |\n");
    printf( format: "| 1. Adicionar cliente                                |\n");
    printf( format: "| 2. Retirar cliente                                  |\n");
    printf( format: "| 3. Alterar os dados de um cliente                   |\n");
    printf( format: "| 4. Mostrar todos os clientes                        |\n");
    printf( format: "| 5. Atualizar Clientes                               |\n");
    printf( format: "| 6. Data final de subscricao de cada cliente         |\n");
    printf( format: "| 7. Retornar                                         |\n");
    printf( format: "|_____________________________________________________|\n");
```

2.2. ExibirMovimentoVeiculos()

Neste menu, o utilizador pode registar a entrada e saída de veículos.



```
case 2:
    printf( format: "\nA Entrar No Submenu De Entrada e Saida De Veiculos...\n");
    printf( format: " _____________________________________________________\n");
    printf( format: "|                    Menu Veiculos                    |\n");
    printf( format: "|                                                     |\n");
    printf( format: "| 1. Registas entradas                                |\n");
    printf( format: "| 2. Registar saidas                                  |\n");
    printf( format: "| 3. Mostrar entradas                                 |\n");
    printf( format: "| 4. Mostrar saidas                                   |\n");
    printf( format: "| 5. Retornar                                         |\n");
    printf( format: "|_____________________________________________________|\n");
    int t;
```

3. Funções do MainMenu()

```c
if (tecla == 'M' || tecla == 'm') {
    do {
        printf( format: " ____________________________________________________\n");
        printf( format: "|                    Estacionamento                  |\n");
        printf( format: "|                                                    |\n");
        printf( format: "| 1. Gerenciar Clientes e Veiculos                   |\n");
        printf( format: "| 2. Entrada e saida de Veiculos                     |\n");
        printf( format: "| 3. Informacoes de Ocupacao                         |\n");
        printf( format: "| 4. Mapa de Ocupacao                                |\n");
        printf( format: "| 5. Listar Saidas e Valor Pago por Dia              |\n");
        printf( format: "| 6. Listar Clientes com Subscricao (Ordem Alfabetica)|\n");
        printf( format: "| 7. Sair                                            |\n");
        printf( format: "|____________________________________________________|\n");

        printf( format: "\nEscolha uma opcao: ");
        scanf( format: "%d", &opcao);
```

3.1. Adicionar Cliente

Nesta função, é questionado ao utilizador os dados para um cliente que efetuou uma subscrição, nomeadamente, a matrícula a ele associado, NIF, código postal, e a data no qual efetuou subscrição.

```c
void addCliente(struct Cliente clientes[], int *num_clientes) {
    if (*num_clientes >= MAX_CLIENTES) {
        printf( format: "Limite maximo de clientes atingido.\n");
        return;
    }

    printf( format: "Digite os dados do cliente:\n");

    // Pede ao utilizador os dados do cliente
    printf( format: "Matricula: ");
    scanf( format: "%s", clientes[*num_clientes].matricula);
    getchar();

    printf( format: "Nome: ");
    scanf( format: " %49[^\n]", clientes[*num_clientes].nome);

    printf( format: "Codigo Postal: ");
    scanf( format: "%s", clientes[*num_clientes].cod_postal);

    printf( format: "NIF: ");
    scanf( format: "%d", &clientes[*num_clientes].NIF);

    printf( format: "Data de inscricao (ano mes dia): ");
    scanf( format: "%d %d %d", &clientes[*num_clientes].data_inscricao.ano,
          &clientes[*num_clientes].data_inscricao.mes,
          &clientes[*num_clientes].data_inscricao.dia);

    // Incrementa o ID do cliente
    clientes[*num_clientes].id_cliente = *num_clientes + 1;

    // Incrementa o número de clientes
    (*num_clientes)++;
    printf( format: "Cliente adicionado na memoria com sucesso.\n");
```

3.2. Retirar Cliente

Nesta função, mostramos ao utilizador quais os clientes já existentes e pedimos qual o id do cliente a remover, de seguida apaga o conteúdo de toda a linha da memória.

```c
void removerCliente(struct Cliente clientes[], int *num_clientes) {
    if (*num_clientes == 0) {
        printf( format: "Não existem clientes para remover.\n");
        return;
    }

    printf( format: "Clientes existentes:\n");
    for (int i = 0; i < *num_clientes; i++) {
        printf( format: "ID: %d\tNome: %s\n", clientes[i].id_cliente, clientes[i].nome);
    }

    printf( format: "Existem %d clientes. Escolha o ID do cliente para remover (de 1 a %d): ", *num_clientes, *num_clientes);

    int id_cliente;

    // Verifica se a leitura do ID do cliente foi bem-sucedida e está dentro do intervalo válido
    while (scanf( format: "%d", &id_cliente) != 1 || id_cliente < 1 || id_cliente > *num_clientes)
        printf( format: "Insira um numero válido (de 1 a %d): ", *num_clientes);

        // Limpa o buffer de entrada para evitar loops infinitos
        int c;
        while ((c = getchar()) != '\n' && c != EOF);
    }

    int indice = -1;

    // Procura o cliente pelo id_cliente
    for (int i = 0; i < *num_clientes; i++) {
        if (clientes[i].id_cliente == id_cliente) {
            indice = i;
            break;
        }
    }

    // Se o cliente for encontrado, remove-o da memória
    if (indice != -1) {
        // Move os clientes subsequentes uma posição para trás na memória
        for (int i = indice; i < *num_clientes - 1; i++) {
            clientes[i] = clientes[i + 1];
        }

        // Decrementa o número de clientes
        (*num_clientes)--;
        printf( format: "Cliente removido da memoria com sucesso.\n");
    } else {
        printf( format: "Cliente com ID %d nao encontrado.\n", id_cliente);
    }
}
```

3.3. Alterar Dados de Cliente

Para esta função é perguntado ao utilizador qual o id de cliente a alterar, e serão perguntadas as mesmas questões da função para adicionar cliente.

```c
void alterarCliente(struct Cliente clientes[], int *num_clientes) {
    if (*num_clientes == 0) {
        printf( format: "Nao existem clientes para alterar.\n");
        return;
    }

    printf( format: "Clientes existentes:\n");
    for (int i = 0; i < *num_clientes; i++) {
        printf( format: "ID: %d\tNome: %s\n", clientes[i].id_cliente, clientes[i].nome);
    }

    printf( format: "Existem %d clientes. Escolha o ID do cliente para alterar (de 1 a %d): ", *num_clientes, *num_clientes);

    int id_cliente;

    // Verifica se a leitura do ID do cliente foi bem-sucedida e está dentro do intervalo válido
    while (scanf( format: "%d", &id_cliente) != 1 || id_cliente < 1 || id_cliente > *num_clientes) {
        printf( format: "Insira um numero valido (de 1 a %d): ", *num_clientes);

        // Limpa o buffer de entrada para evitar loops infinitos
        int c;
        while ((c = getchar()) != '\n' && c != EOF);
    }

    // Encontrar o índice do cliente com base no ID fornecido
    int indice = -1;
    for (int i = 0; i < *num_clientes; i++) {
        if (clientes[i].id_cliente == id_cliente) {
            indice = i;
            break;
```

3.4. Mostrar todos os Clientes

Para esta função, o programa percorre linha por linha e da print ao conteúdo da mesma, sendo que, estes são os dados do cliente.

```c
void mostrarClientes(struct Cliente clientes[], int num_clientes) {
    if (num_clientes == 0) {
        printf( format: "Nao existem clientes para mostrar.\n");
        return;
    }

    printf( format: "\nCLIENTES\n");
    printf( format: "------------------------------------------------------------\n");
    for (int i = 0; i < num_clientes; i++) {
            if (clientes[i].id_cliente != 0) {
                printf( format: "ID Cliente: %d\nMatricula: %s\nNome: %s\nCodigo Postal: %s\nNIF: %d\nData Inscricao: %d/%d/%d\n",
                clientes[i].id_cliente,
                clientes[i].matricula,
                clientes[i].nome,
                clientes[i].cod_postal,
                clientes[i].NIF,
                clientes[i].data_inscricao.ano,
                clientes[i].data_inscricao.mes,
                clientes[i].data_inscricao.dia);
        printf( format: "------------------------------------------------------------\n");
        }
    }
}
```

3.5. Atualizar Clientes

Esta função, depois das alterações feitas pelo utilizador, irá abrir o ficheiro clientes.txt e fazer as alterações pedidas pelo utilizador.

```c
void atualizarClientes(struct Cliente clientes[], int num_clientes) {
    // Abre o arquivo em modo de escrita (apaga tudo e escreve desde o inicio)
    FILE *file;
    file = fopen( Filename: "clientes.txt",  Mode: "w");

    if (file == NULL) {
        printf( format: "Erro ao abrir o arquivo clientes.txt\n");
        return;
    }

    // Escreve os dados dos clientes no arquivo
    for (int i = 0; i < num_clientes; i++) {
        fprintf( stream: file,  format: "%s\t%s\t%s\t%d\t%d\t%d\t%d\n",
                clientes[i].matricula,
                clientes[i].nome,
                clientes[i].cod_postal,
                clientes[i].NIF,
                clientes[i].data_inscricao.ano,
                clientes[i].data_inscricao.mes,
                clientes[i].data_inscricao.dia);
    }

    // Fecha o arquivo
    fclose(file);
    printf( format: "Clientes atualizados no arquivo clientes.txt.\n");
}
```

3.6. Data Final da Subscrição

Para esta função, para cada cliente com subscrição, mostra o seu nome, e a data de quando a mesma acaba.

```c
void mostrarDataFinalSubscricao(struct Cliente *clientes, int num_clientes) {
    printf( format: "\nCLIENTES\n");
    printf( format: "------------------------------------------------------------\n");

    for (int i = 0; i < num_clientes; i++) {
        if (clientes[i].id_cliente != 0) {
            // Adicionando 1 ao ano de inscrição
            adicionarUmAno( cliente: &clientes[i]);

            // Mostrando os detalhes do cliente com a nova data
            printf( format: "ID Cliente: %d\nNome: %s\nData final de subscricao: %d/%d/%d\n",
                   clientes[i].id_cliente,
                   clientes[i].nome,
                   clientes[i].data_inscricao.ano,
                   clientes[i].data_inscricao.mes,
                   clientes[i].data_inscricao.dia);
            printf( format: "------------------------------------------------------------\n");
        }
    }
}
```

3.7. Registar Entrada

Para esta função, é pedido qual a matrícula, ano, mês, dia, hora e minuto de entrada, e após isso o programa fornece automaticamente um lugar livre no parque, escolhido aleatoriamente:

```c
void adicionarEntrada(struct Entradas entradas[], int *num_entradas, struct LugarEstacionamento lugares[], int *totalLugares) {
    // Lê entradas existentes do arquivo para a memória
    lerentradas(entradas, num_entradas, lugares, totalLugares);

    char matricula[MAX_MATRICULA_LENGTH];
    int ano, mes, dia, hora, min;

    printf( format: "Insira a matricula do carro que entrou: ");
    scanf( format: "%s", matricula);

    // Gera a sequência aleatória
    char sequencia[5];
    gerarSequenciaAleatoria(sequencia);

    // Solicita ao usuário inserir o ano, mês, dia, hora e minutos
    printf( format: "Insira o ano, mes, dia, hora e minutos (formato AAAA MM DD HH MM): ");
    scanf( format: "%d %d %d %d %d", &ano, &mes, &dia, &hora, &min);

    // Adiciona a nova entrada ao array de entradas
    strcpy( Dest: entradas[*num_entradas].matriculas, Source: matricula);
    entradas[*num_entradas].data_entrada.ano = ano;
    entradas[*num_entradas].data_entrada.mes = mes;
    entradas[*num_entradas].data_entrada.dia = dia;
    entradas[*num_entradas].hora = hora;
    entradas[*num_entradas].min = min;

    // Adiciona o lugar de estacionamento ao array de lugares
    lugares[*totalLugares].piso = rand() % 3 + 1; // exemplo: gera um número aleatório entre 1 e 3 para o piso
    lugares[*totalLugares].linha = 'A' + rand() % 5; // exemplo: gera uma letra aleatória de A a E para a linha
    lugares[*totalLugares].coluna = rand() % 10 + 1; // exemplo: gera um número aleatório entre 1 e 10 para a coluna

    (*num_entradas)++;
    (*totalLugares)++;
```

3.8. Registar Saída

Nesta função, registamos um veículos que entrou no parque através da matrícula, anotamos a data de entrada e saída e escrevemos no ficheiro “entradas.txt”.

```c
void registrarSaida(struct Entradas entradas[], int *num_entradas, struct EntradaseSaidas saidas[], int *num_saidas) {
    char matricula[MAX_MATRICULA_LENGTH];
    int ano, mes, dia, hora, min;

    printf( format: "Insira a matricula do carro que saiu: ");
    scanf( format: "%s", matricula);

    // Procura a entrada na estrutura de entradas
    int indice_entrada = -1;
    for (int i = 0; i < *num_entradas; i++) {
        if (strcmp(entradas[i].matriculas, matricula) == 0) {
            indice_entrada = i;
            break;
        }
    }

    if (indice_entrada == -1) {
        printf( format: "Veículo não encontrado na lista de entradas.\n");
        return;
    }

    printf( format: "Insira o ano, mes, dia, hora e minutos (formato AAAA MM DD HH MM): ");
    scanf( format: "%d %d %d %d %d", &ano, &mes, &dia, &hora, &min);

    // Adiciona a nova saída ao array de saídas
    strcpy( Dest: saidas[*num_saidas].matriculas,  Source: matricula);
    saidas[*num_saidas].data_entrada = entradas[indice_entrada].data_entrada;
    saidas[*num_saidas].horaent = entradas[indice_entrada].hora;
    saidas[*num_saidas].minent = entradas[indice_entrada].min;
    saidas[*num_saidas].data_saida.ano = ano;
    saidas[*num_saidas].data_saida.mes = mes;
    saidas[*num_saidas].data_saida.dia = dia;
    saidas[*num_saidas].horasaida = hora;
    saidas[*num_saidas].minsaida = min;

    // Remove a linha correspondente ao carro que saiu do arquivo "entradas.txt"
    removerLinha("entradas.txt", matricula);
```

3.9. Mostrar Entradas

Esta função, mostra a lista de entrada de veículos, nomeadamente a sua matrícula data de entrada, e o lugar de estacionamento a ele atribuído.

```c
void mostrarEntradas(struct Entradas entradas[], int num_entradas, struct LugarEstacionamento lugares[], int totalLugares) {
    printf( format: "===== ENTRADAS =====\n");
    for (int i = 0; i < num_entradas; i++) {
        printf( format: "Matricula: %s\n", entradas[i].matriculas);
        printf( format: "Data de Entrada: %d/%d/%d %02d:%02d\n",
               entradas[i].data_entrada.dia, entradas[i].data_entrada.mes, entradas[i].data_entrada.ano,
               entradas[i].hora, entradas[i].min);
        printf( format: "Lugar de Estacionamento: Piso %d, Linha %c, Coluna %d\n",
               lugares[i].piso, lugares[i].linha, lugares[i].coluna);
        printf( format: "----------------------\n");
    }
    printf( format: "Total de Entradas: %d\n", num_entradas);
}
```

3.10. Mostrar Saídas

Nesta função, mostra a lista de saída de veículos, nomeadamente a sua matrícula e data de saída.

```c
void mostrarSaidas(struct EntradaseSaidas saidas[], int num_saidas) {
    printf( format: "===== SAIDAS =====\n");
    for (int i = 0; i < num_saidas; i++) {
        printf( format: "Matricula: %s\n", saidas[i].matriculas);
        printf( format: "Data de Entrada: %d/%d/%d %02d:%02d\n",
               saidas[i].data_entrada.dia, saidas[i].data_entrada.mes, saidas[i].data_entrada.ano,
               saidas[i].horaent, saidas[i].minent);
        printf( format: "Data de Saida: %d/%d/%d %02d:%02d\n",
               saidas[i].data_saida.dia, saidas[i].data_saida.mes, saidas[i].data_saida.ano,
               saidas[i].horasaida, saidas[i].minsaida);
        printf( format: "--------------------\n");
        printf( format: "\n");
    }
    printf( format: "Total de Saidas: %d\n", num_saidas);
}
```

3.11. Informação de ocupação

A terceira opção do menu, “Informações de ocupação” analisa os dados, conta os lugares ocupados, subtrai ao total, e mostra o número de lugares livres ao utilizador.

Não só isso mas também, mostra o piso, linha e coluna do lugar onde uma matrícula especificada se encontra.

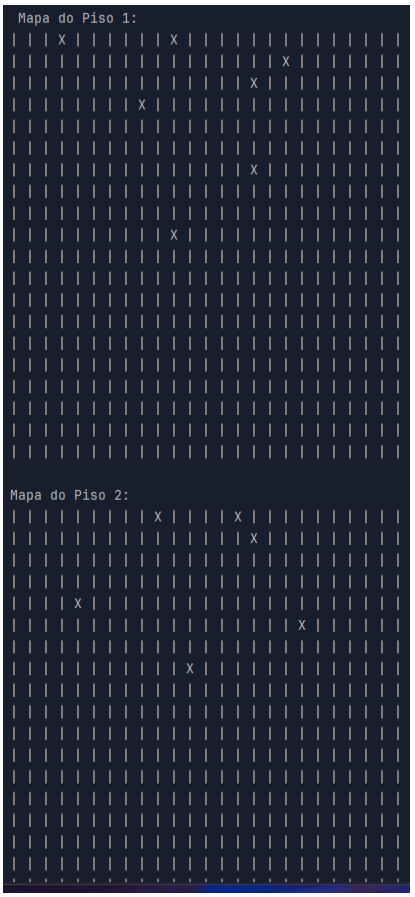
```c
case 3:
        lugaresLivres = MAX_LUGARES - num_dadosest;
        printf( format: "Lugares livres: %d\n", lugaresLivres);

        for (int i = 0; i < num_dadosest; i++) {
            printf( format: "Matricula: %s - Lugar: Piso %d, Linha %c, Coluna %d\n",
                   dadosent[i].matriculas,
                   dadosest[i].piso,
                   dadosest[i].linha,
                   dadosest[i].coluna);
        }
        break;
```

3.12.Mapa de ocupação

A quarta opção do menu, através da função “lerentradas”, retira qual piso, linha e coluna se encontra estacionado, através do código do lugar.

Por fim, desenha 3 grelhas 20x25, preenchendo com um “X” os lugares ocupados.

```c
void desenharMapa(struct LugarEstacionamento lugares[], int *totalLugares) {
    int i, j, piso;

    // A desenhar o mapa para cada piso
    for (piso = 1; piso <= 3; piso++) {
        printf( format: "Mapa do Piso %d:\n", piso);

        // Inicializando a matriz do mapa com espaços vazios
        char mapa[20][25];
        for (i = 0; i < 20; i++) {
            for (j = 0; j < 25; j++) {
                mapa[i][j] = '|'; // espaço vazio em vez de '|'
            }
        }

        // Preenchendo o mapa com as informações dos lugares de estacionamento
        for (int k = 0; k < *totalLugares; k++) {
            struct LugarEstacionamento lugar = lugares[k];

            // Verificando se o piso é o mesmo
            if (lugar.piso == piso) {
                int linha = lugar.linha - 'A';
                int coluna = lugar.coluna - 1;
                mapa[linha][coluna] = 'X';
            }
        }

        // A exibir o mapa
        for (i = 0; i < 20; i++) {
            for (j = 0; j < 25; j++) {
                printf( format: "%c ", mapa[i][j]);
            }
            printf( format: "\n");
```



3.13.Saídas e valores pagos

Na quinta opção, através das informações nos ficheiros de entrada e saída, é feito um cálculo de quanto o cliente tem de pagar, seguindo os padrões da tabela de preços:

Por fim, mostra os pagamentos ao utilizador através da consola, e cria um ficheiro de texto com a lista de todos os pagamentos efetuados.

```
void consulta_pagamento(struct EntradaseSaidas entsaida[], int *num_EntradaseSaidas) {

    int ano_consulta, mes_consulta, dia_consulta;
    printf( format: "Digite a data de consulta (AAAA MM DD): ");
    scanf( format: "%d %d %d", &ano_consulta, &mes_consulta, &dia_consulta);

    float custo_total;

    printf( format: "Carros que sairam na data %d-%02d-%02d e valor pago:\n", ano_consulta, mes_consulta, dia_consulta);

    for(int i; i < *num_EntradaseSaidas; i++) {

        if (entsaida[i].data_saida.ano && entsaida[i].data_saida.mes && entsaida[i].data_saida.dia == dia_consulta) {
            custo_total = 0.0;
            float total_horas = 0.0;

            while (entsaida[i].horaent != entsaida[i].horasaida || entsaida[i].minent != entsaida[i].minsaida) {
            if (entsaida[i].horaent >= 8.0 && entsaida[i].horaent < 22.0) {
                custo_total += T1 / 4.0;
            } else {
                custo_total += T2 / 4.0;
            }

            entsaida[i].minent += 15.0;

            if (entsaida[i].minent >= 60.0) {
                entsaida[i].minent -= 60.0;
                entsaida[i].horaent += 1.0;
            }

            if (entsaida[i].horaent >= 24.0) {
                entsaida[i].horaent -= 24.0;
            }
        }
```

```c
void calculo_final(struct EntradaseSaidas entsaida[], int *num_EntradaseSaidas, struct Cliente clientes[], int num_clientes) {
    FILE *pagamentos = fopen( Filename: "Pagamentos.txt",  Mode: "w");
    if (pagamentos == NULL) {
        printf( format: "Erro ao abrir o arquivo.\n");
        return;
    }

    for (int i = 0; i < *num_EntradaseSaidas; i++) {
        // Verifica se a matricula esta na lista de clientes
        int matricula_encontrada = 0;
        for (int j = 0; j < num_clientes; j++) {
            if (strcmp(entsaida[i].matriculas, clientes[j].matricula) == 0) {
                matricula_encontrada = 1;
                break;
            }
        }

        if (matricula_encontrada) {
            // Se a matricula for encontrada o custo deverá ser igual a 0
            printf( format: "Matricula: %s Custo total: 0.0 (O cliente esta exento)\n", entsaida[i].matriculas);
            fprintf( stream: pagamentos,  format: "Matricula: %s Custo total: 0.0 (O cliente esta exento)\n", entsaida[i].matriculas);
        } else {
            // Arredonda para cima o tempo de entrada para o próximo intervalo de 15 minutos
            if (fmod( X: entsaida[i].minent,  Y: 15.0) > 0) {
                entsaida[i].minent = ((int) (entsaida[i].minent / 15) + 1) * 15;
                if (entsaida[i].minent >= 60.0) {
                    entsaida[i].minent -= 60.0;
                    entsaida[i].horaent++;
                }
            }

            // Arredonda para cima o tempo de saída para o próximo intervalo de 15 minutos
            if (fmod( X: entsaida[i].minsaida,  Y: 15.0) > 0) {
                entsaida[i].minsaida = ((int) (entsaida[i].minsaida / 15) + 1) * 15;
                if (entsaida[i].minsaida >= 60.0) {
                    entsaida[i].minsaida -= 60.0;
                    entsaida[i].horasaida++;
                }
```

```
    // Cálculo do custo total para o estacionamento
    float custo_total = 0.0;

    // Cálculo para minutos de entrada até minutos de saída
    while (entsaida[i].horaent != entsaida[i].horasaida || entsaida[i].minent != entsaida[i].minsaida) {
        if (entsaida[i].horaent >= 8.0 && entsaida[i].horaent < 22.0) {
            custo_total += T1 / 4.0;
        } else {
            custo_total += T2 / 4.0;
        }

        entsaida[i].minent += 15.0;

        if (entsaida[i].minent >= 60.0) {
            entsaida[i].minent -= 60.0;
            entsaida[i].horaent += 1.0;
        }

        if (entsaida[i].horaent >= 24.0) {
            entsaida[i].horaent -= 24.0;
        }
    }

    // Limita o custo total a T3 se ultrapassar
    if (custo_total > T3) {
        custo_total = T3;
    }

    float dias_estacionado = entsaida[i].data_saida.dia - entsaida[i].data_entrada.dia;

    // Imprime e escreve no arquivo de pagamentos
    printf( format: "Matricula: %s Custo total: %.2f\n", entsaida[i].matriculas, custo_total + (8.0 * dias_estacionado));
    fprintf( stream: pagamentos, format: "Matricula: %s Custo total: %.2f\n", entsaida[i].matriculas,
            custo_total + (8.0 * dias_estacionado));
}
```

3.14.Listar clientes com subscrição

Para a última, o programa acede aos dados registados pela função carregar clientes, e escreve por ordem alfabética os clientes presentes no ficheiro, já que este se destina apenas a clientes com subscrição.

```
void mostrarDataFinalSubscricao(struct Cliente *clientes, int num_clientes) {
    printf( format: "\nCLIENTES\n");
    printf( format: "------------------------------------------------------------\n");

    for (int i = 0; i < num_clientes; i++) {
        if (clientes[i].id_cliente != 0) {
            // Adicionando 1 ao ano de inscrição
            adicionarUmAno( cliente: &clientes[i]);

            // Mostrando os detalhes do cliente com a nova data
            printf( format: "ID Cliente: %d\nNome: %s\nData final de subscricao: %d/%d/%d\n",
                    clientes[i].id_cliente,
                    clientes[i].nome,
                    clientes[i].data_inscricao.ano,
                    clientes[i].data_inscricao.mes,
                    clientes[i].data_inscricao.dia);
            printf( format: "------------------------------------------------------------\n");
        }
```

4. Conclusões

O projeto de gestão de estacionamento em linguagem C proporcionou uma implementação prática e modular. A estrutura organizada do código facilita a manutenção e expansão futura.

Apesar de fornecer funcionalidades abrangentes, como o registo de entradas/saídas e a exibição de informações de ocupação, algumas áreas, como o registo de pagamentos de subscrições, exigem desenvolvimento adicional.

A aplicação demonstrou eficiência na utilização de estruturas de decisão e leitura de dados de arquivos. No entanto, otimizações adicionais podem ser exploradas para aprimorar a eficiência global.