Manual do Programador
Mini-Projeto PPP

Autores:

Alexandre Ferreira 2021236702

João Tinoco 2021223708

Introdução

A aplicação a ser desenvolvida tem como objetivo auxiliar um funcionário de uma oficina de forma interativa na gestão das reservas de lavagens e manutenções de veículos. Através dessa aplicação, será possível gerenciar as informações dos clientes, disponibilidade de horários para reservas, bem como as reservas em espera.

Entre as principais operações disponíveis na aplicação, destacam-se a reserva de lavagens e manutenções. Além disso, os clientes também terão a opção de realizar uma pré-reserva, caso não haja horário disponível imediatamente, colocando-se em uma lista de espera para ter prioridade caso alguma reserva seja cancelada.

A aplicação também contempla a funcionalidade de cancelar uma reserva, o que permite dispensar o horário para outros clientes. Caso haja uma pré-reserva compatível com o horário desocupado, ela será automaticamente convertida em uma reserva.

Para facilitar a visualização e organização das reservas, a aplicação oferece recursos de listagem, permitindo que as reservas e pré-reservas sejam exibidas em ordem cronológica, tanto de forma geral quanto para um cliente específico.

Por fim, a aplicação possibilita a realização de uma lavagem ou manutenção, removendo a operação da lista de reservas.

É importante ressaltar que as vagas de horários de trabalho vão das 8h00 às 18h00, sem intervalo de almoço. Além disso, é estabelecido que uma lavagem de veículo tem duração de meia hora, enquanto uma manutenção tem duração de uma hora.

Estrutura geral do programa

Para o correto funcionamento e sucesso no objetivo do programa, este utiliza estruturas de dados adequados com diferentes propósitos, localizados no header file “structs.h”. Estes permitem o armazenamento e acesso aos dados necessários para as operações solicitadas pelo funcionário da oficina. Como estrutura principal é utilizada a lista ligada, que por sua vez, são criadas duas neste programa. A lista “lista_reservas” é responsável por armazenar todas as reservas criadas, como também a lista “lista_pre_reservas”, é responsável por armazenar todas as pré-reservas existentes.

No ficheiro “oficina.c” são definidas todas as funções necessárias para a execução do programa, desde as funções responsáveis por alterar as listas como também as funções que manipulam o ficheiro “Reservas.txt”.

Para que fosse possível criar um menu interativo, na função principal do programa “main”, existe um ciclo do-while com um switch case, dando a possibilidade ao cliente de verificar a lista de operações disponíveis, escolher uma das disponíveis e serem executadas todas as tarefas associadas à mesma. Caso o utilizador queira voltar para o menu sempre que está a criar ou cancelar uma reserva, pode digitar “QUIT”.

Principais estruturas de dados

Estrutura **Data**:

Descrição: Armazena a data e hora de uma reserva.

```
typedef struct {
    int dia,mes,ano,hora_ini,min_ini,hora_fim,min_fim;
} Data;
```

Campos:

- dia: inteiro que armazena o dia.
- mes: inteiro que armazena o mês.
- ano: inteiro que armazena o ano.
- hora_ini: inteiro que armazena a hora de início.
- min_ini: inteiro que armazena o minuto de início.
- hora_fim: inteiro que armazena a hora de término.
- min_fim: inteiro que armazena o minuto de término.

Estrutura **Cliente**:

Descrição: Representa um cliente da oficina.

```
typedef struct {
    int numero;
    char nome [50];
} Cliente;
```

Campos:

- numero: inteiro que armazena o número do cliente.
- nome: string que armazena o nome do cliente

Estrutura **Reserva**:

Descrição: Representa uma reserva de lavagem ou manutenção.

```
typedef struct{
    Cliente cliente;
    Data data;
    char tipo[20];
}Reserva;
```

Campos:

- cliente: estrutura do tipo Cliente que armazena as informações do cliente.
- data: estrutura do tipo Data que armazena as informações da data e hora da reserva.
- tipo: string que armazena o tipo de reserva (“lavagem” ou “manutenção”)

Estrutura **noLista**:

Descrição: Representa um nó de uma lista ligada de reservas.

```
typedef struct noLista {
    Reserva reserva;
    struct noLista * ant;
    struct noLista * prox;
} noLista;
```

Campos:

- reserva: estrutura do tipo Reserva que armazena as informações da reserva.
- ant: ponteiro para o nó anterior da lista.
- prox: ponteiro para o próximo nó da lista.

Estrutura **typeLista**:

Descrição: Representa uma lista ligada de reservas.

Campos:

- inicio: ponteiro para o primeiro nó da lista.
- fim: ponteiro para o último nó da lista.

```c
typedef struct{
    noLista *inicio;
    noLista *fim;
}typeLista;
```

Principais funções usadas

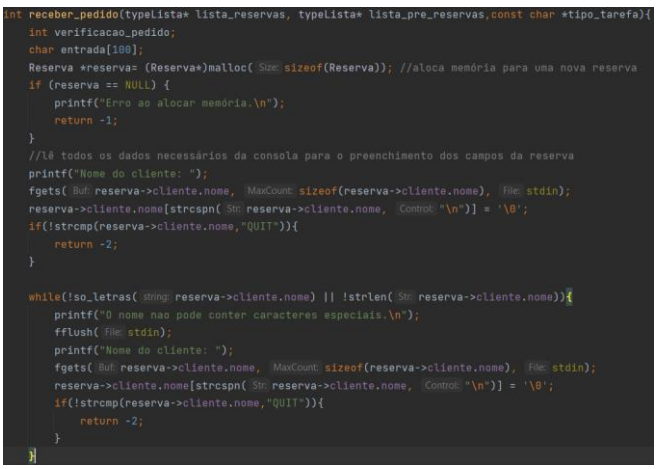
Função **"compara_data"**: Recebe como parâmetros duas estruturas Data "d1" e "d2" e verifica a sua ordem cronológica. Se "d1" for mais antiga que "d2" a função retorna -1, caso seja mais recente esta retorna 1. Se forem iguais é retornado 0.

```c
int compara_data(Data d1, Data d2) {
    // Comparação do ano
    if (d1.ano < d2.ano) return -1;    // Se o ano de d1 for menor que o ano de d2, retorna -1
    else if (d1.ano > d2.ano) return 1;// Se o ano de d1 for maior que o ano de d2, retorna 1
    // Comparação do mês
    if (d1.mes < d2.mes) return -1;    // Se o mês de d1 for menor que o mês de d2, retorna -1
    else if (d1.mes > d2.mes) return 1;// Se o mês de d1 for maior que o mês de d2, retorna 1
    // Comparação do dia
    if (d1.dia < d2.dia) return -1;    // Se o dia de d1 for menor que o dia de d2, retorna -1
    else if (d1.dia > d2.dia) return 1;// Se o dia de d1 for maior que o dia de d2, retorna 1
    // Comparação da hora inicial
    if (d1.hora_ini < d2.hora_ini) return -1;    // Se a hora inicial de d1 for menor que a hora inicial de d2, retorna -1
    else if (d1.hora_ini > d2.hora_ini) return 1;// Se a hora inicial de d1 for maior que a hora inicial de d2, retorna 1
    // Comparação do minuto inicial
    if (d1.min_ini < d2.min_ini) return -1;    // Se o minuto inicial de d1 for menor que o minuto inicial de d2, retorna -1
    else if (d1.min_ini > d2.min_ini) return 1;// Se o minuto inicial de d1 for maior que o minuto inicial de d2, retorna 1
    // Se todas as comparações forem iguais, retorna 0
    return 0;
```

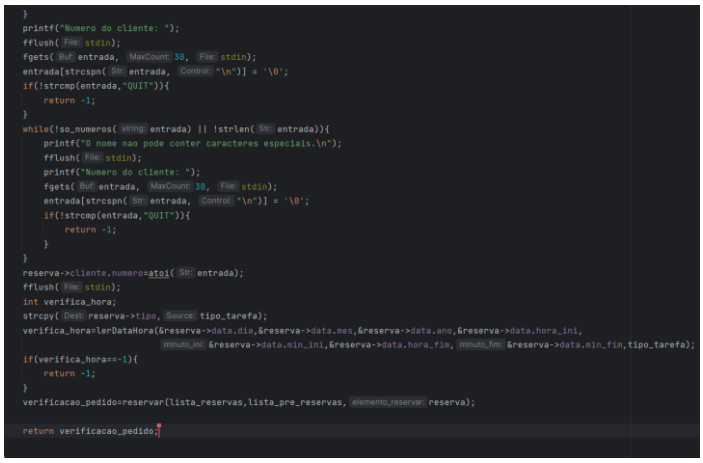
Função **"verifica_disponibilidade"**: Recebe como parâmetros duas estruturas Data "d1" e "d2" e verifica se os seus horários criam conflitos, ou seja, se uma delas está sobreposta à outra. Se as datas não coincidirem, ou se as datas coincidireme e o horário inicial de "d2" for posterior ao horário final de "d1" ou o horário de final de "d2" for anterior ao horário inicial de "d1", é retornado true pois não existe conflito. Caso estas condições não ocorram, então existe conflito e é retornado false.

```c
bool verifica_disponibilidade(Data d1, Data d2) {
    if (d1.ano != d2.ano || d1.mes != d2.mes || d1.dia != d2.dia) { //Verifica se as datas são diferentes
        return true; //nao há conflito
    }
    else if (d1.ano == d2.ano && d1.mes == d2.mes && d1.dia == d2.dia) { //Verifica se as datas são iguais
        // Verifica se o horário inicial d2 é posterior ao horário final de d1 ou verifica se o horário final de d2 é anterior ao horário inicial de d1
        if ((d2.hora_ini > d1.hora_fim || (d2.hora_ini == d1.hora_fim && d2.min_ini >= d1.min_fim)) ||
            (d2.hora_fim < d1.hora_ini || (d2.hora_fim == d1.hora_ini && d2.min_fim <= d1.min_ini))) {
            return true; //nao há conflito
        }
    }
    return false;//há conflito
}
```

Função “**receber_pedido**”: Recebe um ponteiro para a lista de reservas e um ponteiro para a lista de pré-reservas e um string “tipo_tarefa”. Tem como funcionalidade tratar do pedido quando o funcionário deseja reservar uma lavagem ou manutenção. É lida da consola todos os dados necessários para o preenchimento dos campos da uma nova reserva. Por fim, é utilizada a função “reservar” que está encarregue de inserir na lista correta, a nova reserva criada.

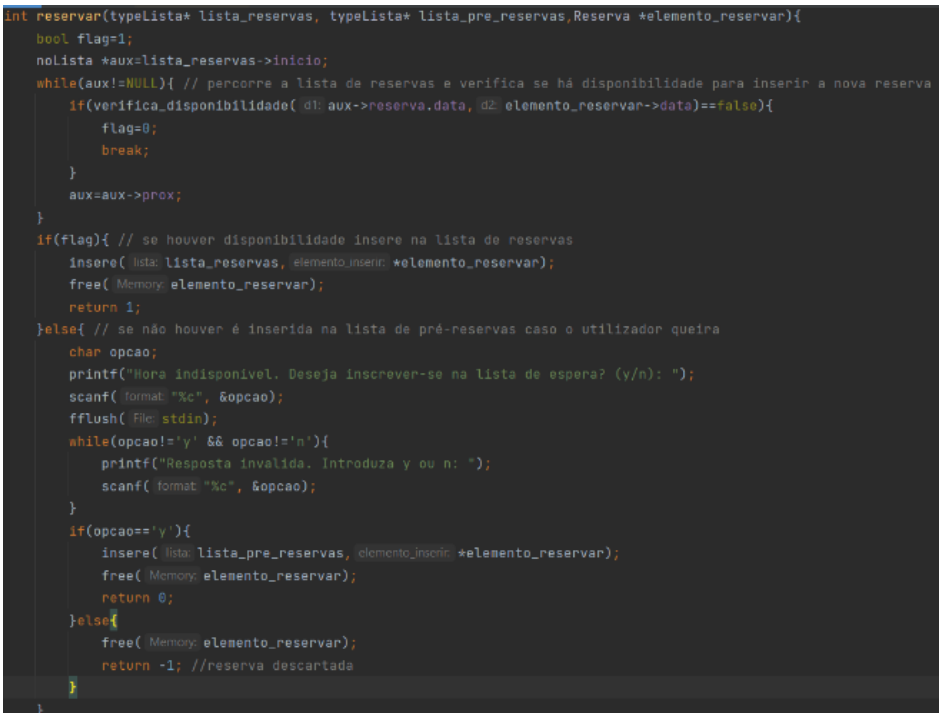


```
int receber_pedido(typeLista* lista_reservas, typeLista* lista_pre_reservas,const char *tipo_tarefa){
    int verificacao_pedido;
    char entrada[100];
    Reserva *reserva= (Reserva*)malloc( Size: sizeof(Reserva)); //aloca memória para uma nova reserva
    if (reserva == NULL) {
        printf("Erro ao alocar memória.\n");
        return -1;
    }
    //lê todos os dados necessários da consola para o preenchimento dos campos da reserva
    printf("Nome do cliente: ");
    fgets( Buf: reserva->cliente.nome,  MaxCount: sizeof(reserva->cliente.nome),  File: stdin);
    reserva->cliente.nome[strcspn( Str: reserva->cliente.nome,  Control: "\n")] = '\0';
    if(!strcmp(reserva->cliente.nome,"QUIT")){
        return -2;
    }

    while(!so_letras( string: reserva->cliente.nome) || !strlen( Str: reserva->cliente.nome)){
        printf("O nome nao pode conter caracteres especiais.\n");
        fflush( File: stdin);
        printf("Nome do cliente: ");
        fgets( Buf: reserva->cliente.nome,  MaxCount: sizeof(reserva->cliente.nome),  File: stdin);
        reserva->cliente.nome[strcspn( Str: reserva->cliente.nome,  Control: "\n")] = '\0';
        if(!strcmp(reserva->cliente.nome,"QUIT")){
            return -2;
        }
    }
```



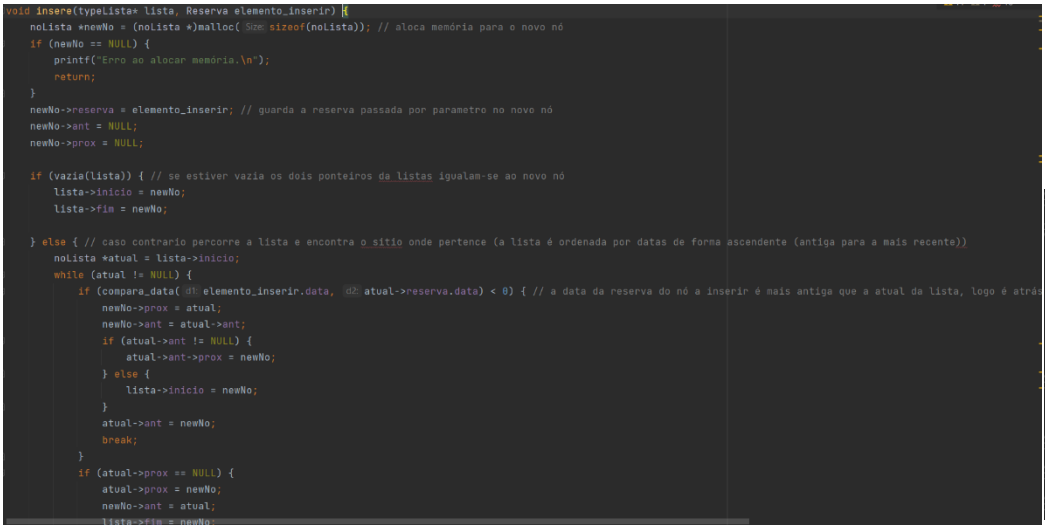
```
    }
    printf("Numero do cliente: ");
    fflush( File: stdin);
    fgets( Buf: entrada,  MaxCount: 30,  File: stdin);
    entrada[strcspn( Str: entrada,  Control: "\n")] = '\0';
    if(!strcmp(entrada,"QUIT")){
        return -1;
    }
    while(!so_numeros( string: entrada) || !strlen( Str: entrada)){
        printf("O nome nao pode conter caracteres especiais.\n");
        fflush( File: stdin);
        printf("Numero do cliente: ");
        fgets( Buf: entrada,  MaxCount: 30,  File: stdin);
        entrada[strcspn( Str: entrada,  Control: "\n")] = '\0';
        if(!strcmp(entrada,"QUIT")){
            return -1;
        }
    }
    reserva->cliente.numero=atoi( Str: entrada);
    fflush( File: stdin);
    int verifica_hora;
    strcpy( Dest: reserva->tipo,  Source: tipo_tarefa);
    verifica_hora=lerDataHora(&reserva->data.dia,&reserva->data.mes,&reserva->data.ano,&reserva->data.hora_ini,
                              minuto_ini: &reserva->data.min_ini,&reserva->data.hora_fim,  minuto_fim: &reserva->data.min_fim,tipo_tarefa);
    if(verifica_hora==-1){
        return -1;
    }
    verificacao_pedido=reservar(lista_reservas,lista_pre_reservas,  elemento_reservar: reserva);

    return verificacao_pedido;
```

Função “**reservar**”: Recebe como parâmetros um ponteiro para a lista de reservas e para a lista de pré-reservas e um ponteiro para uma instância do tipo Reserva. Percorre a lista de reservas e verifica se há disponibilidade para inserir a nova reserva. Se for possível esta é inserida na lista de reservas e a função retorna 1, caso contrário é inserida na lista de pré-reservas caso o utilizador esteja interessado. Se for inserida na lista de pré-reservas a função retorna 0, se não, retorna -1 e a reserva é descartada. As inserções na lista utilizam a função “insere”.



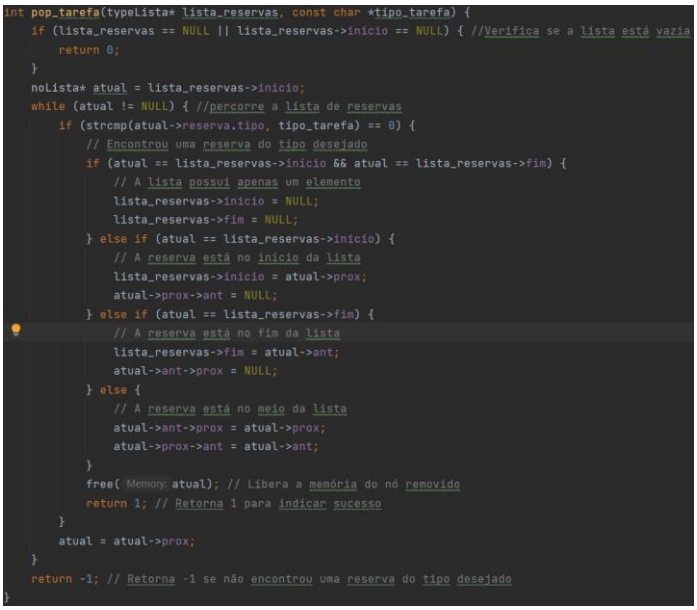
```
int reservar(typeLista* lista_reservas, typeLista* lista_pre_reservas,Reserva *elemento_reservar){
    bool flag=1;
    noLista *aux=lista_reservas->inicio;
    while(aux!=NULL){ // percorre a lista de reservas e verifica se há disponibilidade para inserir a nova reserva
        if(verifica_disponibilidade( d1: aux->reserva.data,  d2: elemento_reservar->data)==false){
            flag=0;
            break;
        }
        aux=aux->prox;
    }
    if(flag){ // se houver disponibilidade insere na lista de reservas
        insere( lista: lista_reservas,  elemento_inserir: *elemento_reservar);
        free( Memory: elemento_reservar);
        return 1;
    }else{ // se não houver é inserida na lista de pré-reservas caso o utilizador queira
        char opcao;
        printf("Hora indisponivel. Deseja inscrever-se na lista de espera? (y/n): ");
        scanf( format: "%c", &opcao);
        fflush( File: stdin);
        while(opcao!='y' && opcao!='n'){
            printf("Resposta invalida. Introduza y ou n: ");
            scanf( format: "%c", &opcao);
        }
        if(opcao=='y'){
            insere( lista: lista_pre_reservas,  elemento_inserir: *elemento_reservar);
            free( Memory: elemento_reservar);
            return 0;
        }else{
            free( Memory: elemento_reservar);
            return -1; //reserva descartada
        }
    }
```

Função **“insere”**: Recebe um ponteiro para uma lista e uma variável do tipo de estrutura Reserva como parâmetros. Aloca nova memória para um novo nó, usando a estrutura noLista, guarda a reserva passada como parâmetro no campo reserva do novo nó e insere o nó na lista. Se a lista estiver vazia o nó é inserido no inicio, caso contrário percorre-se a lista e através da função “compara_data”, encontra-se o sítio onde se deve inserir o nó.



```
void insere(typeLista* lista, Reserva elemento_inserir) {
    noLista *newNo = (noLista *)malloc( Size: sizeof(noLista)); // aloca memória para o novo nó
    if (newNo == NULL) {
        printf("Erro ao alocar memória.\n");
        return;
    }
    newNo->reserva = elemento_inserir; // guarda a reserva passada por parametro no novo nó
    newNo->ant = NULL;
    newNo->prox = NULL;

    if (vazia(lista)) { // se estiver vazia os dois ponteiros da listas igualam-se ao novo nó
        lista->inicio = newNo;
        lista->fim = newNo;

    } else { // caso contrario percorre a lista e encontra o sitio onde pertence (a lista é ordenada por datas de forma ascendente (antiga para a mais recente))
        noLista *atual = lista->inicio;
        while (atual != NULL) {
            if (compara_data( d1: elemento_inserir.data, d2: atual->reserva.data) < 0) { // a data da reserva do nó a inserir é mais antiga que a atual da lista, logo é atrás
                newNo->prox = atual;
                newNo->ant = atual->ant;
                if (atual->ant != NULL) {
                    atual->ant->prox = newNo;
                } else {
                    lista->inicio = newNo;
                }
                atual->ant = newNo;
                break;
            }
            if (atual->prox == NULL) {
                atual->prox = newNo;
                newNo->ant = atual;
                lista->fim = newNo;
                break;
            }
            atual = atual->prox;
        }
    }
}
```

Função “**pop_tarefa**”: Recebe um ponteiro para a lista de reservas e um string “tipo_tarefa”. Esta função é utilizada quando o funcionário deseja realizar uma tarefa, e tem como funcionalidade retirar da lista a primeira reserva encontrada que seja do mesmo tipo de tarefa que a tarefa passada por parâmetro. Verifica se a lista está vazia, caso não esteja percorre-a. Se encontrar algum nó em que o tipo de tarefa da reserva seja igual à tarefa pedida, então é eliminada da lista. São criadas várias condições (if’s) para serem tratados todos os casos possíveis. Os casos são: a lista possuir apenas um elemento, a reserva estar no início da lista, no fim ou no meio da lista. É retornado 1 se for encontrado e eliminado um nó com esse tipo de tarefa, -1 caso não seja encontrado nenhum e 0 se a lista estiver vazia.



```
int pop_tarefa(typeLista* lista_reservas, const char *tipo_tarefa) {
    if (lista_reservas == NULL || lista_reservas->inicio == NULL) { //Verifica se a lista está vazia
        return 0;
    }
    noLista* atual = lista_reservas->inicio;
    while (atual != NULL) { //percorre a lista de reservas
        if (strcmp(atual->reserva.tipo, tipo_tarefa) == 0) {
            // Encontrou uma reserva do tipo desejado
            if (atual == lista_reservas->inicio && atual == lista_reservas->fim) {
                // A lista possui apenas um elemento
                lista_reservas->inicio = NULL;
                lista_reservas->fim = NULL;
            } else if (atual == lista_reservas->inicio) {
                // A reserva está no início da lista
                lista_reservas->inicio = atual->prox;
                atual->prox->ant = NULL;
            } else if (atual == lista_reservas->fim) {
                // A reserva está no fim da lista
                lista_reservas->fim = atual->ant;
                atual->ant->prox = NULL;
            } else {
                // A reserva está no meio da lista
                atual->ant->prox = atual->prox;
                atual->prox->ant = atual->ant;
            }
            free( Memory: atual); // Libera a memória do nó removido
            return 1; // Retorna 1 para indicar sucesso
        }
        atual = atual->prox;
    }
    return -1; // Retorna -1 se não encontrou uma reserva do tipo desejado
}
```

Função “**listar**”: Recebe um ponteiro para uma lista e uma variável inteira que se comporta como um sinal. Se esta variável “tipo_lista” for 1 quer dizer que a funçao vai listar a lista de reservas, se for 2 lista a lista de pré-reservas. A listagem apresenta todos os componentes pertencentes à reserva de cada nó.

Se retornar -1 significa que a lista de reservas estava vazia e se retornar -2 significa que a lista de pré-reservas estava vazia.

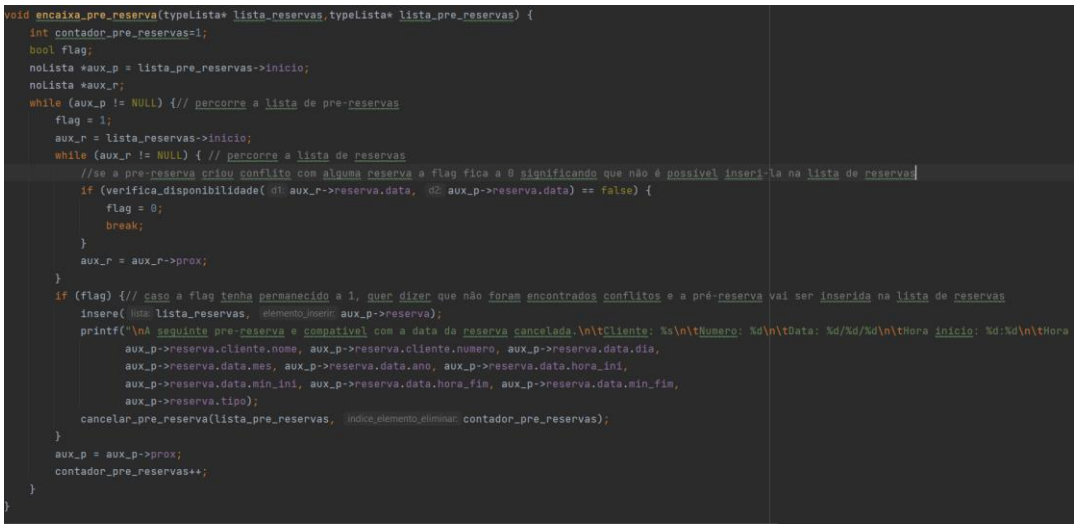
```
int listar(typeLista* lista, const int tipo_lista) {
    int contador=1;
    noLista* aux = lista->inicio;
    if (tipo_lista == 1) {
        if(vazia(lista)){
            return -1;
        }else {
            printf("\nLista de Reservas:\n");
            while (aux != NULL) {
                printf("\t%d.\n",contador);
                printf("\tCliente: %s\n", aux->reserva.cliente.nome);
                printf("\tNumero: %d\n", aux->reserva.cliente.numero);
                printf("\tData: %d/%d/%d\n", aux->reserva.data.dia, aux->reserva.data.mes, aux->reserva.data.ano);
                printf("\tHora inicio: %02d:%02d\n",aux->reserva.data.hora_ini, aux->reserva.data.min_ini);
                printf("\tHora fim: %02d:%02d\n",aux->reserva.data.hora_fim, aux->reserva.data.min_fim);
                printf("\tTipo: %s\n", aux->reserva.tipo);
                printf("\n");
                aux = aux->prox;
                contador++;
            }
        }
    } else if (tipo_lista == 2) {
        if(vazia(lista)){
            return -2;
        }else {
            printf("\nLista de Pre-Reservas:\n");
            while (aux != NULL) {
                printf("\t%d.\n",contador);
                printf("\tCliente: %s\n", aux->reserva.cliente.nome);
                printf("\tNumero: %d\n", aux->reserva.cliente.numero);
                printf("\tData: %d/%d/%d\n", aux->reserva.data.dia, aux->reserva.data.mes, aux->reserva.data.ano);
                printf("\tHora inicio: %02d:%02d\n",aux->reserva.data.hora_ini, aux->reserva.data.min_ini);
                printf("\tHora fim: %02d:%02d\n",aux->reserva.data.hora_fim, aux->reserva.data.min_fim);
                printf("\tTipo: %s\n", aux->reserva.tipo);
                printf("\n");
                aux = aux->prox;
                contador++;
            }
        }
    }
    return 1;
}
```

Função “**listar_cliente**”: Recebe como parâmetro um ponteiro para uma lista e uma string correspondente ao nome do cliente pelo qual se vai listar todas as tarefas associadas, ordenadas por data (da mais recente para a mais antiga). Usa o ponteiro “fim” da lista e o ponteiro “ant” de cada nó, para percorrer a lista no sentido contrário, já que a lista se encontra ordenada por datas (da antiga para a mais recente). Sintaxe semelhante à função “listar”, no entanto contém as diferenças mencionadas anteriormente. Se retornar -1 significa que a lista de reservas estava vazia e se retornar -2 significa que a lista de pré-reservas estava vazia.

Função “**cancelar_pre_reserva**”: Recebe um ponteiro para a lista de pré-reservas e uma variável inteira “indice_elemento_eliminar” que contém o índice da pré-reserva que o funcionário quer cancelar. A lógica da função assemelha-se à função “pop_tarefa”, no entanto a diferença é que esta procura na lista de pre-reservas, o nó correspondente ao qual o funcionário pretende cancelar através da condição (if(contador = =indice_elemento_eliminar). O contador começa em 1 e incrementa quando um nó da lista é percorrido. É retornado 1 se for encontrado e eliminado o nó pretendido, -1 caso não seja encontrado e 0 se a lista estiver vazia.

Função “**cancelar_reserva**”: Semelhante à função “cancelar_pre_reserva”, no entanto, como esta função é responsável por cancelar uma reserva, ou seja, eliminar uma reserva da lista de reservas, e consequentemente, caso haja uma pré-reserva compatível com o horário e dia desocupado, esta ser imediatamente enquadrada, é necessário recorrer à função “encaixa_pre_reserva”. Assim que o nó que se pretenda cancelar é eliminado da lista, indica que criou uma vaga para encaixar uma ou mais pré_reservas.

Função “**encaixa_pre_reserva**”: Recebe um ponteiro para a lista de pré-reservas e um ponteiro para a lista de reservas. Através de dois ciclos “while” percorre cada lista, e verifica se existe disponibilidade para encaixar qualquer pre-reserva na lista de reservas, uma vez que uma das reservas acabou de ser eliminada e abriu vagas. As pre-reservas que foram inseridas na lista de reservas também são mostradas na consola.



```
void encaixa_pre_reserva(typeLista* lista_reservas,typeLista* lista_pre_reservas) {
    int contador_pre_reservas=1;
    bool flag;
    noLista *aux_p = lista_pre_reservas->inicio;
    noLista *aux_r;
    while (aux_p != NULL) {// percorre a lista de pre-reservas
        flag = 1;
        aux_r = lista_reservas->inicio;
        while (aux_r != NULL) { // percorre a lista de reservas
            //se a pre-reserva criou conflito com alguma reserva a flag fica a 0 significando que não é possivel inseri-la na lista de reservas
            if (verifica_disponibilidade( d1: aux_r->reserva.data,  d2: aux_p->reserva.data) == false) {
                flag = 0;
                break;
            }
            aux_r = aux_r->prox;
        }
        if (flag) {// caso a flag tenha permanecido a 1, quer dizer que não foram encontrados conflitos e a pré-reserva vai ser inserida na lista de reservas
            insere( lista: lista_reservas,  elemento_inserir: aux_p->reserva);
            printf("\nA seguinte pre-reserva e compativel com a data da reserva cancelada.\n\tCliente: %s\n\tNumero: %d\n\tData: %d/%d/%d\n\tHora inicio: %d:%d\n\tHora
                   aux_p->reserva.cliente.nome, aux_p->reserva.cliente.numero, aux_p->reserva.data.dia,
                   aux_p->reserva.data.mes, aux_p->reserva.data.ano, aux_p->reserva.data.hora_ini,
                   aux_p->reserva.data.min_ini, aux_p->reserva.data.hora_fim, aux_p->reserva.data.min_fim,
                   aux_p->reserva.tipo);
            cancelar_pre_reserva(lista_pre_reservas,  indice_elemento_eliminar: contador_pre_reservas);
        }
        aux_p = aux_p->prox;
        contador_pre_reservas++;
    }
}
```

Função **“cria”**: Cria e inicializa uma lista ligada vazia. Ela declara uma variável lista do tipo typeLista, define os ponteiros inicio e fim como NULL e retorna a lista criada.

Função **“vazia”**: Verifica se a lista ligada está vazia. Ela recebe um ponteiro para uma lista como parâmetro e verifica se o ponteiro inicio da lista aponta para NULL. Se a lista estiver vazia, a função retorna 1, caso contrário, retorna 0.

Função **“destroi”**: Recebe um ponteiro para uma lista como parâmetro e percorre todos os nós da lista usando um ciclo while, libertando a memória alocada de cada nó com a função “free”.

Função “**verifica_duplos**”: Verifica se já existe alguma reserva num nó na lista recebida como parâmetro, igual à reserva recebida por parâmetro. Desta forma impossibilita de existir dados duplicados no programa.

Função “**verifica_data_hora**”: Função utilizada na função “read_file” para verificar se as informações da reserva estão todas conforme o esperado.

Função “**lerDataHora**”: Função utilizada pela função “receber_pedido” para serem lidas da consola o horário pretendido para a reserva que se está a criar. Tem as devidas proteções.

Principais ficheiros usados

O único ficheiro utilizado neste programa é o “Reservas.txt”. Este ficheiro é utilizado para armazenar as informações das reservas e pré-reservas existentes nas listas “lista_reservas“ e “lista_pre_reservas”.

O conteúdo é armazenado em linhas separadas em que cada linha contém informações sobre uma reserva ou pré-reserva específica, seguindo um formato particular. Caso seja uma reserva a linha inicia pela letra “r” e tem o seguinte formato: “r#nome do cliente#numero do cliente#dia#mes#ano#hora inicial#minuto inicial#hora final#minuto final#tipo de reserva (lavagem ou manutenção)”. Caso seja uma pré-reserva, o formato é o mesmo, ou seja, os campos relativos a uma Reserva separados pelo caracter “#”, no entanto a linha é iniciada pela letra “p”.

Esses ficheiros são manipulados pelas funções “read_file” e “write_file”. A função “read_file” é responsável por ler os dados do ficheiro e carregá-los nas estruturas de dados lista_reservas e lista_pre_reservas, enquanto a função write_file escreve os dados das listas de reservas e pré-reservas no mesmo ficheiro.

Função “**read_file**”:

Função “**write_file**”:

```c
int write_file(typeLista* lista_reservas, typeLista* lista_pre_reservas, const char* nome_arquivo) {
    FILE* arquivo = fopen(nome_arquivo, "w"); //abre o ficheiro como escrita
    if (arquivo == NULL) {
        printf("Erro ao abrir o ficheiro.\n");
        return -1;
    }
    //percorre a lista de reservas
    noLista* atual = lista_reservas->inicio;
    while (atual != NULL) {
        //escreve cada reserva existente da lista reservas no ficheiro com o formato utilizado no "fprintf"
        Reserva reserva = atual->reserva;
        fprintf( stream: arquivo,  format: "r#%s#%d#%d#%d#%d#%d#%d#%d#%d#%s\n",
                reserva.cliente.nome, reserva.cliente.numero,
                reserva.data.dia, reserva.data.mes, reserva.data.ano,
                reserva.data.hora_ini, reserva.data.min_ini,reserva.data.hora_fim, reserva.data.min_fim,
                reserva.tipo);
        atual = atual->prox;
    }
    //percorre a lista de pre-reservas
    atual = lista_pre_reservas->inicio;
    while (atual != NULL) {
        //escreve cada pre-reserva existente da lista pre-reservas no ficheiro com o formato utilizado no "fprintf"
        Reserva pre_reserva = atual->reserva;
        fprintf( stream: arquivo,  format: "p#%s#%d#%d#%d#%d#%d#%d#%d#%d#%s\n",
                pre_reserva.cliente.nome, pre_reserva.cliente.numero,
                pre_reserva.data.dia, pre_reserva.data.mes, pre_reserva.data.ano,
                pre_reserva.data.hora_ini, pre_reserva.data.min_ini,pre_reserva.data.hora_fim, pre_reserva.data.min_fim,
                pre_reserva.tipo);
        atual = atual->prox;
    }
    fclose( File: arquivo);//fecha o ficheiro
    return 1;
```

Exemplos de conteúdos no ficheiro:

```
r#zezito#1323#26#5#2023#8#0#8#30#lavagem
r#joao#121221#26#5#2023#8#30#9#0#lavagem
r#jota#227282#26#5#2023#9#0#10#0#manutencao
r#manuel#21121144#26#5#2023#10#0#11#0#manutencao
r#joao#212123#26#5#2023#12#15#13#15#manutencao
r#manuel#21121#26#5#2023#15#30#16#0#lavagem
p#loko#23232#26#5#2023#9#0#10#0#manutencao
```

Conclusão

A partir da aplicação criada, com as funcionalidades implementadas, é possível proporcionar ao funcionário da oficina uma ferramenta eficaz para o gerenciamento das reservas com uma vasta quantidade de operações, facilitando o controle dos serviços prestados e garantindo a satisfação dos clientes.

De forma, a entregar uma aplicação integra e estável, todos os inputs do utilizador e informações provenientes do ficheiro “Reservas.txt” são controlados de maneira a impedir o corrompimento do programa.