Relatório do projeto PPP

Pedro Miguel Queiroga Trindade nº2022230848
João António Faustino Vaz nº2022231087

Este projeto tem como objetivo a criação de um sistema de agendamento de serviços para clientes de uma oficina. No início, começámos por criar algumas estruturas de dados: struct “data” onde armazena as informações relativamente ao horário da reserva e contém as seguintes variáveis: ano, mês, dia, hora, minuto. Struct “cliente” que armazena as informações dos clientes, como o seu nome, número de telemóvel, tipo de serviço e ainda a struct “data” que tal como explicado armazena a data da reserva. Struct noLista: representa um nó da lista que será utilizada para armazenar as reservas dos clientes. Cada nó contém uma estrutura do tipo cliente e um ponteiro para o próximo nó da lista.

```c
typedef struct data{
    int ano;
    int mes;
    int dia;
    int hora;
    int minuto;
}data;

typedef struct{
    char nome[MAX];
    int numtel;
    char servico[10];
    data data;
}cliente;

typedef struct noLista{
    cliente cliente;
    struct noLista *prox;
}noLista;
```

Para além disto, o código contém diversas funções: “start()” que de um modo geral cria e retorna uma lista vazia, onde se faz uma declaração de uma variável aux do tipo Lista e que será usada para armazenar o ponteiro para o início da lista, declaração de uma variável “d” do tipo data que será usada para configurar os valores iniciais da estrutura “data” e em que que todos os campos desta estrutura são definidos como 0, declaração de uma variável “c” do tipo cliente, que será usada para configurar os valores iniciais do cliente. Neste caso, o nome do cliente é uma string vazia, o número de telefone é 0, o serviço é uma string vazia e a data é a estrutura “d” configurada e mencionada anteriormente.Para além disso é feita a alocação de memória da variável aux através da função malloc(), após isso é feita a verificação dessa alocação, caso não seja bem sucedida é exibida uma mensagem de erro e o programa é encerrado, caso contrário o programa segue, o campo cliente do nó aux é atribuído com a estrutura “c” configurada anteriormente, e o ponteiro prox do nó aux é definido como NULL, indicando que não há um próximo nó na lista. Após isso a função retorna o lista aux.

```c
Lista start(){
    Lista aux;
    data d;
    d.ano=0;
    d.mes=0;
    d.dia=0;
    d.hora=0;
    d.minuto=0;
    cliente c = {"",0,"",d};
    aux = (Lista) malloc (sizeof (noLista));
    if (aux == NULL) {
        printf("Erro a alocar memória!\n");
        exit(-1);
    }else{
        aux->cliente = c;
        aux->prox = NULL;
    }
    return aux;
}
```

A função tempo_servico, através da função strcmp() retorna a duração em minutos de um determinado serviço, com base no seu nome.

```c
int tempo_servico(char*servico){
    if (strcmp(servico,"Manutencao") == 0){
        return 60;
    }else{
        return 30;
    }
}
```

A função cmpDatas, implementa uma comparação entre datas levando em consideração os parâmetros da estrutura “data” (ano, mês, dia, hora e minutos);A função compara cada campo e retorna 1 se o campo do cliente for maior que o campo atual, -1 se for menor, e passa para o próximo campo se os valores forem iguais.Se todas as comparações resultarem em igualdade, a função retorna 0, indicando que as datas são iguais.

```c
int cmpDatas(data*atual,data*cliente_atual){
    if(cliente_atual->ano > atual->ano){
        return 1;
    }else if(cliente_atual->ano < atual->ano){
        return -1;
    }else if(cliente_atual->mes > atual->mes){
        return 1;
    }else if(cliente_atual->mes < atual->mes){
        return -1;
    }else if(cliente_atual->dia > atual->dia){
        return 1;
    }else if (cliente_atual->dia < atual->dia){
        return -1;
    }else if(cliente_atual->hora > atual->hora){
        return 1;
    }else if(cliente_atual->hora < atual->hora){
        return -1;
    }else if(cliente_atual->minuto > atual->minuto){
        return 1;
    }else if(cliente_atual->minuto < atual->minuto){
        return -1;
    }else{
        return 0;
    }
}
```

A função procura tem como objetivo procurar na lista (2 a 2) o local adequado para inserir o cliente d, comparando a data do anterior “ant_reservas”e o proximo “atual” atualizando os ponteiros ant_reservas e atual para os elementos corretos.

```c
void procura(Lista lista, cliente cliente_atual, Lista *ant_reservas, Lista *atual){
    *ant_reservas = lista;
    *atual = lista->prox;
    while ((*atual) != NULL && cmpDatas(&(*atual)->cliente.data,&cliente_atual.data)== 1){
        *ant_reservas = *atual;
        *atual = (*atual)->prox;
    }
}
```

A função cmpDuração2, através de um if verifica se duas datas forem no mesmo dia, converte para minutos as duas datas e caso a diferença da data anterior pela atual seja maior ou igual que a duração do serviço retorna 1, caso contrário retorna 0. É usada esta função na funcao marcar() quando o prox é NULL, ou seja, só compara o elemento a inserir com o anterior “ant_reservas”.

```
int cmpDuracao2(cliente*ant_reservas,cliente*c) {
    if(ant_reservas->data.ano==c->data.ano && ant_reservas->data.mes==c->data.mes && ant_reservas->data.dia==c->data.dia){
        if(((c->data.hora*60+c->data.minuto)-(ant_reservas->data.hora*60+ant_reservas->data.minuto))>=tempo_servico(ant_reservas->servico)){
            return 1;
        }else{
            return 0;
        }
    }else{
        return 1;
    }
}
```

A função cmpDuracao1, muito similar à cmpDuração2, compara a duração de um serviço com as reservas anteriores e posteriores de um cliente, com base nas suas datas.

```
int cmpDuracao1(cliente*ant_reservas,cliente*c,cliente*atual,int duracao_servico) {
    if(ant_reservas->data.ano==c->data.ano && ant_reservas->data.mes==c->data.mes && ant_reservas->data.dia==c->data.dia){
        if(((c->data.hora*60+c->data.minuto)-(ant_reservas->data.hora*60+ant_reservas->data.minuto))>=duracao_servico){
            return 1;
        }else{
            return 0;
        }
    }else if(atual->data.ano==c->data.ano && atual->data.mes==c->data.mes && atual->data.dia==c->data.dia){
        if(((atual->data.hora*60+atual->data.minuto)-(c->data.hora*60+c->data.minuto))>=duracao_servico){
            return 1;
        }else{
            return 0;
        }
    }else{
        return 1;
    }
}
```

A função elimina_cliente recebe a lista e a struct “cliente” que representa o cliente a ser eliminado. Em primeiro lugar, utilizamos a função procura para encontrar o cliente a ser removido da lista e depois de verificarmos que o cliente foi encontrado atualizamos o ponteiro anterior do nó para apontar para o nó seguinte, removendo o cliente. Por fim, liberamos a memória do cliente através do free.

```
void elimina_cliente(Lista lista, cliente c){
    Lista ant, atual;
    procura (lista,c,&ant, &atual);
    if (atual != NULL) {
        ant->prox = atual->prox;
        free (atual);
    }
}
```

A função marcar é responsável por agendar uma reserva ou pré-reserva para um cliente. Ela recebe os parâmetros: `reservas`: um ponteiro para o início da lista das reservas. ,`pré reservas`: um ponteiro para o início da lista das pré-reservas, ficheiro_reservas`: um ponteiro para o arquivo de reservas, `ficheiro_prereservas`: um ponteiro para o arquivo de pré-reservas e `c`: a estrutura do tipo `cliente` que representa o cliente a ser agendado. A função começa por alocar memória para dois novos nós da lista : `ptr` e `ptr_prereservas`. Em seguida, a função verifica se a alocação de memória foi bem-sucedida para ambos os nós. Se a alocação de memória for bem-sucedida, a função prossegue.Em seguida o cliente, antes de verificar se é possivel colocar nas reservas entre 2 clientes(cmpDuracao1) ou entre o anterior(cmpDuracao2), é colocado nas reservas, após ser colocado, sera verificado se é possivel estar reservado naquele horario, ou seja, verifica se tanto o anterior como o que foi colocado, conseguem executar os serviços sem os sobrepor. Caso possa ficar reservado, é escrito no ficheiro das reservas, todos os dados do cliente. Caso não possa estar reservado,usamos a função procura na listas das prereservas para encontrar o lugar para ser colocado nas prereservas. este é eliminado da lista das reservas através da função “elimina_cliente” e é colocado nas listas das pre-reservas e escrito no ficheiro das pre-reservas todos os seus dados.

```c
void marcar(Lista reservas,Lista prereservas,FILE*ficheiro_reservas,FILE*ficheiro_prereservas,cliente c){
    Lista ptr,ptr_prereservas,ant_reservas,atual_reservas,ant_prereservas,atual_prereservas;
    ptr = (Lista) malloc (sizeof (noLista));
    ptr_prereservas = (Lista) malloc (sizeof (noLista));

    int duracao_servico;
    duracao_servico = tempo_servico(c.servico);

    if (ptr != NULL && ptr_prereservas!= NULL) {
        ptr->cliente = c;
        ptr_prereservas->cliente = c;

        /*COLOCA O CLIENTE NA LISTA DAS RESERVAS*/
        procura(reservas, c, &ant_reservas, &atual_reservas);
        ptr->prox = ant_reservas->prox;
        ant_reservas->prox = ptr;

        /*VERIFICAR SE É POSSIVEL COLOCAR O CLIENTE NA LISTAS DAS RESERVAS E SE NAO FOR, LISTAR NAS PRE-RESERVAS*/
        if(ant_reservas!=NULL && ptr->prox!=NULL){
            if(cmpDuracao1(&(ant_reservas->cliente),&(ptr->cliente),&(ptr->prox->cliente),duracao_servico)==0){
                printf("Horário Indisponivel! %s %d %02d/%02d/%02d %02d:%02d %s sera colocado nas pre-reservas\n",ptr->cliente.nome,ptr->cliente.
                procura(prereservas,c,&ant_prereservas,&atual_prereservas);
                ptr_prereservas->prox = ant_prereservas->prox;
                ant_prereservas->prox = ptr_prereservas;
                fprintf(ficheiro_prereservas,"%s %d %02d/%02d/%02d %02d:%02d %s\n",ptr->cliente.nome,ptr->cliente.numtel,ptr->cliente.data.dia,ptr
                elimina_cliente(reservas,ptr->cliente);
                return;
            }
        }else if (ant_reservas!=NULL && ptr->prox==NULL){
            if(cmpDuracao2(&(ant_reservas->cliente),&(ptr->cliente))==0){
                printf("Horário Indisponivel! %s %d %02d/%02d/%02d %02d:%02d %s sera listado nas pre-reservas\n",ptr->cliente.nome,ptr->cliente.nu
                procura(prereservas,c,&ant_prereservas,&atual_prereservas);
                ptr_prereservas->prox = ant_prereservas->prox;
                ant_prereservas->prox = ptr_prereservas;
                /*ESCREVER NO FICHEIRO DAS PRE-RESERVAS OS DADOS DO CLIENTE*/
                fprintf(ficheiro_prereservas,"%s|%d %02d/%02d/%02d %02d:%02d %s\n",ptr->cliente.nome,ptr->cliente.numtel,ptr->cliente.data.dia,ptr
                elimina_cliente(reservas,ptr->cliente);
                return;
            }
        }
        /*ESCREVER NO FICHEIRO DAS RESERVAS OS DADOS DO CLIENTE*/
        fprintf(ficheiro_reservas,"%s|%d %02d/%02d/%02d %02d:%02d %s\n",ptr->cliente.nome,ptr->cliente.numtel,ptr->cliente.data.dia,ptr->cliente.
    }
}
```

A função dados_linha é responsável por extrair os dados de uma linha de texto e preencher a estrutura do tipo cliente com esses dados. A função utiliza a função strtok que separa os elementos em tokens delimitados por (“|”),(“ “).

```c
void dados_linha(char*linha,cliente*c){

    char*token=strtok(linha,"|");
    strcpy(c->nome,token);

    token=strtok(NULL," ");
    sscanf(token,"%d",&c->numtel);

    token=strtok(NULL," ");
    sscanf(token,"%d/%d/%d",&c->data.dia,&c->data.mes,&c->data.ano);

    token=strtok(NULL," ");
    sscanf(token,"%d:%d",&c->data.hora,&c->data.minuto);

    token=strtok(NULL," ");
    sscanf(token,"%s",c->servico);

}
```

A função imprimir_crescente imprime a lista por ordem crescente relativamente à data.
Por outro lado, também criamos uma função de nome imprimir_decrescente que imprime de forma decrescente através da criação de uma pilha.

```c
void imprimir_crescente(Lista lista){
    Lista ptr=lista->prox;
    while(ptr!=NULL){
        printf("%s %d %02d/%02d/%02d %02d:%02d %s\n",ptr->cliente.nome,ptr->cliente.numtel,ptr->cliente.data.dia,ptr->cliente.data.mes,
        ptr->cliente.data.ano,ptr->cliente.data.hora,ptr->cliente.data.minuto,ptr->cliente.servico);
        ptr=ptr->prox;
    }
}

void imprimir_decrescente(Lista lista) {
    Lista ptr = lista->prox;
    Lista pilha = NULL;

    while (ptr != NULL)
        Lista novoNo = (Lista)malloc(sizeof(noLista));
        if (novoNo != NULL) {
            novoNo->cliente = ptr->cliente;
            novoNo->prox = pilha;
            pilha = novoNo;
            ptr = ptr->prox;
        }else{
            printf("Erro a alocar memoria\n");
            exit(-1);
        }
    }
    while (pilha != NULL) {
        printf("%s %d %02d/%02d/%02d %02d:%02d %s\n", pilha->cliente.nome, pilha->cliente.numtel, pilha->cliente.data.dia,
        pilha->cliente.data.mes, pilha->cliente.data.ano, pilha->cliente.data.hora, pilha->cliente.data.minuto, pilha->cliente.servico);
        Lista temp = pilha;
        pilha = pilha->prox;
        free(temp);
    }}
```

```c
typedef noLista *Lista
```

A função verificacao_dados verifica se os dados inseridos como por exemplo o mês e o dia existem e se o número de telemóvel tem 9 digitos. Para além disso também verifica se o horário não antecede as 8 horas do dia e também não excede as 18 horas, como predefinido.

```
int verificacao_dados(cliente c){
    int cont=0;
    int temp_num=c.numtel;
    int flag;
    /*Verificar se o mes que o cliente insere existe e se o dia desse mes tambem existe*/
    int array[12][2]={{1,31},{2,28},{3,31},{4,30},{5,31},{6,30},{7,31},{8,31},{9,30},{10,31},{11,30},{12,31}};
    if(c.data.mes<=12 && c.data.mes>0 && c.data.ano>0){
        for(int i=0;i<12;i++){
            if(c.data.mes==array[i][0]){
                if(c.data.dia<=array[i][1] && c.data.dia>0){
                    flag=1;
                    break;
                }else{
                    flag=0;
                    break;
                }
            }
        }
    }else{
        flag=0;
    }
    while(temp_num!=0){
        temp_num/=10;
        cont++;
    }
    int duracao_servico=tempo_servico(c.servico);
    if(cont==9 && (c.data.hora>=8 && c.data.hora*60+c.data.minuto+duracao_servico<=18*60) && flag==1){
        return 1;
    }else{
        return 0;
    }
}
```

A função eliminar ficheiro lê cada linha do arquivo original usando a função fgets, e a função dados_linha é usada para extrair os dados do cliente da linha lida e armazená-los na estrutura c2.Em seguida, é feita uma comparação entre os campos de data e hora da estrutura c2 e da estrutura c1. Se os campos forem iguais, a linha é ignorada usando a instrução continue, o que significa que a linha não será escrita no arquivo temporário.
Se os campos não forem iguais, a linha é escrita no ficheiro temporário usando a função fprintf. No final alteramos o nome do ficheiro “temp.txt” para o nome do ficheiro pretendido “ficheiro_txt”.

```
void elimina_ficheiro(const char*ficheiro_txt,cliente c1){
    FILE*ficheiro;
    FILE*temp_ficheiro;
    char linha[MAX];
    ficheiro=fopen(ficheiro_txt,"r");
    temp_ficheiro=fopen("temp.txt","w");

    if(ficheiro!=NULL && temp_ficheiro!=NULL){
        fseek(ficheiro, 0, SEEK_SET);
        while(fgets(linha,MAX,ficheiro)!=NULL){
            cliente c2;
            dados_linha(linha,&c2);
            if(c2.data.ano==c1.data.ano && c2.data.mes==c1.data.mes && c2.data.dia==c1.data.dia &&
            c2.data.hora==c1.data.hora && c2.data.minuto==c1.data.minuto){
                continue;
            }else{
                fprintf(temp_ficheiro,"%s|%d %02d/%02d/%02d %02d:%02d %s\n",c2.nome,c2.numtel,c2.data.dia,
                c2.data.mes,c2.data.ano,c2.data.hora,c2.data.minuto,c2.servico);
            }
        }
    }
    fclose(ficheiro);
    fclose(temp_ficheiro);
    remove(ficheiro_txt);
    rename("temp.txt",ficheiro_txt);
}
```

A função imprimir_cliente procura o nome do cliente que queremos na lista e mostra todas as reservas ou prereservas associadas a esse cliente,consoante a lista recebida.

```
void imprimir_cliente(Lista lista,cliente c){
    Lista ptr=lista->prox;
    while(ptr!=NULL){
        if(strcmp(ptr->cliente.nome,c.nome)==0){
            printf("%s %d %02d/%02d/%02d %02d:%02d %s\n",ptr->cliente.nome,ptr->cliente.numtel,ptr->cliente.data.dia,
            ptr->cliente.data.mes,ptr->cliente.data.ano,ptr->cliente.data.hora,ptr->cliente.data.minuto,ptr->cliente.servico);
            ptr=ptr->prox;
        }
    }
}
```

As duas funções, menu_cliente e menu_trabalhador, são responsáveis por exibir um menu de opções para o usuário e retornar a opção selecionada.

```
int menu_cliente(void){
    int res;
    printf("O que deseja fazer?\n");
    printf("1-Ver as minhas reservas.\n");
    printf("2-Ver as minhas pre-reservas");
    scanf("%d",&res);
    getchar();
    return res;
}
```

```
int menu_trabalhador(void){
    int res;
    printf("O que deseja fazer?\n");
    printf("1-Eliminar cliente das reservas\n");
    printf("2-Eliminar cliente das pre-reservas\n");
    printf("3-Mostrar os clientes listados nas reservas\n");
    printf("4-Mostrar os clientes listados nas pre-reservas\n");
    printf("5-Executar tarefa!\n");
    scanf("%d",&res);
    getchar();/*Elimina o ultimo carater do buffer de entrada*/
    return res;
}
```

A função apagar_dados_aplicação percorre as listas de reservas e pré-reservas e apaga os dados, apagando no final também os ficheiros associados.

```
void apagar_dados_aplicacao(Lista reservas, Lista prereservas, char*ficheiro_reservas, char* ficheiro_prereservas){
    Lista ptr=reservas,ptr2=prereservas;
    while (ptr != NULL) {
        Lista prox = ptr->prox;
        free(ptr);
        ptr = prox;
    }
    while (ptr2 != NULL) {
        Lista prox = ptr2->prox;
        free(ptr2);
        ptr2 = prox;
    }
    remove(ficheiro_reservas);
    remove(ficheiro_prereservas);

}
```

A função inicializar_aplicacao é responsável por inicializar a aplicação e executar as restantes funções. A função possui várias partes de código, cada um executando uma tarefa específica. Começa-se por declarar algumas variáveis e abrir os ficheiros necessários. A seguir é feita a leitura dos dados de marcações e é criado as listas e os ficheiros das reservas e prereservas .
Após isso, introduz-se toda a parte de interação entre a aplicação, o cliente e o trabalhador. função solicita ao usuário que escolha entre ser um trabalhador da aplicação ou um cliente, e solicita a senha do trabalhador ou o nome do cliente, respectivamente.

```c
while(flag==0){
    printf("1 - Trabalhador da aplicacao.\n");
    printf("2 - Cliente.\n");
    scanf("%d",&flag1);
    getchar();
    if(flag1==1){
        printf("Insira a palavra-passe!\n");
        scanf("%s",str);
        getchar();
        if(strcmp(str,"trabalhar123")==0){
            flag=1;
            flag2=1;
        }else{
            printf("Palavra-passe errada, tente novamente!\n");
        }
    }else if(flag1==2){
        printf("Insira o seu nome:\n");
        scanf("%s",cli.nome);
        getchar();
        flag=1;
        flag4=1;
    }else{
        printf("Opcao invalida!\n");
    }
}
```

Se o usuário selecionar a opção de cliente, a função entra em um menu para o cliente que permite escolher entre ver as suas reservas ou as suas pré-reservas.

```c
while(flag4==1){
    switch(menu_cliente()){
        case 1:
            imprimir_cliente(reservas,cli);
        break;
        case 2:
            imprimir_cliente(prereservas,cli);
        break;
        default:
            printf("Opcao invalida!\n");
        break;
    }
    printf("Pretende continuar a operar?\n");
    printf("| 1-Sim | Outra-Nao |");
    scanf("%d",&flag4);
    getchar();
}
```

Se o usuário selecionar a opção de trabalhador, ele necessita de uma palavra-passe para aceder à aplicação por completo. A função exibe um menu para o trabalhador onde pode escolher entre várias operações, como eliminar um cliente das reservas, eliminar um cliente das pré-reservas, mostrar os clientes listados nas reservas, mostrar os clientes listados nas pré-reservas e executar uma tarefa/serviço.

```
while(flag2==1){
    switch(menu_trabalhador()){
        char data_str[MAX];
        cliente c;
        int res2;
        case 1:
            if(reservas->prox!=NULL){
                printf("Insira o nome do cliente:\n");
                scanf("%s",c.nome);
                getchar();
                printf("Insira a data da reserva deste cliente\n");
                fgets(data_str,MAX,stdin);
                sscanf(data_str,"%d/%d/%d %d:%d %s",&c.data.dia,&c.data.mes,&c.data.ano,&c.data.hora,&c.data.minuto,c.servico);
                elimina_cliente(reservas,c);
                printf("Foi desmarcada com sucesso a %s de %s em %d/%02d/%d %02d:%02d!\n",c.servico,
                c.nome,c.data.dia,c.data.mes,c.data.ano,c.data.hora,c.data.minuto);
                elimina_ficheiro("reservas.txt",c);
                Lista pre_reservas=prereservas->prox;
                printf("Sera verificado se existe alguem das pre-reservas que possa ser colocado nas reservas!\n");
                /*Caso nao seja possivel colocar o cliente das pre reservas, este voltara novamente a ser colocado nas pre-reservas*/
                while(pre_reservas!=NULL){
                    elimina_ficheiro("pre-reservas.txt", pre_reservas->cliente);
                    marcar(reservas, prereservas, ficheiro_reservas, ficheiro_prereservas, pre_reservas->cliente);
                    elimina_cliente(prereservas,c);
                    pre_reservas=pre_reservas->prox;
                }
            }else{
                printf("Nao existem reservas!\n");
            }
        break;
        case 2:
            if(prereservas->prox!=NULL){
                printf("Insira o nome do cliente:\n");
                scanf("%s",c.nome);
                getchar();
                printf("Insira a data da pre-reserva deste cliente\n");
                fgets(data_str,MAX,stdin);
                sscanf(data_str,"%d/%d/%d %d:%d",&c.data.dia,&c.data.mes,&c.data.ano,&c.data.hora,&c.data.minuto);
                elimina_cliente(prereservas,c);
                printf("Foi desmarcada com sucesso a pre-reserva de %s em %d/%d/%d %d:%d!\n",c.nome,c.data.dia,
                c.data.mes,c.data.ano,c.data.hora,c.data.minuto);
                elimina_ficheiro("reservas.txt",c);
            }else{
                printf("Nao existem pre-reservas!\n");
            }
        break;
        case 3:
            if(reservas->prox!=NULL){
                int i=1;
                while(i){
                    printf("1-Imprimir a lista das reservas por ordem crescente da data.\n");
                    printf("2-Imprimir a lista das reservas por ordem decrescente da data.\n");
                    scanf("%d",&res2);
                    getchar();
                    if(res2==1){
                        imprimir_crescente(reservas);
                        i=0;
                    }else if(res2==2){
                        imprimir_decrescente(reservas);
                        i=0;
                    }else{
                        printf("Opcao invalida!\n");
                    }
                }
            }else{
                printf("Nao existem reservas!\n");
```

```
case 4:
    if(prereservas->prox!=NULL){
        int j=1;
        while(j){
            printf("1-Imprimir a lista das pre-reservas por ordem crescente da data.\n");
            printf("2-Imprimir a lista das pre-reservas por ordem decrescente da data.\n");
            scanf("%d",&res2);
            getchar();
            if(res2==1){
                imprimir_crescente(prereservas);
                j=0;
            }else if(res2==2){
                imprimir_decrescente(prereservas);
                j=0;
            }else{
                printf("Opcao invalida!\n");
            }
        }
    }else{
        printf("Nao existem pre-reservas!\n");
    }
break;
```

```
case 5:
    if(reservas->prox!=NULL){
        strcpy(c.nome,reservas->prox->cliente.nome);
        c.data.ano=reservas->prox->cliente.data.ano;
        c.data.mes=reservas->prox->cliente.data.mes;
        c.data.dia=reservas->prox->cliente.data.dia;
        c.data.hora=reservas->prox->cliente.data.hora;
        c.data.minuto=reservas->prox->cliente.data.minuto;
        elimina_cliente(reservas,c);
        printf("Foi executada com sucesso, a reserva de %s em %02d/%02d/%02d %02d:%02d\n",c.nome,
        c.data.ano,c.data.mes,c.data.ano,c.data.hora,c.data.minuto);
        elimina_ficheiro("reservas.txt",c);
    }else{
        printf("Nao existe reservas para executar!\n");
    }
break;
default:
    printf("Opcao invalida!\n");
break;
```

No final de cada operação é perguntado ao utilizador se quer fazer mais operações, caso a resposta seja afirmatica, continuamos dentro do loop, voltando assim aos menus e consequentemente às opções, se não for afirmativa, a função pergunta ao usuário se deseja apagar todos os dados da aplicação. Se a resposta for afirmativa, a função chama a função apagar_dados_aplicacao para apagar as listas e os ficheiros usados.Esta função é chamada no main().

```
printf("Deseja apagar todos os dados da aplicacao?\n");
printf("| 1-Sim | Outra-Nao |");
scanf("%d",&limpar);
getchar();
if(limpar==1){
    apagar_dados_aplicacao(reservas,prereservas,"reservas.txt","pre-reservas.txt");
    printf("Aplicacao formatada com sucesso!\n");
}
```