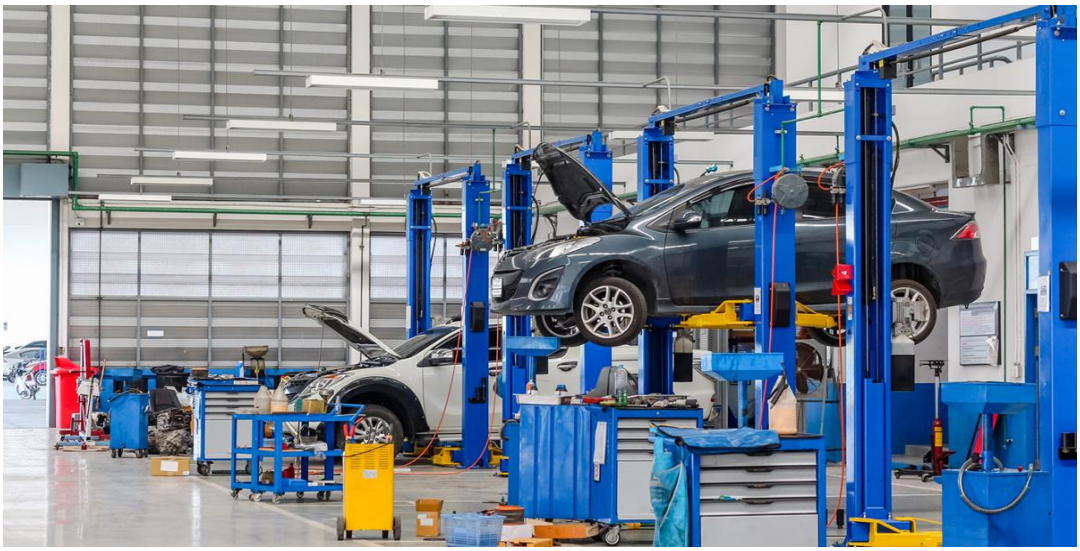
UNIVERSIDADE da MADEIRA

Faculdade de Ciências Exatas e da Engenharia

Segunda Fase do Projeto

Oficina de EDA



Estruturas de Dados e Algoritmos

Grupo: **19**

Professores: Filipe Quintal e Karolina Baras

Projeto realizado por :

- Leonardo Filipe Nóbrega Ferreira Nº: 2122422

- Paulo Alexandre Alves Rodrigues Nº:2120722

- Renato Gabriel Silva Pêssego Nº:2121922

- Francisco Ferreira Afonseca Nº:2120622

INDICE:

1. Introdução

Este projeto, realizado no âmbito da disciplina de Estrutura de Dados e Algoritmos, desenvolvido em C++, tem como objetivo realizar a simulação de uma oficina. Desde o processo da inserção de um carro numa lista de espera, seguido da introdução do carro gerado nas estações de trabalho, que, por sua vez, serão reparados. Sendo possível atender os clientes diretamente através de um menu de gestão. Em suma , colocar todos os termos teóricos e práticos em execução.

É de salientar que é mantido um registo de todos os carros reparados e de todos os carros a serem reparados, no momento, em cada estação de trabalho, o número de dias em que a oficina esteve operacional, quantos dias demorou a reparar esse carro, qual foi o prazo que o mecânico teria de cumprir para o reparar e todas as outras informações específicas de cada estação de trabalho, como faturação total, e o que caracteriza o mecânico, tal como nome, marca e preço de reparação por dia. Para além disso todos os carros mantêm um ID, marca, modelo, tempo de reparação máximo, assim como o seu custo de reparação final.

2. Implementação

2.1. Inicialização das estacoes de trabalho e Oficina

Para inicializar as estações de trabalho, sentimos a necessidade de agrupar as informações numa estrutura. De seguida criamos uma função que retorna uma lista de ETs , em que a mesma inicializa as , com os seus respetivos mecânicos , capacidade, arvore de carros reparados e lista de carros a serem reparados . Em seguida a criação de uma oficina , recorrendo novamente a uma estrutura e a uma função que inicializa a oficina com as suas respetivas características .

Uma oficina é caracterizada por uma lista ligada de ETs , o número de ETs, uma lista de espera, sendo esta uma lista ligada com carros, o tamanho da fila de espera, o número de todos os carros gerados e os ciclos que correspondem aos dias decorridos na Oficina.

De seguida também foi inicializado os mecânicos , com as suas respetivas características , nome , marca especializada e preço por reparação .

```
Oficina CriarOficina(LinhasFicheiro& marcas) {
    Oficina novaof = Oficina();
    novaof.numero_ets = numero_de_et();
    novaof.ets = CriarET(novaof.numero_ets,marcas);

    novaof.carrostotais = 0;
    novaof.ciclos = 0;
    novaof.fila_espera_tamanho = 0;
    novaof.listaespera = new Carro();
    return novaof;
}
Carro* CriarCarrosNaFila(Oficina& Of, LinhasFicheiro& marcas, LinhasFicheiro& modelos, int num)
    Carro* primeirocarro = new Carro();
    while (!MarcaPresente(Of.ets, primeirocarro->marca)) {
        primeirocarro = CriarCarro(marcas, modelos);
        primeirocarro->ID = Of.carrostotais + 1;
    }
    Of.carrostotais = Of.carrostotais + 1;

    Carro* Carroauxiliar = primeirocarro;
    for (int i = 0; i < num-1; i) {
        Carro* novo = new Carro();
        while (!MarcaPresente(Of.ets, novo->marca)) {
            novo = CriarCarro(marcas, modelos);
            novo->ID = Of.carrostotais + 1;
            Carroauxiliar->seguinte = novo;
        }
        Carroauxiliar->seguinte = novo;
        Carroauxiliar = Carroauxiliar->seguinte;
        Of.carrostotais = Of.carrostotais + 1;
        i++;
        }
    Carroauxiliar->seguinte = NULL;
    system("CLS");
    return primeirocarro;
}


bool MarcaPresente(EstacaoTrabalho * primeiraET, string marca) {
    bool temp = false;
    EstacaoTrabalho* atual = primeiraET;
    while (atual != NULL) {
        temp = (marca == atual->mecanico.marca);
        if (temp) {
            break;
        }
        atual = atual->seguinte;
    }

    return temp;

}
```

```
struct EstacaoTrabalho {
    Mecanico mecanico = Mecanico();
    int ID = 0;
    int capacidade = 0;
    int faturacao = 0;
    Carro* carros_a_ser_reparados = new Carro;
    int num_carros_a_ser_reparados = 0;
    Arvore* Carrosreparados;
    int num_carros_reparados = 0;
    EstacaoTrabalho* seguinte;
};
```

```
EstacaoTrabalho * CriarET(int numeroETs,LinhasFicheiro& marcas)
    EstacaoTrabalho* primeiraET = new EstacaoTrabalho;
    primeiraET->mecanico = CriarMecanico(marcas);
    primeiraET->capacidade = 2 + (rand() % 3);
    primeiraET->faturacao = 0;
    primeiraET->ID = 1;
    primeiraET->Carrosreparados = new Arvore;
    primeiraET->carros_a_ser_reparados = new Carro;
    primeiraET->num_carros_a_ser_reparados = 0;
    primeiraET->num_carros_reparados = 0;
    primeiraET->seguinte = NULL;
    EstacaoTrabalho* ETauxiliar = primeiraET;
    for (int i = 1; i < numeroETs; i++) {
        EstacaoTrabalho* novo = new EstacaoTrabalho;
        novo->mecanico = CriarMecanico(marcas);
        novo->capacidade = 2 + (rand() % 3);
        novo->faturacao = 0;
        novo->ID = i + 1;
        novo->Carrosreparados = new Arvore;
        novo->carros_a_ser_reparados = new Carro;
        novo->num_carros_a_ser_reparados = 0;
        novo->num_carros_reparados = 0;
        ETauxiliar->seguinte = novo;
        ETauxiliar = ETauxiliar->seguinte;
        if (i == numeroETs - 1)
            novo->seguinte = NULL;

    }
    return primeiraET;
}
```

2.2. Inicialização dos carros

O código apresentado contém funções que manipulam carros em uma lista ligada. As principais funções são as seguintes:

Nomeadamente a função geririProbabilidades que retorna um *bool*, tendo em conta o valor aleatório.

A função criarCarro que inicializa as características dos carros , nomeadamente o seu modelo , marca, tempo de reparação , dias em reparação , valor que varia conforme os ciclos ,prioridade e custo reparação .

Recorremos também a duas funções auxiliares como a função adicionar e remover carro . A adicionar recebe como argumento a lista ligada de carros onde deseja adicionar e o respetivo carro. Já o remover recebe como argumento a lista ligada de onde deseja remover esse mesmo carro e o seu respetivo ID , as duas foram criadas com o intuito de facilitar a implementação das funções .

A função criar carros na fila tem como intuito criar só e unicamente os carros com as mesmas marcas que os mecânicos das ETs .

Também foi criado uma estrutura para os carros que estão nas arvores na zona dos carros reparados nas ETs. Nomeadamente arvores binarias e as suas funções auxiliares .

```
bool GerarProbalidades(double probalidade) {
    bool probalidade_final = rand() < (probalidade * (RAND_MAX + 1.0));
    return probalidade_final;
}
```

```
void removercarro(Carro* primeirocarro, int ID) {
    Carro* primeirocarro2 = primeirocarro;
    Carro* aux = primeirocarro2;
    Carro* novoprimeirocarro = new Carro;

    if (aux == NULL) {
        return;
    }
    else if (aux != NULL && aux->ID != ID) {
        novoprimeirocarro = aux;
        aux = aux->seguinte;
    }
    else if (aux != NULL && aux->ID == ID) {
        novoprimeirocarro = aux->seguinte;
        aux = aux->seguinte;

        if (aux != NULL) {
            aux = aux->seguinte;

        }
    }
    Carro* final = novoprimeirocarro;
    while (aux != NULL) {
        if (aux->ID == ID) {
            aux = aux->seguinte;
        }
        novoprimeirocarro->seguinte = aux;
        novoprimeirocarro = novoprimeirocarro->seguinte;
        if (aux != NULL) {
            aux = aux->seguinte;
        }
    }
    primeirocarro = final;
}
```

```
Carro* CriarCarro(LinhasFicheiro& marcas, LinhasFicheiro& modelos) {
    Carro* novo_carro = new Carro;
    novo_carro->marca = marcas.linhas[rand() % marcas.tamanho];
    novo_carro->modelo = modelos.linhas[rand() % modelos.tamanho];
    novo_carro->ID = 0;
    novo_carro->tempo_reparacao_max = 2 + (rand() % 3);
    novo_carro->dias_em_reparacao = 0;
    novo_carro->prioritario = GerarProbalidades(0.05);
    novo_carro->custoreparacao = 0;
    novo_carro->seguinte = NULL;
    return novo_carro;
}
```

```
Carro * adicionarcarro(Carro* listaadicionar, Carro* car) {
    Carro* primeirocarro = listaadicionar;
    Carro* aux = primeirocarro;
    Carro* final = aux;
    Carro* novo = car;
    if (aux == NULL) {
        aux = novo;
    }
    else {
        while (aux  -> seguinte != NULL) {
            aux = aux->seguinte;
        }
        aux ->seguinte = novo;
    }
    return final;

}
```

2.3 Funcionamento:

A oficina funciona por simulação de dias, para tal criamos uma função (“seguinte”). Esta função é importante pois permite-nos recorrer o (“MenuInfo”) que nos mostra todas as informações da Oficina. Dentro do (“seguinte”) temos a função (“Menu”) que nos mostra todas as funções de gestão. Seguidamente criamos a função (“organizarprioritario”) que recebe a fila de espera da oficina e percorre-a, transportando os carros prioritários para o início da fila.

Através da função (“ColocarCarrosET”), que recebe um inteiro para sabermos quantos carros necessita de remover da fila de espera. A função ao percorrer a lista de espera se existir algum carro em que exista um mecânico que o possa reparar passará para essa estação de trabalho se houver capacidade para tal. Senão permanecerá na fila de espera. Seguidamente usamos dois menus (“Menu”) e (“MenuInfo”) para apresentar as informações relevantes para simular o dia a dia desta oficina.

O (“Menu”) é onde é nos permitido premir a tecla “s” para continuar ou a letra “g” para aceder o menu de gestão ou ainda a letra “t” para terminar o programa. O programa realiza as mesmas funções caso as letras sejam inseridas em maiúsculas. Usamos uma função dentro do (“Menu”), função (“seguinte”), que ao pressionar a tecla “s” incrementa o número de ciclos, gera novamente dez novos carros na fila e, verifica mais uma vez se foi gerado algum novo carro prioritário e, por sua vez remove até um máximo de oito carros, se possível, dessa mesma fila e distribui-os , pelas respetivas ETs, e, tenta reparar os carros das ETs conforme os critérios dados. Neste caso, um carro tem 15% de chance de ser reparado, caso já esteja em reparação à pelo menos 1 dia e, esteja em reparação a menos dias do que o tempo de reparação máximo do carro.

Caso, seja reparado, este carro é colocado na árvore binaria de carros reparados e, removido da lista ligada de carros a ser reparados associada a cada ET tornando o seu ID = 0, tornando-o um carro inválido, ou seja, que não está a ocupar um lugar nesta lista ligada. Caso não se verifique esta chance, apenas é incrementado em 1 os dias em reparação do carro. No caso, onde se verifique, que os dias em reparação de um carro são iguais ao tempo de reparação máximo do mesmo, o carro é reparado automaticamente, da mesma maneira descrita anteriormente. Sempre que um carro é reparado é calculada a nova faturação da ET, somando à faturação guardada o produto dos dias em reparação do carro pelo preço diário do mecânico. É também calculada e, atribuído o custo de reparação do carro reparado, que será este mesmo produto. Vale lembrar que

como nenhum carro válido pode ter o ID = 0, qualquer carro com ID == 0, é equivalente a um lugar vazio, já que 0 é o valor padrão definido na estrutura do Carro. A chance, nesta função, é feita com o (“gerarprobalidades”), sendo que há 15% de chance para ser reparado. Pela mesma lógica, poderiam ser utilizados quaisquer outros intervalos de 15 valores, ou até mesmo 15 valores separados, entre 1 e 100, para testar a chance de 15%, mas preferimos, escolher o intervalo entre 1 e 15, pois é mais fácil de interpretar.

Novamente, todos estes menus mencionados, foram pensados para lidarem com o máximo de erros possíveis vindos do utilizador, como inserir mais de uma letra quando é pedido apenas uma, lidar com números excessivos para o tipo int, colocar letras em vez de números, para além dos erros relacionados a erros de escrita mencionados anteriormente.

***Figura 3**-Funcao MenuInfo*

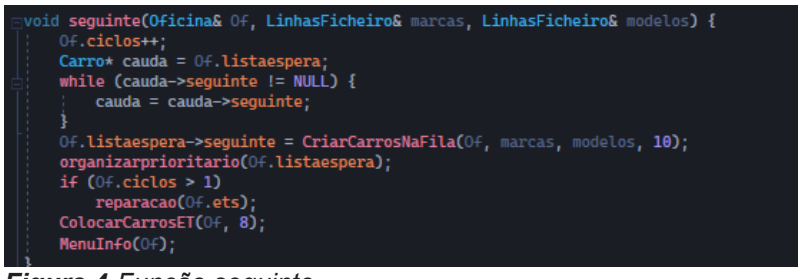


***Figura 4**-Função seguinte*

```
void organizarprioritario(Carro*& listaespera) {
    Carro* primeirocarro = listaespera;
    Carro* percorrerlistaesperal = primeirocarro;
    percorrerlistaesperal = percorrerlistaesperal->seguinte;
    Carro* novolistaespera = new Carro;
    novolistaespera->seguinte = NULL;
    Carro* finalprioritario = novolistaespera;
    Carro* alterarfinal = finalprioritario;

    Carro* novolistaesperanp = new Carro;
    novolistaesperanp->seguinte = NULL;
    Carro* finalnp = novolistaesperanp;
    Carro* alterarfinalnp = finalnp;
    while (percorrerlistaesperal != NULL) {
        if (percorrerlistaesperal->prioritario) {
            alterarfinal->seguinte = percorrerlistaesperal;
            alterarfinal = alterarfinal->seguinte;

        }
        else if (percorrerlistaesperal->prioritario == false) {
            alterarfinalnp->seguinte = percorrerlistaesperal;
            alterarfinalnp = alterarfinalnp->seguinte;



        }
        percorrerlistaesperal = percorrerlistaesperal->seguinte;
    }
    alterarfinal->seguinte = finalnp->seguinte;

    listaespera = finalprioritario;

}
```

Figura 5-*Funcao Organizar prioritário*

```
void criarcarrosaserreparados(EstacaoTrabalho *& et){
    EstacaoTrabalho* primeiraet = et;
    EstacaoTrabalho* percorreret = primeiraet;
    while (percorreret != NULL) {
        Carro* primeirocarro = percorreret->carros_a_ser_reparados;
        Carro* atualcarro = primeirocarro;
        for (int i = 0; i < percorreret->capacidade-1; i++) {
            atualcarro->seguinte = new Carro;
            atualcarro = atualcarro->seguinte;
        }
        percorreret = percorreret->seguinte;
    }

}
```

Figura 6*-Função criarcarrosreparados*

```
void ColocarCarrosET(Oficina& Of, int num) {
    int colocados = 0;
    EstacaoTrabalho* primeiraet = Of.ets;
    EstacaoTrabalho* percorreret = primeiraet;

    while (colocados < num) {
        while (percorreret!=NULL) {
            Carro* primeirocarro = Of.listaespera;
            Carro* percorrerespera = primeirocarro;
            percorrerespera = percorrerespera->seguinte;
            while (percorrerespera != NULL) {
                if ((percorreret->mecanico.marca == percorrerespera->marca) && (colocados < num)) {
                    Carro* primeirocarrroaserreparado = percorreret->carros_a_ser_reparados;
                    Carro* percorreraserreparados = primeirocarrroaserreparado;
                    for (int i = 0; i < percorreret->capacidade; i++) {
                        if (percorreraserreparados->ID == 0) {
                            percorreraserreparados->custoreparacao = percorrerespera->custoreparacao;
                            percorreraserreparados->dias_em_reparacao = percorrerespera->dias_em_reparacao;
                            percorreraserreparados->ID = percorrerespera->ID;
                            percorreraserreparados->marca = percorrerespera->marca;
                            percorreraserreparados->modelo = percorrerespera->modelo;
                            percorreraserreparados->prioritario = percorrerespera->prioritario;
                            percorreraserreparados->seguinte = percorreraserreparados->seguinte;
                            percorreraserreparados->tempo_reparacao_max = percorrerespera->tempo_reparacao_max;
                            removercarro(Of.listaespera, percorrerespera->ID);
                            colocados++;
                            percorreret->num_carros_a_ser_reparados++;

                            break;
                        }
                        percorreraserreparados = percorreraserreparados->seguinte;
                    }

                }
                percorrerespera = percorrerespera->seguinte;
            }
            percorreret = percorreret->seguinte;
        }
        break;
    }
}
```

Figura 7 *-Função ColocarcarrosETS*

```
Arvore* novoNodo(Carro* item){

    Arvore* temp = new Arvore();
    Carro* temp2 = item;
    temp->carros_reparados = temp2;
    temp->esquerda = temp->direita = NULL;
    return temp;
}
void insert(Arvore* nodo, Carro* chave)
{
    Arvore* primeironodo = nodo;
    Arvore* aux = primeironodo;
    Arvore* nova = novoNodo(chave);
    while (aux != NULL) {
        if (chave->modelo > aux->carros_reparados->modelo || chave->modelo == aux->carros_reparados->modelo) {
            if (aux->direita == NULL) {
                aux->direita = nova;
                break;
            }
            else {
                aux = aux->direita;
            }
        }
        else if (chave->modelo < aux->carros_reparados->modelo) {
            if (aux->esquerda == NULL) {
                aux->esquerda = nova;
                break;
            }
            else {
                aux = aux->esquerda;
            }
        }
    }
    nodo = primeironodo;
}

void adicionarar(Arvore* raiz, Carro* chave) {
    if (raiz->carros_reparados->ID == 0) {
        raiz->carros_reparados = chave;
        raiz->direita = NULL;
        raiz->esquerda = NULL;
    }
    else {
        insert(raiz, chave);
    }
}
```

Figura 8- *Criação das arvores , usada apara os carros reparados nas ETs*

3.Gestao

Nesta andamento do trabalho , algumas operacoes efetuadas anteriormente foram precissas e por isso mesmo algumas estao incompletas .

Contudo neste parametro é inicializado um menu com as opcoes “seguinte” e “gestao“ em que o utilizador tem a opcao de passar para o dia seguinte ou entao abrir a seccao da gestao , com as seguintes funcionalidades a escolha .

```
void Menu(Oficina& Of, LinhasFicheiro& marcas, LinhasFicheiro& modelos) {
    string escolha;
    bool sair = false;
    do {
        cout << "Dia (s)eguinte ********** (g)estao" << endl << "(T)erminar programa" << endl;

        cout << "Seleccione a sua opcao : " << endl;
        getline(cin, escolha);
        while (escolha.length() != 1) {
            cout << "Escolha invalida! Digite apenas uma das letras destacadas abaixo." << endl;
            cout << "Dia (s)eguinte ********** (g)estao" << endl << "(T)erminar programa" << endl;

            cout << "Seleccione a sua opcao : " << endl;
            getline(cin, escolha);
            cout << endl;
        }

        switch (escolha[0]) {
        case 's':
        case 'S':
            seguinte(Of, marcas, modelos);
            MenuInfo(Of);
            Menu(Of, marcas, modelos);
            sair = true;
            break;
        case 'g':
        case 'G':
            gestao(Of, marcas, modelos);
            sair = true;
            break;
        case 't':
        case 'T':
            sair = true;
            break;

        default:
            cout << "Opcao invalida" << endl;
            break;

        }
    } while (!sair);
}
```

3.1. Atualizar tempo de reparacao

A funcao atualizar tempo de reparacao recebe como argumentos a oficina , pede ao utilizador o tempo a atualizar e as caracteristicas do carro a atualizar e percorre a lista de espera a verificar se existe algum carro com estas caracteristicas .

```
void atualizar_tempo_reparacao(Oficina& Of) {

    string marca;
    string modelo;
    string entrada;
    double tempotemp = 0;
    int tempo;
    cout << "indique a marca a atualizar o tempo de reparacao: " << endl;
    cin >> ws;
    getline(cin, marca);
    cout << "indique o modelo a atualizar o tempo de reparacao: " << endl;
    cin >> ws;
    getline(cin, modelo);
    cout << "Introduza o respetivo tempo: \n ";
    getline(cin, entrada);
    while ((tempotemp > INT_MAX) || (tempotemp <= 0)) {
        while (!verificarnumero(entrada)) {
            cout << "Tempo invalido!" << endl << "introduza um tempo de reparacao valido (inteiro entre 0 e 2147483647): " << endl;
            getline(cin, entrada);
        }
        tempotemp = stod(entrada);
        if ((tempotemp <= INT_MAX) && (tempotemp > 0)) {
            tempo = (int)tempotemp;

        }
        else {
            cout << "Tempo invalido!" << endl << "introduza um tempo de reparacao valido (inteiro entre 0 e 2147483647): " << endl;
            getline(cin, entrada);
            tempotemp = 0;
        }
    }
    //percorrer lista de espera
    Carro* primeiroespera = Of.listaespera;
    Carro* atual = primeiroespera;
    while (atual != NULL) {
        if (removerespacos((maiuscula(atual->marca))) == removerespacos(maiuscula(marca)) && (removerespacos(maiuscula(atual->modelo)) == removerespacos(maiuscula(modelo)))) {
            atual->tempo_reparacao_max = tempo;

        }
        atual = atual->seguinte;
    }
}
```

3.2. Adicionar prioridade

A funcao adicionar prioridade recebe como argumentos a oficina , pede ao utilizador que insira o id do carro da lista de espera que quer dar prioridade , e de seguida a lista de espera volta a ser organizada .

```
void adicionar_prioridade(Oficina& Of) {
    string entrada;
    double Idtemp = 0;
    int Id;
    cout << "Indique O ID do carro que quer por a Prioritario: " << endl;
    cin >> ws;
    getline(cin, entrada);
    while ((Idtemp > INT_MAX) || (Idtemp <= 0)) {
        while (!verificarnumero(entrada)) {
            cout << "ID invalido!" << endl << "Indique um ID valido para colocar como prioritario (inteiro): " << endl;
            getline(cin, entrada);
        }
        Idtemp = stod(entrada);
        if ((Idtemp <= INT_MAX) && (Idtemp > 0)) {
            Id = (int)Idtemp;

        }
        else {
            cout << "ID invalido!" << endl << "Indique um ID valido para colocar como prioritario (inteiro entre 0 e 2147483647): " << endl;
            getline(cin, entrada);
            Idtemp = 0;
        }
    }
    //percorrer lista de espera2
    Carro* primeiroCarro = Of.listaespera;
    Carro* atual = primeiroCarro;
    while (atual != NULL) {
        //faz se id for igual e se n for prioritario
        if ((atual->ID == Id) && (atual->prioritario == false)) {
            atual->prioritario = true;
        }

        atual = atual->seguinte;
    }

    organizarprioritario(Of.listaespera);
}
```

3.3.Remover mecanico

A funcao remover mecanico recebe como argumento a oficina , pede ao utilizador o nome do mecanico que pretende eliminar , depois são percorridas todas as ets a fim de verificar se existe alguma et com essse mesmo mecanico , se sim o mesmo é eliminado e de seguida um novo mecanico e criado , com assuarespetivas caracteristicas . Todos os carros que estavam na lista de espera dos carros a serem reparados.

```
void remover_mecanico(Oficina& Of, LinhasFicheiro& marcas) {
    string mecanicoRem;
    string mecanicoAdi;
    cout << "Indique o nome do mecanico que deseja remover: " << endl;
    cin >> ws;
    getline(cin, mecanicoRem);
    EstacaoTrabalho* primeiraET = Of.ets;
    EstacaoTrabalho* atualET = primeiraET;
    while (atualET != NULL) {
        if (removerespacos(maiuscula(atualET->mecanico.nome)) == removerespacos(maiuscula(mecanicoRem))) {
            break;
        }
        else {
            atualET = atualET->seguinte;
        }
    }
    if (atualET == NULL) {
        cout << "Nao existe nenhuma ET com esse mecanico!" << endl;
        return;
    }
    //Reparar todos os carros
    Carro* primeirocarro = atualET->carros_a_ser_reparados;
    Carro* percorrerlistacarros = primeirocarro;
    int n = atualET->num_carros_a_ser_reparados;
    for (int i = 0; i < n; i++) {
        if (percorrerlistacarros->ID != 0) {
            percorrerlistacarros->custoreparacao = percorrerlistacarros->dias_em_reparacao * atualET->mecanico.preco_reparacao_por_dia;
            //inserir arvore
            atualET->num_carros_a_ser_reparados = atualET->num_carros_a_ser_reparados - 1;
            percorrerlistacarros->ID = 0;
            atualET->faturacao = atualET->faturacao + percorrerlistacarros->custoreparacao;
            atualET->num_carros_reparados = atualET->num_carros_reparados + 1;
        }
    }
    //adicionar novo mecanico
    cout << "O mecanico com o nome " << mecanicoRem << ", foi removido da et" << endl;
    cout << "ET: " << atualET->ID << " | " << "Mecanico: " << atualET->mecanico.nome
        << " | " << "Capacidade: " << atualET->capacidade << " | " <<
        "Marca: " << atualET->mecanico.marca << " | " << "Total Faturacao: " << atualET->faturacao << endl;
    cout << endl;

    //remover a ET antiga e criar uma nova ou algo semelhante
    EstacaoTrabalho* temp = CriarET(1, marcas);
    atualET->ID = atualET->ID;
    atualET->capacidade = temp->capacidade;
    atualET->Carrosreparados = temp->Carrosreparados;
    atualET->carros_a_ser_reparados = temp->carros_a_ser_reparados;
    atualET->faturacao = temp->faturacao;
    atualET->mecanico = temp->mecanico;
    atualET->num_carros_a_ser_reparados = temp->num_carros_a_ser_reparados;
    atualET->num_carros_reparados = temp->num_carros_reparados;
    Carro* primeirocarroaux = atualET->carros_a_ser_reparados;
    Carro* atualcarro = primeirocarroaux;
    for (int i = 0; i < atualET->capacidade - 1; i++) {
        atualcarro->seguinte = new Carro;
        atualcarro = atualcarro->seguinte;
    }
}
```

3.4.Gravar oficina

A função gravarOficina tem como objetivo gravar os dados do estado atual da "Oficina" em um arquivo de texto. Esses dados são organizados em um formato específico, separados por "|" e "\n", de modo que possam ser lidos e interpretados posteriormente, sendo estas separações utilizadas, para definir interrupções de leitura, posteriormente.

O primeiro passo é definir o caminho do arquivo onde os dados serão gravados. Em seguida, é criado um objeto ofstream para manipular o arquivo. O arquivo é então aberto e é verificado se a abertura foi bem-sucedida.

Se o arquivo foi aberto com sucesso, os dados da "Oficina" são gravados no arquivo, começando com as informações gerais da oficina, como o número de ciclos, o número total de carros e o número de estações de trabalho (ETs). Em seguida, para cada ET, as informações específicas são gravadas, como o ID, a capacidade, a faturação, o número de carros em reparação, o número de carros reparados e os detalhes de cada carro. Em seguida, os carros em espera são listados, seguidos pelas informações dos mecânicos que trabalham nas ETs.

Por fim, a fila de espera de carros é gravada no arquivo, juntamente com as informações de cada carro na fila. Após a conclusão da gravação, o arquivo é fechado. Se ocorrer algum erro durante o processo de gravação, a mensagem "Erro ao abrir o ficheiro" é exibida no ecrã. Sendo gravada, por padrão num arquivo “Oficina.txt”

```
void gravar_oficina(const Oficina& oficina) {
    string caminho = "oficina.txt";
    ofstream ficheiro(caminho);

    if (ficheiro.is_open()) {
        // Gravação dos dados da oficina
        ficheiro << oficina.ciclos << "|"
            << oficina.carrostotais << "|"
            << oficina.numero_ets << "\n";

        // Gravação dos dados de cada EstacaoTrabalho
        EstacaoTrabalho* atualET = oficina.ets;
        while (atualET != nullptr) {
            ficheiro << atualET->ID << "|"
                << atualET->capacidade << "|"
                << atualET->faturacao << "|"
                << atualET->num_carros_a_ser_reparados << "|"
                << atualET->num_carros_reparados << "\n";

            // Gravação dos carros a ser reparados
            Carro* atualcarrosaserreparados = atualET->carros_a_ser_reparados;
            while (atualcarrosaserreparados != nullptr) {
                ficheiro << atualcarrosaserreparados->dias_em_reparacao << "|"
                    << atualcarrosaserreparados->ID << "|"
                    << atualcarrosaserreparados->marca << "|"
                    << atualcarrosaserreparados->modelo << "|"
                    << atualcarrosaserreparados->prioritario << "|"
                    << atualcarrosaserreparados->tempo_reparacao_max << "\n";
                atualcarrosaserreparados = atualcarrosaserreparados->seguinte;
            }

            // Gravação dos dados do mecânico
            ficheiro << atualET->mecanico.marca << "|"
                << atualET->mecanico.nome << "|"
                << atualET->mecanico.preco_reparacao_por_dia << "\n";

            atualET = atualET->seguinte;
        }

        // Gravação dos dados da fila de espera
        ficheiro << oficina.fila_espera_tamanho << "\n";
        Carro* atualFilaEspera = oficina.listaespera;
        while (atualFilaEspera != nullptr) {
            ficheiro << atualFilaEspera->ID << "|"
                << atualFilaEspera->dias_em_reparacao << "|"
                << atualFilaEspera->marca << "|"
                << atualFilaEspera->modelo << "|"
                << atualFilaEspera->prioritario << "|"
                << atualFilaEspera->tempo_reparacao_max << "\n";
            atualFilaEspera = atualFilaEspera->seguinte;
        }

        ficheiro.close();
    }
    else {
        cout << "Erro ao abrir o arquivo" << endl;
    }
}
```

3.5.Carregar oficina

De seguida, há o segmento de código, carregarOficina, em que carrega dados de uma oficina de carros a partir de um arquivo de texto. Ele lê as informações de cada estação de trabalho e dos carros que estão sendo reparados ou que já foram reparados, e armazena essas informações em uma nova “Oficina”.

As informações são lidas do arquivo linha por linha, e são armazenadas em uma nova “Oficina” temporária, que é atualizada conforme o arquivo é lido. No final da leitura do arquivo, a nova instância da “Oficina” é atribuída à instância passada como parâmetro para a função carregar oficina.

A função começa verificando se o arquivo pode ser aberto, e então lê a primeira linha do arquivo, que contém as informações gerais da oficina (número de ciclos, número total de carros e número de estações de trabalho). Em seguida, para cada estação de trabalho, a função lê as informações correspondentes do arquivo e cria uma nova “EstacaoTrabalho”. Isso inclui informações como a capacidade da estação, a faturação, o número de carros que já foram reparados nessa estação, o número de carros que estão sendo reparados atualmente e a lista de carros que foram reparados. Esta função permite-nos carregar uma oficina previamente gravada pelo programa, utilizando o caminho padrão, ou, utilizar um caminho definido pelo utilizador, de forma a carregar uma outra oficina presente nos arquivos do programa.

Para cada carro que foi reparado, a função lê as informações correspondentes do arquivo e cria um novo “Carro”, que é adicionado à lista de carros reparados. Para cada carro que está sendo reparado atualmente, a função lê as informações correspondentes do arquivo e cria um “Carro”, que é adicionado à lista de carros que estão sendo reparados atualmente. Por fim, a função lê as informações do mecânico que está atualmente trabalhando nessa estação de trabalho e cria um novo “Mecanico”.

Ao final da leitura do arquivo, a “Oficina” temporária é atribuída à "Oficina" original, passada como parâmetro para a função “carregar_oficina”.

```
}
void carregar_oficina(Oficina& Of, string& caminho) {
    ifstream ficheiro(caminho);

    if (ficheiro.is_open()) {
        Oficina nova;
        string linha;

        getline(ficheiro, linha);
        stringstream ss(linha);
        string temp;

        // Obtém ciclos
        getline(ss, temp, '|');
        nova.ciclos = stoi(temp);

        // Obtém carrostotais
        getline(ss, temp, '|');
        nova.carrostotais = stoi(temp);

        // Obtém numero_ets
        getline(ss, temp, '|');
        nova.numero_ets = stoi(temp);

        // Cria as Estacoes de Trabalho
        nova.ets = nullptr;
        EstacaoTrabalho* atualET = nullptr;
        for (int i = 0; i < nova.numero_ets; i++) {
```

```
nova.ets = nullptr;
EstacaoTrabalho* atualET = nullptr;
for (int i = 0; i < nova.numero_ets; i++) {
    getline(ficheiro, linha);
    stringstream ss(linha);

    // Obtém ID
    getline(ss, temp, '|');
    int id = stoi(temp);

    // Obtém capacidade
    getline(ss, temp, '|');
    int capacidade = stoi(temp);

    // Obtém faturacao
    getline(ss, temp, '|');
    float faturacao = stof(temp);

    // Obtém num carros a ser reparados
    getline(ss, temp, '|');
    int carros_a_reparar = stoi(temp);

    // Obtém carros reparados
    getline(ss, temp, '|');
    int carros_reparados = stoi(temp);

    EstacaoTrabalho* novaET = new EstacaoTrabalho;
    if (nova.ets == nullptr) {
        nova.ets = novaET;
        atualET = nova.ets;
    }
    else {
        atualET->seguinte = novaET;
        atualET = atualET->seguinte;
    }
}

atualET = nova.ets;
while (atualET != nullptr) {
    getline(ficheiro, linha);
    stringstream ss(linha);

    // Obtém Mecânico
    Mecanico mecanico;

    // Obtém marca
    getline(ss, temp, '|');
    mecanico.marca = temp;

    // Obtém nome do mecânico
    getline(ss, temp, '|');
    mecanico.nome = temp;
```

```
    mecanico.nome = temp;

    // Obtém preço de reparação por dia
    getline(ss, temp, '|');
    mecanico.preco_reparacao_por_dia = stoi(temp);

    atualET->mecanico = mecanico;

    atualET = atualET->seguinte;
}

// Carrega carros a serem reparados
atualET = nova.ets;
while (atualET != nullptr) {
    int capacidade = atualET->capacidade;
    atualET->carros_a_ser_reparados = nullptr;
    Carro* atualcarrosaserreparados = nullptr;
    for (int i = 0; i < capacidade; i++) {
        getline(ficheiro, linha);
        stringstream ss(linha);

        // Obtém dias em reparação
        getline(ss, temp, '|');
        int dias_em_reparacao = stoi(temp);

        // Obtém ID do carro em reparação
        getline(ss, temp, '|');
        int ID = stoi(temp);

        // Obtém marca
        getline(ss, temp, '|');
        string marca = temp;

        // Obtém modelo
        getline(ss, temp, '|');
        string modelo = temp;

        // Obtém prioridade
        getline(ss, temp, '|');
        int prioridade = stoi(temp);

        // Obtém tempo de reparação máximo
        getline(ss, temp, '|');
        int tempo_reparacao_max = stoi(temp);
```

```
            getline(ss, temp, '|');
            int tempo_reparacao_max = stoi(temp);

            Carro* novoCarro = new Carro;
            if (atualET->carros_a_ser_reparados == nullptr) {
                atualET->carros_a_ser_reparados = novoCarro;
                atualcarrosaserreparados = atualET->carros_a_ser_reparados;
            }
            else {
                atualcarrosaserreparados->seguinte = novoCarro;
                atualcarrosaserreparados = atualcarrosaserreparados->seguinte;
            }
        }
        atualET = atualET->seguinte;
    }

    // Carrega carros reparados
    atualET = nova.ets;
    while (atualET != nullptr) {
        int num_carros_reparados = atualET->num_carros_reparados;
        atualET->Carrosreparados = nullptr;
        Arvore* atualcarrosreparados = nullptr;
        for (int i = 0; i < num_carros_reparados; i++) {
            getline(ficheiro, linha);
            stringstream ss(linha);

            // Obtém ID
            getline(ss, temp, '|');
            int ID = stoi(temp);

            // Obtém dias_em_reparacao
            getline(ss, temp, '|');
            int dias_em_reparacao = stoi(temp);

            // Obtém marca
            getline(ss, temp, '|');
            string marca = temp;

            // Obtém modelo
            getline(ss, temp, '|');
            string modelo = temp;

            // Obtém prioridade
            getline(ss, temp, '|');
            int prioridade = stoi(temp);

            // Obtém tempo_reparacao_max
            getline(ss, temp, '|');
            int tempo_reparacao_max = stoi(temp);
```

```
                int tempo_reparacao_max = stoi(temp);

                Arvore* novoCarro = new Arvore;
                if (atualET->Carrosreparados == nullptr) {
                    atualET->Carrosreparados = novoCarro;
                    atualcarrosreparados = atualET->Carrosreparados;
                }
                else {
                    atualcarrosreparados->esquerda = novoCarro;
                    atualcarrosreparados = atualcarrosreparados->esquerda;
                }
            }
            atualET = atualET->seguinte;
        }

        // Carrega fila de espera
        getline(ficheiro, linha);

        getline(ss, temp, '|');
        int fila_espera_tamanho = stoi(temp);

        nova.listaespera = nullptr;
        Carro* atualfilaespera = nullptr;
        for (int i = 0; i < fila_espera_tamanho; i++) {
            getline(ficheiro, linha);
            stringstream ss(linha);

            // Obtém ID
            getline(ss, temp, '|');
            int ID = stoi(temp);

            // Obtém dias_em_reparacao
            getline(ss, temp, '|');
            int dias_em_reparacao = stoi(temp);

            // Obtém marca
            getline(ss, temp, '|');
            string marca = temp;

            // Obtém modelo
            getline(ss, temp, '|');
            string modelo = temp;

            // Obtém prioridade
            getline(ss, temp, '|');
            int prioridade = stoi(temp);

            // Obtém tempo_reparacao_max
            getline(ss, temp, '|');
            int tempo_reparacao_max = stoi(temp);

            Carro* novoCarro = new Carro;
            if (nova.listaespera == nullptr) {
                nova.listaespera = novoCarro;
                atualfilaespera = nova.listaespera;
            }
            else {
                atualfilaespera->seguinte = novoCarro;
                atualfilaespera = atualfilaespera->seguinte;
            }
        }

        of = nova;
        ficheiro.close();
        cout << "Oficina carregada com sucesso!" << endl;
    }
    else {
        cout << "Erro: Não foi possivel abrir o arquivo de carregamento!" << endl;
    }
```

3.3. Adicionar ET

A funcao adicionar ET recebe como argumentos a oficina , as marcas e os modelos , a mesma pede ao utilizador todas as caracteristicas de uma ET , inicializaa e de seguida adiciona as ets já existentes .

```
void adicionar_ET(Oficina& of, LinhasFicheiro& marcas, LinhasFicheiro& modelos) {
    EstacaoTrabalho* primeiraet = of.ets;
    EstacaoTrabalho* percorreret = primeiraet;
    while (percorreret->seguinte != NULL) {
        percorreret = percorreret->seguinte;
    }
    percorreret->seguinte = CriarET(1, marcas);
    percorrer(variável local) EstacaoTrabalho *percorreret
    percorrer Pesquisar Online
    Carro* pr                                          ier_reparados;
    Carro* percorrercarro = primeirocarro;
    for (int i = 0; i < percorreret->capacidade - 1; i++) {
        percorrercarro->seguinte = new Carro;
        percorrercarro = percorrercarro->seguinte;
    }

    of.ets = primeiraet;
    of.numero_ets = of.numero_ets + 1;
}
```

3.4. Imprimir carros reparados

A funcao impimir carros reparados pede ao utilizador que o mesmo indique a forma como o mesmo quer imprimir as carros reparados de uma certa ET com o seu respetivo ID, de seguida é verificado se existe alguma ET com esse ID e são imprimidos os carros reparados atravez da representacao visual de uma arvore binaria .

```
void imprimiroficinaporrerpresentacao(Oficina & of) {
    double IDtemp = 0;
    int ID = 0;
    while (IDtemp > of.numero_ets || IDtemp <= 0) {
        cout << "Insira um ID da ET: " << endl;
        cin >> IDtemp;
    }
    ID = int(IDtemp);
    EstacaoTrabalho* primeiraET = of.ets;
    EstacaoTrabalho* percorrerET = primeiraET;
    while (percorrerET != NULL) {
        if (percorrerET->ID == ID) {
            cout << "Marca: " << endl;
            imprimirarvore(percorrerET->Carrosreparados, numeroniveis(percorrerET->Carrosreparados));
            return;
        }
    }
    cout << "Ocorreu algum erro. Tente novamente!" << endl;
}
```

```
void imprimirarvore(Arvore* no, int nivel) {
    if (no == NULL || no->carros_reparados->ID == 0){
        cout << endl;
        return;
    }
    imprimirarvore(no->direita, nivel + 1);
    for (int i = 1; i < nivel; i++)
    {
        cout << "\t";
    }
    cout << no->carros_reparados->ID << endl;
    imprimirarvore(no->esquerda, nivel + 1);
}
```

```
int numeroniveis(Arvore* raiz){
    int h = 0;
    if (raiz != NULL )
    {
        if (raiz->carros_reparados->ID != 0) {
            int alturaesq = numeroniveis(raiz->esquerda);
            int alturadir = numeroniveis(raiz->direita);
            int alturamax = max(alturaesq, alturadir);
            h = alturamax + 1;
        }
    }
    return h;
}
```

4.Divisao de tarefas e incumprimentos

O trabalho foi devidido tendo em conta as capacidades de cada membro do grupo e disponibilidade , O aluno Renato Pessego ficou responsavel pela estrutura e bom funcionamento da oficina e estacoes de trabalho como as suas respetivas funções auxiliares , “reparacao” ,”criarcarrosnafila” ,”organizarprioritario” , “criarcarrosreparados” e “colocarcarrosets”. O aluno Leo Ferreira ficou responsavel pelo bom funcionamento da funcao “gravar , carregar oficina” e “imprimir carros reparados”, na seccao da gestao . O aluno Paulo Alexandre foi responsavel pelo bom funcionamento das diversas funcoes relacionadas com as arvores, “numerodevertices” , “insert” ,”adicionar” e o bom funcionamento da função “adicionar prioridade” na gestao . O aluno Francisco Afonseca foi responsavel pelo bom funcionamento das listas de carros , nomeadamente a funcao “criar carro” , estrutura do carro , “remover e adicionar carro” , como tambem a funcao “atualizar tempo de reparacao” na gestao.

Ficaram por fazer algumas funcoes tais como imprimir alfabeticamente a arvore , corretamente , o organizarprioritario funciona com um pequeno precaulco , se o ultimo carro fa lista de espera for prioritario o programa entra dentro de um ciclo e não funciona corretamente . O gravar e carregar oficina , n estavam a funcionar corretamente como desejado .

5.Conclusao

Em jeito de conclusão, este projeto, ajudou-nos a melhorar os nosso conhecimentos em C++ . Apesar de na primeira fase o projeto ter sido implementado com maior facilidade, esta segunda fase foi mais difícil implementar todas as suas funções.

Apesar das adversidades encontradas ao longo do projeto , foi possivel complementar a junção dos dois grupos (Grupo 4 e Grupo 19) .

Este projeto mostrou-nos a verdadeira essência de programar em C++, pois, pode tornar-se uma grande dor de cabeça, principalmente sem usar bibliotecas em que já têm funções pré-definidas. Para realizar por exemplo a lista de espera, em que se podia por exemplo em vez de usar listas ligadas, poderíamos ter usado a classe *queue* que facilitaria muito o trabalho principalemnete na lista des espera onde os carros permaneciam antes de entrarem nas ET.