UNIVERSIDADE da MADEIRA

Faculdade de Ciências Exatas e da Engenharia

Primeira Fase Projeto

Oficina De EDA

Estruturas de Dados e Algoritmos

Professores: Filipe Quintal e Karolinas Baras

Alunos:	**Grupo**: **19**
Leonardo Filipe Nóbrega Ferreira	nº: 2122422
Paulo Alexandre Alves Rodrigues	nº:2120722
Renato Gabriel Silva Pêssego	nº:2121922

Funchal, 24 de abril de 2023

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 fila 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.

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.

Implementação

2.0 Inicialização:

Carregar Ficheiros:

Primeiramente, para realizar o projeto, foi necessário recolher as informações das marcas e dos modelos a utilizar. Diretamente dos ficheiros fornecidos: marcas.txt e modelo.txt, respetivamente.
Para isso usamos a biblioteca <fstream> e criamos uma struct chamada “LinhasFicheiro”, caracterizada pelo seu tamanho e por um array de strings.

Para saber a quantidade de espaço que seria necessário alocar no array de strings decidimos fazer uma função (“int TamanhoFicheiro”), que recebe o nome do ficheiro e retorna um número inteiro das linhas que está a contar. Enquanto está a receber linhas do (“getline”), incrementa o número de linhas. Seguidamente a função (“LinhasFicheiro CarregarFicheirosLista”) que recebe o nome do ficheiro e retorna um objeto do tipo (“LinhasFicheiro”). Esta função aloca um array de strings com o tamanho do ficheiro (“linhas”) e enquanto está a receber linhas do (“getline”), guarda cada linha no array de strings (“linhas”).

```
4   int TamanhoFicheiro(string Nome_Ficheiro) {
5       int numero_linhas = 0;
6       string line;
7       ifstream file(Nome_Ficheiro);
8
9       if (file.is_open())
10      {
11          while (getline(file, line))
12          {
13              numero_linhas++;
14          }
15          file.close();
16      }
17      return numero_linhas;
18  }
```

```
19  LinhasFicheiro CarregarFicheirosLista(string Nome_Ficheiro) {
20
21      LinhasFicheiro novo = LinhasFicheiro();
22
23      string* linhas = new string[TamanhoFicheiro(Nome_Ficheiro)];
24
25      string line;
26
27      ifstream file(Nome_Ficheiro);
28      if (file.is_open())
29      {
30          for (int i = 0; i < TamanhoFicheiro(Nome_Ficheiro);i++)
31          {
32              getline(file, line);
33              *(linhas + i) = line;
34          }
35          file.close();
36      }
37
38      novo.tamanho = TamanhoFicheiro(Nome_Ficheiro);
39      novo.linhas = linhas;
40
41      return novo;
42  }
```

2.1 Inicialização das Estações de Trabalho:

Para inicializar as estações de trabalho, sentimos a necessidade de agrupar as informações numa struct (“Oficina”). Uma oficina é caracterizada por uma array(1) de ETs(2) , o número de ETs, uma fila de espera, sendo esta um array do tipo (“Carro”), o tamanho da fila de espera, o número de todos os carros gerados e os ciclos que correspondem aos dias decorridos na Oficina.

Em seguida, inicializamos a Oficina usando uma função auxiliar que retorna inteiros aleatórios entre 3 e 8 (“numero_de_et”) para gerar o número de ETs e para podermos alocar na array de ETS o espaço necessário. Usamos depois uma função(“criarOficina”) que recebe as marcas e modelos e retorna um objeto do tipo (“Oficina”). Esta contém um for para percorrer a array de ETs e assim atribuir a cada espaço uma função para criar a ET (“CriarET”) que recebe um ID e retorna uma ET e outra função para criar o mecânico (“CriarMecanico”).

A função (“CriarET”) recebe um ID(mas como este está dentro de um for cada ID será atribuído sequencialmente até o ID=número de Ets).Esta função inicializa o que é necessário numa ET, como o Mecânico, a capacidade, a faturação total, um array de carros com os carros a ser reparados(“carros_a_ser_reparados”) e o seu tamanho, um array de carros com os carros reparados (“Carrosreparados”) e o seu tamanho.

A função (“CriarMecanico”) recebe as marcas e retorna um objeto do tipo (“Mecanico”) com uma marca aleatória vindo do ficheiro (“marcas.txt”). É pedido em seguida ao utilizador o nome do mecânico e o preço por dia do mecânico que deve variar entre 0 e 100 euros. O programa está preparado para o caso do utilizador inserir um valor superior ao limite do tipo int, ou até mesmo uma string em vez de um número, sem causar o crash do programa. Para isso recolhemos o input como uma string e usamos a função (“verificarnumero”), para verificar se este input é um número. Caso seja, então convertemos este input para um double e verificamos se esse valor encontra-se dentro dos limites estabelecidos (0 e 100), e, caso esteja, atribuímos esse valor à variável desejada. Caso alguma destas condições não se verifique, o programa pede para o utilizador inserir o valor novamente. É ainda pedida a marca do mecânico. O programa também está preparado para erros simples de escrita, como espaços a mais ou menos e, diferenças de maiúsculas e minúsculas entre a marca escrita pelo utilizador e a

presente no arquivo (“marcas.txt”). Para isso foram criadas duas funções. A (“maiuscula”) que coloca todas as letras maiúsculas, sendo usada para colocar ambas as marcas, tanto a inserida pelo utilizador, como a presente do arquivo de texto, ignorando assim, erros relacionados a isso. A (“removerespacos”), remove todos os espaços em branco de uma string, de forma a evitar erros relacionados a isso. Ambas estas funções serão utilizadas, quando precisarmos comparar strings inseridas pelo utilizador com outras strings, como é o caso de outras funções, como a (“removermecanico”), (“reparacaomanual”), entre outras.

```
4   int numero_de_et() {
5       int et_numero = 3 + (rand() % 5); // numero de ET por oficina random
6       return et_numero;
7   }
8
9   Oficina criarOficina(LinhasFicheiro& marcas, LinhasFicheiro& modelos) {
10      Oficina nova = Oficina();
11      nova.ciclos = 0; //dias de trabalho
12      nova.numero_ets = numero_de_et();
13      nova.ets = new EstacaoTrabalho[nova.numero_ets];
14      nova.fila_espera = new Carro[0];
15      nova.carrostotais = 0;
16      nova.fila_espera_tamanho = 0;
17
18
19      /* Criar os mecanicos conforme o numero de ets gerado*/
20      for (int i = 0; i < nova.numero_ets; i++) {
21          nova.ets[i] = CriarET(i+1);
22          nova.ets[i].mecanico = CriarMecanico(marcas);
23      }
24      /*Criar 10 carros random*/
25      CriarCarrosNaFila(nova,marcas,modelos,10);
26
27
28      //Criar ET; criar os mecanicos; encher a fila ; criar os carros (lista das marcas e dos modelos);
29      return nova;
30  }
31
32
```

```cpp
#include "ET.h"

EstacaoTrabalho CriarET(int ID) {
    EstacaoTrabalho nova = EstacaoTrabalho();
    nova.mecanico;
    nova.capacidade = 2 + (rand() % 3);
    nova.faturacao = 0;
    nova.ID = ID;
    nova.Carrosreparados = new Carro[0];
    nova.carros_a_ser_reparados = new Carro[nova.capacidade];
    nova.num_carros_a_ser_reparados = 0;
    return nova;
}
```

```cpp
bool verificarnumero(string n) {
    if (n.length() == 0) {
        return false;
    }
    for (int i = 0; i < n.length(); i++) {
        if (isdigit(n[i]) == false) {
            return false;
        }
    }
    return true;
}
```

```cpp
Mecanico CriarMecanico(LinhasFicheiro& marcas)
{
    Mecanico novo = Mecanico();
    string entrada;
    double precotemp = 0;
    int preco = 0;
    string marca;
    bool sair = false;

    while (!sair) {
        cout << "Insira uma marca valida do mecanico: " << endl;
        getline(cin, marca);
        for (int i = 0; i < marcas.tamanho; i++) {
            if (removerespacos(maiuscula(marca)) == removerespacos(maiuscula(marcas.linhas[i]))) {
                novo.marca = marcas.linhas[i];
                sair = true;
                break;
            }
        }

    }
    cout << "Introduza o nome do mecanico " << endl;
    cin >> ws;
    getline(cin,novo.nome);
    while ((precotemp > 100) || (precotemp <= 0)) {
        cout << "Introduza um preco por dia do mecanico valido (inteiro entre 0 e 100)" << endl;
        getline(cin,entrada);
        while (!verificarnumero(entrada)) {
            cout << "Entrada invalida!" << endl << "Introduza o preco por dia do mecanico (inteiro entre 0 e 100)" << endl;
            getline(cin,entrada);
        }
    precotemp = stod(entrada);
    }
    preco = (int)precotemp;
    novo.preco_reparacao_por_dia = preco;
    return novo;
}
```

```
string maiuscula(string str) {
    int tamanho = str.length();
    char* novo = new char[tamanho];
    for (int i = 0; i <= tamanho; i++) {
        novo[i] = toupper(str[i]);
    }
    return novo;
}

string removerespacos(string str) {
    int tamanho = str.length();
    int k = 0;
    char* novo = new char[tamanho];

    for (int i = 0; i <= str.length(); i++) {
        if (str[i] != ' ') {
            novo[k] = str[i];
            k++;
            tamanho--;
        }
    }
    return novo;
}
```

2.2 Inicialização dos carros:

Para inicializar os carros usamos uma função auxiliar (“GerarProbalidades”) que recebe um double, como probabilidade e retorna um bool que será usado para fazer o carro prioritário. Primeiramente usamos uma função para os criar (“CriarCarro”) que recebe as marcas e os modelos e retorna um objeto do tipo (“Carro”). Esta função cria um carro com uma marca e um modelo aleatório, com um tempo de reparação também aleatório entre 2 e 5, e com 5% de chance de ser um carro prioritário.

Para uma melhor gestão do projeto sentimos a necessidade de adicionar duas funções essenciais para a manipulação de arrays. O (“Adiciona”) e o (“Remove”), ambas recebem o array de carros e o seu tamanho, sendo que (“Adiciona”) também recebe o carro a receber e o (“Remove”) recebe o índice a remover.
O (“Adiciona”) coloca o carro recebido no final do array. O (“Remove”) remove o índice que foi passado no último argumento da sua função.

Para colocar carros na fila de espera usamos a função (“CriarCarrosNaFila”) criando um novo objeto do tipo (“Carro”) que será a invocação da função (“CriarCarro”). O ID do carro criado será o número de carros totais + 1, assim um carro válido terá sempre um ID único e maior que 0. Seguidamente, usamos a função (“MarcaPresente”) dentro de um if para saber se a marca que foi criada no (“Carro”) é igual a uma das marcas dos mecânicos presentes nas ETs, se for verdadeiro invoca a função (“Adiciona”) e atualiza os carros totais. Durante todo o programa, é usado algumas vezes o system(“CLS”), que apaga toda a informação da tela, tornando tudo mais legível ao utilizador.

```
7   bool  GerarProbalidades(double probalidade) {
8       bool probalidade_final =  rand() < (probalidade * (RAND_MAX + 1.0));
9       return probalidade_final;
10  }
11
```

```
13  Carro CriarCarro(LinhasFicheiro &marcas, LinhasFicheiro &modelos) {
14      Carro novo_carro = Carro();
15      novo_carro.marca = marcas.linhas[rand() % marcas.tamanho];
16      novo_carro.modelo = modelos.linhas[rand() % modelos.tamanho];
17      novo_carro.ID = 0;
18      novo_carro.tempo_reparacao_max = 2 + (rand() % 3);
19      novo_carro.dias_em_reparacao = 0;
20      novo_carro.prioritario = GerarProbalidades(0.05);
21      novo_carro.custoreparacao = 0;
22
23      return novo_carro;
24  }
```

```cpp
/*Recebe o array de carros, o tamanho e o carro a receber*/
/*Adiciona elemento no final do array*/
void Adiciona(Carro*& v, int& tamanho, Carro& carro)
{
    int tamanho_novo = tamanho + 1;
    Carro* novo = new Carro[tamanho_novo];
    for (int i = 0; i < tamanho; i++) {
        novo[i] = v[i];
    }
    novo[tamanho] = carro;

    delete[] v;

    v = novo;
    tamanho = tamanho_novo;
}
```

```cpp
/*Recebe o array de carros, o tamanho e o indice a ser removido*/
/*Verifica que o array nao fica vazio com o if*/
/*Copia os itens até o indice a ser removido e pega nos itens que estao na direita do el. a remover*/
/*Deslocando-os para a esquerda*/
void Remove(Carro*& v, int& tamanho, int& ind)
{
    int tamanho_novo = tamanho - 1;
    Carro* novo = new Carro[tamanho_novo];
    if (tamanho_novo > 0) {
        for (int i = 0; i < ind; i++) {
            novo[i] = v[i];
        }
        for (int i = ind; i < tamanho_novo; i++) {
            novo[i] = v[i + 1];
        }
    }
    v = novo;
    tamanho = tamanho_novo;

}

```

```cpp

bool MarcaPresente(Oficina &Of, string marca) {
    bool temp = false;
    for (int i = 0; i < Of.numero_ets; i++) {
        temp = (marca == Of.ets[i].mecanico.marca);
        if (temp) {
            break;
        }
    }

    return temp;
}

```

```
50  void CriarCarrosNaFila(Oficina& Of, LinhasFicheiro& marcas, LinhasFicheiro& modelos, int num) {
51      for (int i = 0; i < num; i) {
52          Carro novo = CriarCarro(marcas, modelos);
53          novo.ID = Of.carrostotais + 1;
54          if (MarcaPresente(Of, novo.marca)) {
55                  Adiciona(Of.fila_espera, Of.fila_espera_tamanho, novo);
56                  Of.carrostotais = Of.carrostotais + 1;
57                  i++;
58          }
59      }
60      system("CLS");
61  }
```

2.3 Funcionamento:

Para simular a oficina por ciclos(3), como quando a oficina é criada não existem carros nas ETs não poderemos usar a função (“reparacão”), mas sim apenas a partir do dia 1 da oficina. Sentimos a necessidade de criar uma função (“colocarprioritario”) que recebe a fila de espera da oficina e percorre-a, transportando os carros prioritários para o início da fila, através da função (“Transportar”). Esta função (“Transportar”) cria uma nova array onde o elemento a ser transportado será colocado logo atrás do último carro prioritário da fila. Esta função identifica quantos carros prioritários já existem na fila e mantem o array igual até o ultimo prioritário desta, depois coloca o elemento a ser transportado, o ultimo prioritário a chegar, e por fim, coloca todos os outros carros pela mesma ordem de antes, pela ordem de chegada. Assim, todos os carros prioritários são colocados sempre no início da fila de espera, pela sua ordem de chegada.

Seguidamente usamos dois menus (“Menu”) e (“MenuInfo”) para apresentar as informações relevantes para a simulação e 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 array de carros reparados e, removido da array de carros a ser reparados, tornando o seu ID = 0, tornando-o um carro inválido, ou seja, que não está a ocupar um lugar neste array, sendo este substituido assim que possível. 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 mais ou 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 da struct de Carro. A chance, nesta função, é feita, gerando, um número aleatório entre 1 e 100, e verificando se o número gerado está

entre 1 e 15, pois há 15% de chance de um número aleatório entre 1 e 100 estar entre estes valores. 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.

```
void Transportar(Oficina& Of, int ind) {
    int tamanho = Of.fila_espera_tamanho;
    int indice = 0;
    for (int z = 0; z < ind; z++) {
        if (Of.fila_espera[z].prioritario) {
            indice = indice + 1;
        }
    }
    Carro* novo = new Carro[tamanho];
    novo[indice] = Of.fila_espera[ind];
    for (int k = 0; k < indice; k++) {
        novo[k] = Of.fila_espera[k];
    }
    for (int i = indice; i < ind; i++) {
        novo[i + 1] = Of.fila_espera[i];
    }
    for (int j = ind + 1; j < tamanho; j++) {
        novo[j] = Of.fila_espera[j];
    }

    for (int i = 0; i < tamanho; i++) {
        Of.fila_espera[i] = novo[i];
    }
    delete[] novo;
}
```

```
88  void colocarprioritario(Oficina& Of) {
89      for (int i = 0; i < Of.fila_espera_tamanho; i++) {
90          if (Of.fila_espera[i].prioritario) {
91              Transportar(Of, i);
92          }
93      }
94  }
95
```

```
void reparacao(EstacaoTrabalho& ID1) {
    int chance;

    for (int i = 0; i < ID1.capacidade; i++) {
        if ((ID1.carros_a_ser_reparados[i].dias_em_reparacao < ID1.carros_a_ser_reparados[i].tempo_reparacao_max) && (ID1.carros_a_ser_reparados[i].ID != 0)) {
            chance = rand() % 100 + 1;
            if ((chance >= 1 and chance <= 15) && (ID1.carros_a_ser_reparados[i].dias_em_reparacao > 0)) {
                int tamanho = (ID1.num_carros_reparados);
                Adiciona(ID1.Carrosreparados, tamanho, ID1.carros_a_ser_reparados[i]);

                ID1.num_carros_a_ser_reparados = ID1.num_carros_a_ser_reparados - 1;
                ID1.carros_a_ser_reparados[i].ID = 0;
                ID1.faturacao = ID1.faturacao + (ID1.carros_a_ser_reparados[i].dias_em_reparacao * ID1.mecanico.preco_reparacao_por_dia);
                ID1.Carrosreparados[ID1.num_carros_reparados].custoreparacao = ID1.mecanico.preco_reparacao_por_dia * ID1.Carrosreparados[ID1.num_carros_reparados].dias_em_reparacao;
                ID1.num_carros_reparados = ID1.num_carros_reparados + 1;

            }
            else
                ID1.carros_a_ser_reparados[i].dias_em_reparacao++;
        }

        else if ((ID1.carros_a_ser_reparados[i].dias_em_reparacao >= ID1.carros_a_ser_reparados[i].tempo_reparacao_max) && (ID1.carros_a_ser_reparados[i].ID != 0)) {
            int tamanho = (ID1.num_carros_reparados);
            Adiciona(ID1.Carrosreparados, tamanho, ID1.carros_a_ser_reparados[i]);

            ID1.num_carros_a_ser_reparados = ID1.num_carros_a_ser_reparados - 1;
            ID1.carros_a_ser_reparados[i].ID = 0;
            ID1.faturacao = ID1.faturacao + (ID1.carros_a_ser_reparados[i].dias_em_reparacao * ID1.mecanico.preco_reparacao_por_dia);
            ID1.Carrosreparados[ID1.num_carros_reparados].custoreparacao = ID1.mecanico.preco_reparacao_por_dia * ID1.Carrosreparados[ID1.num_carros_reparados].dias_em_reparacao;
            ID1.num_carros_reparados = ID1.num_carros_reparados + 1;
        }
    }
}
```

```cpp
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, marcas, modelos);
            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);

}
```

```
void MenuInfo(Oficina& Of,LinhasFicheiro& marcas,LinhasFicheiro& modelos){
    cout << "Dia: " << Of.ciclos << endl;
    for (int i = 0; i < Of.numero_ets; i++) {
        cout << "ET: " << Of.ets[i].ID << " | " << "Mecanico: " << Of.ets[i].mecanico.nome
            << " | " << "Capacidade: " << Of.ets[i].capacidade << " | " << "Carros: " << Of.ets[i].num_carros_a_ser_reparados << " | " <<
            "Marca: " << Of.ets[i].mecanico.marca << " | " << "Total Faturacao: " << Of.ets[i].faturacao << endl;
        if (Of.ciclos >= 1) {
            cout << "Carros a ser reparados: " << endl;
        }
        int r = Of.ets[i].capacidade;
        if(Of.ets[i].num_carros_a_ser_reparados>0){
            for (int t = 0; t < r; t++) {
                if ((Of.ciclos >= 1) && (Of.ets[i].carros_a_ser_reparados[t].ID != 0)) {
                    cout << "ID: " << Of.ets[i].carros_a_ser_reparados[t].ID << " | " << "Carro: " << Of.ets[i].carros_a_ser_reparados[t].marca << "-"
                        << Of.ets[i].carros_a_ser_reparados[t].modelo << " | " << "Prioritario: ";
                    if (Of.ets[i].carros_a_ser_reparados[t].prioritario == 1) {
                        cout << "Sim";
                    }
                    else {
                        cout << "Nao";
                    }
                    cout << " | " << "Tempo de reparacao: " << Of.ets[i].carros_a_ser_reparados[t].dias_em_reparacao << " | " << "Tempo de reparacao maximo: ";
                    cout << Of.ets[i].carros_a_ser_reparados[t].tempo_reparacao_max << endl;
                }
            }
    }
        cout << endl;



    if (Of.ciclos >= 1) {
            cout << "Carros reparados: " << endl;
        }
        int g = Of.ets[i].num_carros_reparados;
        for (int t = 0; t < g; t++) {
            if ((Of.ciclos >= 1) && (Of.ets[i].Carrosreparados[t].ID != 0)) {
                cout << "ID: " << Of.ets[i].Carrosreparados[t].ID << " | " << "Carro: " << Of.ets[i].Carrosreparados[t].marca << "-"
                    << Of.ets[i].Carrosreparados[t].modelo << " | " << "Prioritario: ";
                if (Of.ets[i].Carrosreparados[t].prioritario == 1) {
                    cout << "Sim";
```

```
                cout << "ID: " << Of.ets[i].Carrosreparados[t].ID << " | " << "Carro: " << Of.ets[i].Carrosreparados[t].marca << "-"
                    << Of.ets[i].Carrosreparados[t].modelo << " | " << "Prioritario: ";
                if (Of.ets[i].Carrosreparados[t].prioritario == 1) {
                    cout << "Sim";
                }
                else {
                    cout << "Nao";
                }
                cout << " | " << "Tempo de reparacao: " << Of.ets[i].Carrosreparados[t].dias_em_reparacao << " | " << "Tempo de reparacao maximo: ";
                cout << Of.ets[i].Carrosreparados[t].tempo_reparacao_max << "| Custo de reparacao: " << Of.ets[i].Carrosreparados[t].custoreparacao << endl;
            }

        }
        cout << endl;



    }
    cout << "--------------------------------------------------------------------------------" << endl;
    cout << "Lista de Espera: " << endl;
    for (int i = 0; i < Of.fila_espera_tamanho; i++) {
        cout << "ID: " << Of.fila_espera[i].ID << " | " << "Modelo: " << Of.fila_espera[i].modelo << " | " << "Marca: " << Of.fila_espera[i].marca << " | "
            << "Prioritario: ";
        if (Of.fila_espera[i].prioritario == 1) {
            cout << "Sim";
        }
        else {
            cout << "Nao";
        }
        cout<< " | " << "Tempo de reparacao maximo: " << Of.fila_espera[i].tempo_reparacao_max << endl;
    }
}
```

```
void seguinte(Oficina& Of, LinhasFicheiro& marcas, LinhasFicheiro& modelos)
{
    for (int i = 0; i < Of.numero_ets; i++) {
        reparacao(Of.ets[i]);

    }
    Of.ciclos++;
    CriarCarrosNaFila(Of, marcas, modelos, 10);
    colocarprioritario(Of);
    ColocarCarrosET(Of, 8);

}
```

3. Gestão

Esta parte do código, a gestão de uma oficina de carros, contém diversas funções que permitem aos utilizadores do sistema executar diferentes tarefas manualmente. Para a escolha da função desejada, o utilizador é apresentado com o menu (“gestão”), responsável por chamar as funções selecionadas pelo utilizador.

```
void gestao(Oficina& Of, LinhasFicheiro& marcas, LinhasFicheiro& modelos) {
    int opcao;
    string escolha1temp;
    double escolha1temporaria = 0;
    int escolha = 0;
    string escolha2temp;
    double escolha2temporaria = 0;
    int escolha2 = 0;
    string caminho = "oficina.txt";
    bool sair = false;
    do {
        cout << " ***** Bem Vindo Gestor *****" << endl;
        cout << "(1).Reparacao Manual" << endl;
        cout << "(2).Atualizar tempo de reparacao" << endl;
        cout << "(3).Adicionar Prioridade" << endl;
        cout << "(4).Remover Mecanico" << endl;
        cout << "(5).Gravar Oficina" << endl;
        cout << "(6).Carregar Oficina" << endl;
        cout << "(7).Imprimir Oficina" << endl;
        cout << "(8).Sair da gestao" << endl;
        cout << "Seleccione a sua opcao : ";
        cin >> opcao;

        switch (opcao) {

        case 1:
            reparacao_manual(Of);
            system("CLS");

            MenuInfo(Of, marcas, modelos);
            Menu(Of, marcas, modelos);
            sair = true;
            break;

        case 2:
            atualizar_tempo_reparacao(Of);
            system("CLS");
            MenuInfo(Of, marcas, modelos);
            Menu(Of, marcas, modelos);
```

```
case 2:
    atualizar_tempo_reparacao(Of);
    system("CLS");
    MenuInfo(Of, marcas, modelos);
    Menu(Of, marcas, modelos);
    sair = true;
    break;

case 3:
    adicionar_prioridade(Of);
    system("CLS");
    MenuInfo(Of, marcas, modelos);
    Menu(Of, marcas, modelos);
    sair = true;
    break;

case 4:
    remover_mecanico(Of, marcas);
    system("CLS");
    MenuInfo(Of, marcas, modelos);
    Menu(Of, marcas, modelos);
    sair = true;
    break;

case 5:
    gravar_oficina(Of);
    system("CLS");
    MenuInfo(Of, marcas, modelos);
    Menu(Of, marcas, modelos);
    sair = true;
    break;
case 6:

    while ((escolha2temporaria > 2) || (escolha2temporaria <= 0)) {
        cout << "Digite: " << endl << "(1) Escolher o caminho manualmente" << endl << "(2) Usar o caminho padrao (Oficina.txt)" << endl;
        cin >> ws;
        getline(cin, escolha2temp);
        while (!verificarnumero(escolha2temp)) {
```

```
case 6:

    while ((escolha2temporaria > 2) || (escolha2temporaria <= 0)) {
        cout << "Digite: " << endl << "(1) Escolher o caminho manualmente" << endl << "(2) Usar o caminho padrao (Oficina.txt)" << endl;
        cin >> ws;
        getline(cin, escolha2temp);
        while (!verificarnumero(escolha2temp)) {
            cout << "Opcao invalida! Tente novamente" << endl;;
            cout << "Digite: " << endl << "(1) Escolher o caminho manualmente" << endl << "(2) Usar o caminho padrao (Oficina.txt)" << endl;
            getline(cin, escolha2temp);
        }
        escolha2temporaria = stod(escolha2temp);
        if ((escolha2temporaria > 2) || (escolha2temporaria <= 0)) {
            escolha2temporaria = 0;
        }
        else {
            escolha2 = escolha2temporaria;
        }
    }
    switch (escolha2) {

    case 1:
        cout << "Digite o caminho (inclua .txt ao fim do nome do arquivo): " << endl;
        cin >> caminho;
        break;
    case 2:
        break;
    default:
        cout << "Opcao invalida" << endl << "Imprimindo usando o caminho padrao..." << endl;
        break;
    }

    carregar_oficina(Of, caminho);
    system("CLS");
    MenuInfo(Of, marcas, modelos);
    Menu(Of, marcas, modelos);
    sair = true;
    break;
```

```
        case 7:


            while ((escolha1temporaria > 2) || (escolha1temporaria <= 0)) {
                cout << "Escolha como quer imprimir a Oficina" << endl << "(1) Alfabeticamente" << endl << "(2) Por dias em reparacao" << endl;
                cin >> ws;
                getline(cin, escolha1temp);
                while (!verificarnumero(escolha1temp)) {
                    cout << "Opcao invalida! Tente novamente" << endl;;
                    cout << "Escolha como quer imprimir a Oficina" << endl << "(1) Alfabeticamente" << endl << "(2) Por dias em reparacao" << endl;
                    getline(cin, escolha1temp);
                }
                escolha1temporaria = stod(escolha1temp);
                if ((escolha1temporaria > 2) || (escolha1temporaria <= 0)) {
                    escolha1temporaria = 0;
                }
                else {
                    escolha = escolha1temporaria;
                }
            }
            escolha = (int)escolha1temporaria;
            if (escolha == 1) {
                system("CLS");
                cout << "Lista ordenada alfabeticamente: " << endl;
                imprimir_oficinaalfabeticamente(Of);
                cout << endl;
            }
            else if (escolha == 2) {
                system("CLS");
                cout << "Lista ordenada por tempo de reparacao: " << endl;
                imprimir_oficinaportempo(Of);
                cout << endl;
            }
            else {
                cout << "Opcao invalida" << endl;
            }
            MenuInfo(Of, marcas, modelos);
```

```
            cout << endl;
        }
        else {
            cout << "Opcao invalida" << endl;
        }
        MenuInfo(Of, marcas, modelos);
        Menu(Of, marcas, modelos);
        sair = true;
        break;
    case 8:
        Menu(Of, marcas, modelos);
        sair = true;

        break;

    default:
        cout << "Opcao invalida" << endl;
        break;
    }
} while (!sair);
}
```

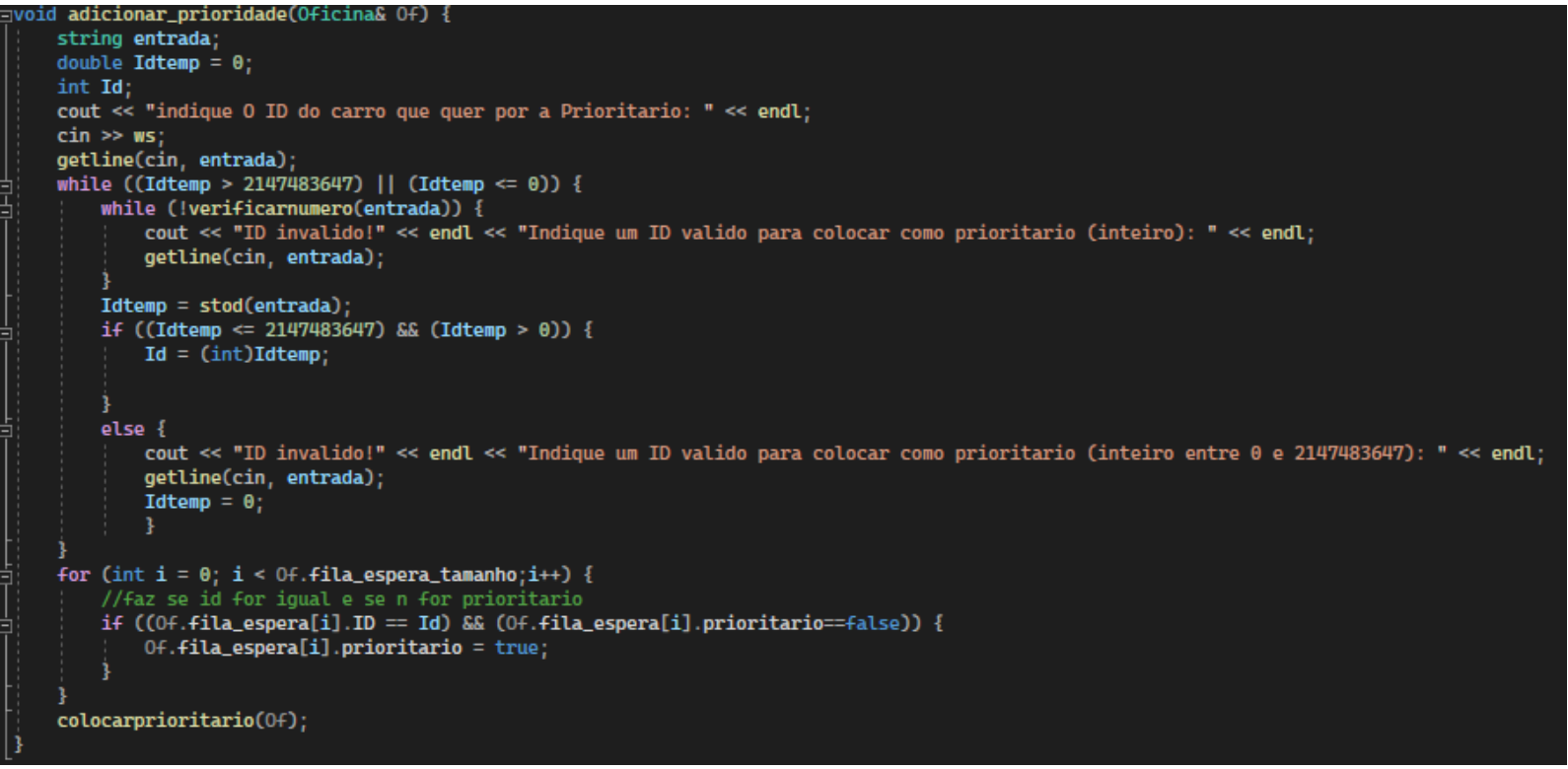
3.1. A primeira função, "reparacao_manual", permite que o utilizador introduza a marca e modelo de um carro para que seja reparado manualmente. O sistema percorre todas as ETs da oficina e verifica se os carros com a marca e modelo especificados estão presentes na lista de carros a serem reparados. Se os carros forem encontrados, são adicionados à lista de carros reparados da respetiva ET e removidos da lista de carros a serem reparados, de forma idêntica à reparação automática descrita anteriormente. A faturação da ET é atualizada de acordo com o tempo de reparação dos carros e o preço por dia de reparação, como explicado previamente.

```
void reparacao_manual(Oficina& Of) {

    string marca;
    string modelo;
    cout << "indique a marca a reparar: " << endl;

    //ws tira os espaços em branco
    cin >> ws;
    getline(cin, marca);
    cout << "indique o modelo a reparar: " << endl;
    cin >> ws;
    getline(cin, modelo);
    /*Percorre ETs*/
    for (int i = 0; i < Of.numero_ets; i++) {

        /*Percorre carros a serem reparados*/
        for (int j = 0; j < Of.ets[i].capacidade; j++) {
            /*Se alguma das marcas corresponder à marca e modelo em questao*/
            if (removerespacos(maiuscula((Of.ets[i].carros_a_ser_reparados[j].marca))) == removerespacos(maiuscula(marca))
                && (removerespacos(maiuscula(Of.ets[i].carros_a_ser_reparados[j].modelo)) == removerespacos(maiuscula(modelo)))
                && (Of.ets[i].carros_a_ser_reparados[j].ID != 0) && (Of.ets[i].carros_a_ser_reparados[j].dias_em_reparacao >= 1)) {
                int tamanho = (Of.ets[i].num_carros_reparados);
                Adiciona(Of.ets[i].Carrosreparados, tamanho, Of.ets[i].carros_a_ser_reparados[j]);
                Of.ets[i].num_carros_a_ser_reparados = Of.ets[i].num_carros_a_ser_reparados - 1;
                Of.ets[i].carros_a_ser_reparados[j].ID = 0;
                Of.ets[i].faturacao = Of.ets[i].faturacao + (Of.ets[i].carros_a_ser_reparados[j].dias_em_reparacao * Of.ets[i].mecanico.preco_reparacao_por_dia);
                Of.ets[i].Carrosreparados[Of.ets[i].num_carros_reparados].custoreparacao = Of.ets[i].mecanico.preco_reparacao_por_dia * Of.ets[i].Carrosreparados[Of.ets[i].num_carros_reparados].dias_em_reparacao;
                Of.ets[i].num_carros_reparados = Of.ets[i].num_carros_reparados + 1;

            }
        }

    }
}
```

3.2. A segunda função, "atualizar_tempo_reparacao", permite que o utilizador atualize o tempo máximo de reparação de um ou mais carros na fila de espera. O sistema pede ao utilizador que introduza a marca e modelo dos carros a serem atualizados e, em seguida, o novo tempo máximo de reparação. O sistema percorre a fila de espera e atualiza o tempo máximo de reparação dos carros especificados. Tal como todos os menus anteriores, os menus destas funções gestão, também foram pensados para serem à prova de erros do utilizador.

```
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 > 2147483647) || (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 <= 2147483647) &&( 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;
        }
    }

    for (int i = 0; i < Of.fila_espera_tamanho;i++) {
        if (removerespacos((maiuscula(Of.fila_espera[i].marca))) == removerespacos(maiuscula(marca)) && (removerespacos(maiuscula(Of.fila_espera[i].modelo)) == removerespacos(maiuscula(modelo)))) {
            Of.fila_espera[i].tempo_reparacao_max = tempo;

        }
    }
}
```

3.3. A terceira função, "adicionar_prioridade", permite que o utilizador adicione um carro à lista de prioridades da fila de espera. O sistema pede ao utilizador que introduza o ID do carro a ser adicionado à lista de prioridades. O programa percorre a fila de espera e, se encontrar o carro com o ID especificado e, se ele ainda não estiver na lista de prioridades, adiciona-o. Por fim, usa a função colocar prioritário, de forma a colocar os carros prioritários no início da fila, pela sua ordem de chegada, como descrito anteriormente.



```
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 > 2147483647) || (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 <= 2147483647) && (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;
            }
    }
    for (int i = 0; i < Of.fila_espera_tamanho;i++) {
        //faz se id for igual e se n for prioritario
        if ((Of.fila_espera[i].ID == Id) && (Of.fila_espera[i].prioritario==false)) {
            Of.fila_espera[i].prioritario = true;
        }
    }
    colocarprioritario(Of);
}
```

3.4. Este código implementa uma função chamada "remover_mecanico" que recebe dois parâmetros, um do tipo "Oficina" e outro do tipo "LinhasFicheiro". O objetivo da função é remover um mecânico da oficina, que é identificado pelo seu nome.

A função começa por pedir ao usuário o nome do mecânico que deseja remover e, em seguida, percorre todas as ETs que estão na oficina. Se o mecânico a ser removido estiver associado a alguma das ETs, então essa ET é identificada e o mecânico é removido.

A função então percorre todos os carros que estavam em reparação com o mecânico removido, colocando-os na lista de carros reparados, atualizando o número de carros reparados e a faturação total da ET. O carro é removido da lista de carros em reparação, atualizando o número de carros em reparação.

Em seguida, a função remove o mecânico da ET e pede ao usuário para digitar as novas informações de um novo mecânico para a ET. O novo mecânico é adicionado, à ET, da mesma forma que os outros mecânicos, e a função atualiza todas as informações da ET.

Em suma, esta função é responsável por remover um mecânico da oficina e atualizar as informações de todas as ETs afetadas por essa remoção. Além disso, a função adiciona um novo mecânico à ET, atualizando as suas informações.

```
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);
    for (int i = 0; i < Of.numero_ets;i++) {
        if (removerespacos(maiuscula(Of.ets[i].mecanico.nome)) == removerespacos(maiuscula(mecanicoRem))) {


            //repara todos os carros da et
            for (int j = 0; j < Of.ets[i].capacidade; j++) {

                if (Of.ets[i].carros_a_ser_reparados[j].ID != 0) {
                    //coloca carro em lista de carros reparados
                    int tamanho = (Of.ets[i].num_carros_reparados);
                    Adiciona(Of.ets[i].Carrosreparados, tamanho, Of.ets[i].carros_a_ser_reparados[j]);
                    Of.ets[i].num_carros_a_ser_reparados = Of.ets[i].num_carros_a_ser_reparados - 1;
                    Of.ets[i].carros_a_ser_reparados[j].ID = 0;
                    Of.ets[i].faturacao = Of.ets[i].faturacao + (Of.ets[i].carros_a_ser_reparados[j].dias_em_reparacao * Of.ets[i].mecanico.preco_reparacao_por_dia);
                    Of.ets[i].Carrosreparados[Of.ets[i].num_carros_reparados].custoreparacao = Of.ets[i].mecanico.preco_reparacao_por_dia * Of.ets[i].Carrosreparados[Of.ets[i].num_carros_reparados].dias_em_reparacao;
                    Of.ets[i].num_carros_reparados = Of.ets[i].num_carros_reparados + 1;
                }
            }

            //adicionar novo mecanico
            cout << "O mecanico com o nome " << mecanicoRem << ", foi removido da et" << endl;
            cout << "ET: " << Of.ets[i].ID << " | " << "Mecanico: " << Of.ets[i].mecanico.nome
                << " | " << "Capacidade: " << Of.ets[i].capacidade << " | " <<
                "Marca: " << Of.ets[i].mecanico.marca << " | " << "Total Faturacao: " << Of.ets[i].faturacao << endl;
            cout << endl;
            Mecanico novo = CriarMecanico(marcas);

            Of.ets[i] = CriarET(Of.ets[i].ID);
            Of.ets[i].mecanico = novo;
        }
    }
}
```

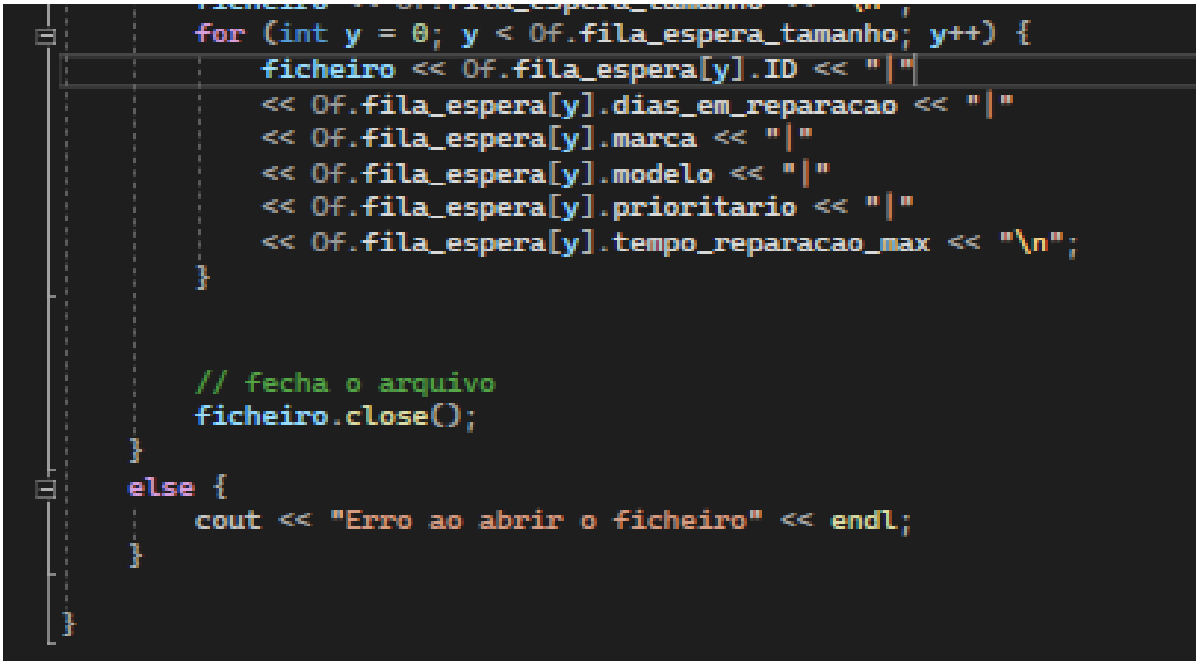
3.5. A função gravarOficina tem como objetivo gravar os dados do estado atual da "Oficina" em um arquivo de texto. Esses dados são organizados em um formato específico, separados por "|" e "\n", de modo que possam ser lidos e interpretados posteriormente, sendo estas separações utilizadas, para definir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

    string caminho = "oficina.txt";

    // cria um objeto ofstream
    ofstream ficheiro;

    // abre o ficheiro
    ficheiro.open(caminho);

    // verifica se o ficheiro foi aberto com sucesso
    if (ficheiro.is_open()) {

        ficheiro << Of.ciclos << "|"
        << Of.carrostotais << "|"
        << Of.numero_ets << "\n";
        //ets
        for (int i = 0; i < Of.numero_ets;i++) {
            ficheiro << Of.ets[i].ID << "|"
            << Of.ets[i].capacidade << "|"
            << Of.ets[i].faturacao << "|"

            << Of.ets[i].num_carros_a_ser_reparados << "|"
            << Of.ets[i].num_carros_reparados << "\n";
            for (int j = 0; j < Of.ets[i].num_carros_reparados; j++) {
                ficheiro << Of.ets[i].Carrosreparados[j].ID << "|"
                << Of.ets[i].Carrosreparados[j].dias_em_reparacao << "|"
                << Of.ets[i].Carrosreparados[j].marca << "|"
                << Of.ets[i].Carrosreparados[j].modelo << "|"
                << Of.ets[i].Carrosreparados[j].prioritario << "|"
                << Of.ets[i].Carrosreparados[j].custoreparacao << "|"
                << Of.ets[i].Carrosreparados[j].tempo_reparacao_max << "\n";
            }
            //carros a ser reparados
            for (int k = 0; k < Of.ets[i].capacidade;k++) {
                ficheiro << Of.ets[i].carros_a_ser_reparados[k].dias_em_reparacao << "|"
                << Of.ets[i].carros_a_ser_reparados[k].ID << "|"
                << Of.ets[i].carros_a_ser_reparados[k].marca << "|"
                << Of.ets[i].carros_a_ser_reparados[k].modelo << "|"
                << Of.ets[i].carros_a_ser_reparados[k].prioritario << "|"
                << Of.ets[i].carros_a_ser_reparados[k].tempo_reparacao_max <<"\n";
            }
            //mecanico
            ficheiro << Of.ets[i].mecanico.marca << "|"
            << Of.ets[i].mecanico.nome << "|"
            << Of.ets[i].mecanico.preco_reparacao_por_dia << "\n";
        }
        //fila de espera
        ficheiro << Of.fila_espera_tamanho << "\n";
        for (int y = 0; y < Of.fila_espera_tamanho; y++) {
            ficheiro << Of.fila_espera[y].ID << "|"
```



3.6. 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” temporaria é atribuída à "Oficina" original, passada como parâmetro para a função “carregar_oficina”

```
void carregar_oficina(Oficina& Of, string caminho) {
    //string caminho = "oficina.txt";


    ifstream ficheiro(caminho);

    if (ficheiro.is_open()) {

        Oficina nova = Oficina();
        string linha;

        getline(ficheiro, linha);

        stringstream ss(linha);
        string temp;

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

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

        //obtem numero_ets
        getline(ss, temp, '|');
        nova.numero_ets = stoi(temp);
        //cria as ets
        nova.ets = new EstacaoTrabalho[nova.numero_ets];

        for (int i = 0; i < nova.numero_ets;i++) {

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

            //obtem ID
            getline(ss, temp, '|');
            nova.ets[i].ID = stoi(temp);
            //obtem capacidade
            getline(ss, temp, '|');
            nova.ets[i].capacidade = stoi(temp);
            //faturacao
            getline(ss, temp, '|');
            nova.ets[i].faturacao = stof(temp);

            //num carros a ser reparados
            getline(ss, temp, '|');
            nova.ets[i].num_carros_a_ser_reparados = stoi(temp);
            //carros reparados
            getline(ss, temp, '|');
            nova.ets[i].num_carros_reparados = stoi(temp);
```

```
getline(ss, temp, '|');
nova.ets[i].num_carros_reparados = stoi(temp);
nova.ets[i].Carrosreparados = new Carro[nova.ets[i].num_carros_reparados];
if (nova.ets[i].num_carros_reparados > 0) {

    for (int j = 0; j < nova.ets[i].num_carros_reparados; j++) {
        getline(ficheiro, linha);
        stringstream ss(linha);
        //obtem o id do carro reparado
        getline(ss, temp, '|');
        nova.ets[i].Carrosreparados[j].ID = stoi(temp);
        //obtem o numero de dias de reparacao
        getline(ss, temp, '|');
        nova.ets[i].Carrosreparados[j].dias_em_reparacao = stoi(temp);
        //obtem a marca
        getline(ss, temp, '|');
        nova.ets[i].Carrosreparados[j].marca = temp;
        //obtem o modelo
        getline(ss, temp, '|');
        nova.ets[i].Carrosreparados[j].modelo = temp;
        //obtem prioritario
        getline(ss, temp, '|');
        nova.ets[i].Carrosreparados[j].prioritario = stoi(temp);
        //obtem custo de reparacao
        getline(ss, temp, '|');
        nova.ets[i].Carrosreparados[j].custoreparacao = stoi(temp);
        //obtem tempo de reparacao maximo
        getline(ss, temp, '|');
        nova.ets[i].Carrosreparados[j].tempo_reparacao_max = stoi(temp);

    }
}
//carros a ser reparados
nova.ets[i].carros_a_ser_reparados = new Carro[nova.ets[i].capacidade];
if (nova.ets[i].capacidade > 0) {

    for (int k = 0; k < nova.ets[i].capacidade; k++) {
        getline(ficheiro, linha);
        stringstream ss(linha);
        //obtem dias em reparacao
        getline(ss, temp, '|');
        nova.ets[i].carros_a_ser_reparados[k].dias_em_reparacao = stoi(temp);
        //obtem id do carro em peracao
        getline(ss, temp, '|');
        nova.ets[i].carros_a_ser_reparados[k].ID = stoi(temp);
        //obtem marca
        getline(ss, temp, '|');
        nova.ets[i].carros_a_ser_reparados[k].marca = temp;
        //obtem modelo
        getline(ss, temp, '|');
        nova.ets[i].carros_a_ser_reparados[k].modelo = temp;
        //obtem prioridade
        getline(ss, temp, '|');
        nova.ets[i].carros_a_ser_reparados[k].prioritario = stoi(temp);
        //obtem tempo de reparacao maximo
        getline(ss, temp, '|');
        nova.ets[i].carros_a_ser_reparados[k].tempo_reparacao_max = stoi(temp);
```

```
                //obtem tempo de reparacao maximo
                getline(ss, temp, '|');
                nova.ets[i].carros_a_ser_reparados[k].tempo_reparacao_max = stoi(temp);

            }
        }
        getline(ficheiro, linha);
        stringstream ss1(linha);
        //mecanico
        nova.ets[i].mecanico = Mecanico();
        //obtem marca
        getline(ss1, temp, '|');
        nova.ets[i].mecanico.marca = temp;
        //obtem nome do mecanico
        getline(ss1, temp, '|');
        nova.ets[i].mecanico.nome = temp;
        //obtem preço de reparacao por dia
        getline(ss1, temp, '|');
        nova.ets[i].mecanico.preco_reparacao_por_dia = stof(temp);

    }
    getline(ficheiro, linha);
    stringstream ss2(linha);
    //carrega fila de espera
    getline(ss2, temp, '|');
    nova.fila_espera_tamanho = stoi(temp);
    nova.fila_espera = new Carro[nova.fila_espera_tamanho];
    for (int y = 0; y < nova.fila_espera_tamanho; y++) {
        getline(ficheiro, linha);
        stringstream ss3(linha);
        //obtem id do carro em espera
        getline(ss3, temp, '|');
        nova.fila_espera[y].ID = stoi(temp);
        //obtem dias em reparacao do carro em espera
        getline(ss3, temp, '|');
        nova.fila_espera[y].dias_em_reparacao = stoi(temp);
        //obtem marca do carro
        getline(ss3, temp, '|');
        nova.fila_espera[y].marca = temp;
        //obtem modelo do carro
        getline(ss3, temp, '|');
        nova.fila_espera[y].modelo = temp;
        //obtem prioridade do carro em espera
        getline(ss3, temp, '|');
        nova.fila_espera[y].prioritario = stoi(temp);
        //obtem tempo de reparacao maximo do carro em espera
        getline(ss3, temp, '|');
        nova.fila_espera[y].tempo_reparacao_max = stoi(temp);
    }
    ficheiro.close();

    //liberta memoria
```

```
            getline(sss, temp, '|');
            nova.fila_espera[y].tempo_reparacao_max = stoi(temp);
        }
        ficheiro.close();

        //liberta memoria
        delete[] Of.fila_espera;
        for (int i = 0; i < Of.numero_ets; i++) {
            delete[] Of.ets[i].Carrosreparados;
            delete[] Of.ets[i].carros_a_ser_reparados;
        }
        delete[] Of.ets;
        //atribui a nova oficina
        Of = nova;


    }
    else {
        cout << "Erro: Ficheiro não encontrado!!" << endl << "usando valores locais" << endl;
    }

}
```

3.7. Para finalizar, esta parte do código representa um sistema de gerenciamento de uma oficina mecânica, onde é possível imprimir uma lista de carros ordenada alfabeticamente e por tempo de reparo. Ele usa duas funções para imprimir a lista de carros, sendo a primeira função imprimir_oficinaalfabeticamente() responsável por imprimir a lista de carros ordenada alfabeticamente e a segunda função imprimir_oficinaportempo() responsável por imprimir a lista de carros ordenada por tempo de reparo. Ambas estas funções irão primeiro criar uma array com todos os carros da oficina, incluindo os a ser reparados e os reparados de cada ET, mais os carros da fila de espera e, depois, recorrem às funções auxiliares, descritas abaixo, para realizar a ordenação e impressão.

Para ordenar os carros, a função OrdenarCarrosAlfabeticamenteEPorDiasReparacao() é usada na primeira função e ela é responsável por ordenar e imprimir os carros

pelo nome da marca, modelo, com o critério de desempate sendo o tempo de reparo, enquanto a função OrdenarCarrosPorDiasReparacao() é usada na segunda função e ela é responsável por ordenar e imprimir os carros pelo tempo de reparo.

No geral, o código aloca memória para um array de objetos Carro e copia todos os carros das diferentes entidades de trabalho (ETs) e da fila de espera para esse array. Em seguida, chama uma das funções de ordenação, dependendo da opção selecionada pelo usuário, e exibe a lista ordenada de carros.

```
void imprimir_oficinaalfabeticamente(Oficina& Of) {
    int num_carros_total = 0;
    for (int i = 0; i < Of.numero_ets; i++) {
        num_carros_total += Of.ets[i].num_carros_a_ser_reparados + Of.ets[i].num_carros_reparados;
    }
    num_carros_total += Of.fila_espera_tamanho;
    Carro* carros_total = new Carro[num_carros_total]; // aloca memória para o array de carros

    // Copia os carros de todas as ETs e da fila de espera para o array carros_total
    int k = 0;
    for (int i = 0; i < Of.numero_ets; i++) {
        for (int j = 0; j < Of.ets[i].num_carros_a_ser_reparados; j++) {
            carros_total[k] = Of.ets[i].carros_a_ser_reparados[j];
            k++;
        }
        for (int r = 0; r < Of.ets[i].num_carros_reparados; r++) {
            carros_total[k] = Of.ets[i].Carrosreparados[r];
            k++;
        }
    }
    for (int i = 0; i < Of.fila_espera_tamanho; i++) {
        carros_total[k] = Of.fila_espera[i];
        k++;
    }

    // Ordena o array de carros alfabeticamente e por dias_reparacao
    OrdenarCarrosAlfabeticamenteEPorDiasReparacao(carros_total, num_carros_total);


}
```

```cpp
void imprimir_oficinaportempo(Oficina& Of) {
    int num_carros_total = 0;
    for (int i = 0; i < Of.numero_ets; i++) {
        num_carros_total += Of.ets[i].num_carros_a_ser_reparados + Of.ets[i].num_carros_reparados;
    }
    num_carros_total += Of.fila_espera_tamanho;
    Carro* carros_total = new Carro[num_carros_total]; // aloca memória para o array de carros

    // Copia os carros de todas as ETs e da fila de espera para o array carros_total
    int k = 0;
    for (int i = 0; i < Of.numero_ets; i++) {
        for (int j = 0; j < Of.ets[i].num_carros_a_ser_reparados; j++) {
            carros_total[k] = Of.ets[i].carros_a_ser_reparados[j];
            k++;
        }
        for (int r = 0; r < Of.ets[i].num_carros_reparados; r++) {
            carros_total[k] = Of.ets[i].Carrosreparados[r];
            k++;
        }
    }
    for (int i = 0; i < Of.fila_espera_tamanho; i++) {
        carros_total[k] = Of.fila_espera[i];
        k++;
    }

    // Ordena o array de carros alfabeticamente e por dias_reparacao
    OrdenarCarrosPorDiasReparacao(carros_total, num_carros_total);


}
```

```cpp
void OrdenarCarrosAlfabeticamenteEPorDiasReparacao(Carro*& carros, int num_carros) {

    // Ordenar o array de "carros" alfabeticamente e por dias_em_reparacao
    for (int i = 0; i < num_carros - 1; i++) {
        for (int j = i + 1; j < num_carros; j++) {
            if (carros[i].marca > carros[j].marca ||
                (carros[i].marca == carros[j].marca && carros[i].modelo > carros[j].modelo) ||
                (carros[i].marca == carros[j].marca && carros[i].modelo == carros[j].modelo && carros[i].dias_em_reparacao > carros[j].dias_em_reparacao) ||
                (carros[i].marca == carros[j].marca && carros[i].modelo == carros[j].modelo && carros[i].dias_em_reparacao == carros[j].dias_em_reparacao && carros[i].prioritario < carros[j].prioritario)) {
                Carro temp = carros[i];
                carros[i] = carros[j];
                carros[j] = temp;
            }
        }
    }

    for (int k = 0; k < num_carros; k++)
    {
        cout << "ID: " << carros[k].ID << " | Marca: " << carros[k].marca << " | Modelo: " << carros[k].modelo << " | Tempo: " << carros[k].dias_em_reparacao << " | Prioritario: ";
            if (carros[k].prioritario == 1) {
                cout << "Sim" << endl;
            }
            else {
                cout << "Nao" << endl;
            }
    }
}
```

```cpp
void OrdenarCarrosPorDiasReparacao(Carro*& carros, int num_carros) {

    // Ordenar o array de "carros" alfabeticamente e por dias_em_reparacao
    for (int i = 0; i < num_carros - 1; i++) {
        for (int j = i + 1; j < num_carros; j++) {
            if (carros[i].dias_em_reparacao > carros[j].dias_em_reparacao) {
                Carro temp = carros[i];
                carros[i] = carros[j];
                carros[j] = temp;
            }
        }
    }

    for (int k = 0; k < num_carros; k++)
    {
        cout << "ID: " << carros[k].ID << " | Marca: " << carros[k].marca << " | Modelo: " << carros[k].modelo << " | Tempo: " << carros[k].dias_em_reparacao << " | Prioritario: ";
            if (carros[k].prioritario == 1) {
                cout << "Sim" << endl;
            }
            else {
                cout << "Nao" << endl;
            }
    }
}
```

Conclusão

Com este projeto, tivemos uma boa introdução à linguagem de C++, mais propriamente na manipulação de estruturas, arrays dinâmicos e pointers.

Achamos que concluímos o projeto com sucesso percorrendo as etapas e todos os pontos necessários para a conclusão do projeto e para o seu bom funcionamento. Durante o projeto, enfrentamos alguns erros nas fases mais iniciais do projeto, sendo que depois de resolvidos, o projeto ficou bem encaminhado e finalizado.

Durante a realização do projeto, um dos elementos do grupo não se manifestou para realizar o projeto, sendo este contactado em diversas instâncias pelo discord. Pedimo-lo para criar conta no GitHub pois seria mais fácil para colaborar na realização do projeto. Tentamos mais uma vez contactar este elemento a ver se conseguiria aparecer na universidade para realizar em conjunto o projeto, contudo esta colaboração sem sucesso.

Em suma, achamos que este projeto enriqueceu o nosso conhecimento e leque de competências. Por necessidade, achamos melhor usar o GitHub, que nos deu flexibilidade na partilha das soluções que cada membro contribui para a realização do projeto.

Divisão das tarefas de implementação pelos membros do grupo

Leonardo Ferreira:

- Reparação manual;
- Atualizar tempo de reparação;
- Adicionar prioridade;
- Remover mecânico;
- Gravar oficina;
- Carregar oficina;
- Imprimir oficina.

Renato Pêssego:

- Função (“Gestao”)
- Reparação;
- imprimir_oficinaportempo e OrdenarCarrosPorDiasReparacao;
- Verificarnumero;
- Criarmecanico;
- ColocarCarrosET;
- CriarCarrosnafila;
- Colocar prioritario;
- Transportar;
- seguinte;
- Maiuscula;
- removerespacos;

- Debug;

Paulo Alves:

- CarregarFicheirosLista;
- TamanhoFicheiro;
- Adiciona;
- Remove;
- CriarET;
- numero_ets;
- Gerarprobalidades;
- criarcarro(parte inicial);
- menuinfo;
- marcapresente;
- CriarCarrosNaFila;

Glossário

1. ETs = estações de trabalho;
2. array = vetor;
3. ciclos = cada ciclo corresponde a um dia decorrido na oficina