Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Unidade Curricular: Programação orientada aos objetos

Relatório Relativo ao Trabalho Prático

Tema: Gestão de ficheiros e diretorias

Realizado por: Artur Santos – 20251

Alexandre Moreira – 20223

Luís Anselmo – 20245

Viseu, 2020

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Relatório relativo ao Trabalho Prático

PROJETO FINAL POO

Curso de Licenciatura em Engenharia Informática

Unidade Curricular de Programação Orientada aos Objetos

Gestão de ficheiros e diretorias

Ano Letivo 2020/21

Realizado por: Artur Santos – 20251

Alexandre Moreira – 20223

Luís Anselmo – 20245

Viseu, 2020

ÍNDICE

Conteúdo

1. Introdução

O objetivo principal deste trabalho é mostrar e explicar o nosso projeto de gestão de diretorias, de acordo com o enunciado e instruções que nos foram atribuídas.

Ao longo do relatório serão abordados vários temas. Entre os quais estão:

- Desenvolvimento da estrutura do trabalho;
- Soluções encontradas para os problemas encontrados ao longo do trabalho;
- Explicação detalhada das diferentes partes do programa

Este programa foi desenvolvido na versão padrão *ISO* C++ 2020(stdc++20) da linguagem C++ lançada a 15 de dezembro de 2020, de modo a poder usar a biblioteca “filesystem” que passou a ser parte do *ISO* C++ na versão C++17.

2. Início do Programa

2.1 Header files

Header files são ficheiros onde é possivel declarar funções assim como defenir macros para serem utilizadas ao longo do programa.

Nas imagens seguintes podemos ver todos os header files utilizados ao longo deste programa:

Menu:

```
#pragma once
#include "libs.h"
#include "SistemaFicheiros.h"

class Menus
{
public:
    static int Menu();
    static int MenuSearch();
    static string NomeSearch(int Op);
    static int MenuRemover();
    static string RemoverDiretoria();
    static int MenuTree();
    static int MenuRenomear();
    static void Select(SistemaFicheiros* P);
};
```

Objeto geral:

```
#pragma once
#include "libs.h"

class Diretoria;
class ObjetoGeral
{
    struct tm* DataCriacao;
    struct stat attrib_file;
    string Nome;
    string Path;
public:
    ObjetoGeral(tm* _data, string _caminho, string _nome);
    string GetNome() { return Nome; };
    string GetPath() { return Path; };
    void AddTabulacoesXML(ofstream& Ficheiro, int Espacos);
    void EscreveElementoXML(ofstream& Ficheiro, int Espacos, string Elemento);
    void FechaElementoXML(ofstream& Ficheiro, int Espacos, string Elemento);
    tm* GetData(const string& NFich);
    void AtualizaPath(const string& NovoPath) { Path = NovoPath; };
    void Renomear(const string& NomeNovo) { Nome = NomeNovo; };
    virtual ~ObjetoGeral();
};
```

Diretoria:

```
#pragma once
#include "ObjetoGeral.h"
#include "Ficheiro.h"

class Diretoria :
    public ObjetoGeral
{
    list<Diretoria*> LDir;
    list<Ficheiro*> LFich;

public:
    Diretoria(tm* _data, string _caminho, string _nome);
    void AddFich(Ficheiro* Fich) { LFich.push_back(Fich); };
    void AddDir(Diretoria* Dir) { LDir.push_back(Dir); };
    void ContarFicheiros(int* NumFich);
    void ContarDiretorias(int* NumDir);
    int GetNumFich() { return LFich.size(); };
    int GetNumDir() { return LDir.size(); };
    void Memoria(int* MemoriaTotal);
    int GetTamanhoDir();
    int GetNumElem() { return LFich.size() + LDir.size(); };
    void DirMaisElementos(int* NumElementos, string* DirPath, string* DirName);
    void DirMaisElementosRoot(int* NumElementos, string* DirPath, string* DirName);
    void DirMenosElementos(int* NumElementos, string* DirPath, string* DirName);
    void DirMenosElementosRoot(int* NumElementos, string* DirPath, string* DirName);
    void FichMaior(int* Tamanho, string* FichPath, string* NomePath);
    void DiretoriaMaior(int* Tamanho, string* DirPath, string* NomeDir);
    void DiretoriaMaiorRoot(int* Tamanho, string* DirPath, string* NomeDir);
    void SearchFicheiros(const string& FileName, string& Caminho);
    void Search(const string& DirName, string& Caminho);
    bool RemoveDiretoria(const string& DirName);
    bool RemoveFicheiros();
    void EscreverXML(ofstream& File, int Espacos);
    Diretoria* GetPonteiroDirFicheiro(const string& NomeFicheiro, Ficheiro* F);
    bool MoveFicheiro(const string& Fich, string DirAntiga, string DirNova);
    bool MoverDirectoria(const string& DirOld, const string& DirNew);
    tm* DataFicheiro(const string& NomeFich);
    void PesquisarAllDiretorias(list<string>& LNomes, const string& NomeDir);
    void PesquisarAllDiretoriasRoot(list<string>& LNomes, const string& NomeDir);
    void PesquisarAllFicheiros(list<string>& LNomes, const string& NomeFicheiro);
    void RenomearFicheiros(const string& NFich, const string& NNovo);
    void RenomearFicheirosRoot(const string& NFich, const string& NNovo);
    bool VerificaNomes(list<string>& LNomes, string NomeVerificar);
    bool FicheirosDuplicados(list<string>& LNomes);
    virtual ~Diretoria();
};
```

Ficheiro:

```h
#pragma once
#include "ObjetoGeral.h"

class Ficheiro :
    public ObjetoGeral
{
    int Tamanho;

public:
    Ficheiro(tm* _data, int _tamanho, string _caminho, string _nome);
    int GetTamanho() { return Tamanho; };
    void EscreverXML(ofstream& File, int Espacos);
    virtual ~Ficheiro();
};
```

Sistema de Fisheiros:

```h
#pragma once
#include "libs.h"
#include "Diretoria.h"
#include "Ficheiro.h"

class SistemaFicheiros
{
    Diretoria* Raiz = NULL; //Enquanto nao for especificada, a Diretoria Raiz sera NULL
    string Path;

    public:
        SistemaFicheiros();
        bool LoadRoot(const string& path);
        bool Load(const string& path, Diretoria* Dir);
        int ContarFicheiros();
        int ContarDirectorias();
        int Memoria();
        string DiretoriaMaisElementos();
        string DiretoriaMenosElementos();
        string FicheiroMaior();
        string DiretoriaMaisEspaco();
        string Search(const string& s, int Tipo);
        bool RemoverAll(const string& s, const string& tipo);
        void Escrever_XML(const string& s);
        bool Ler_XML(const string& s);
        bool MoveFicheiro(const string& Fich, string DirAntiga, string DirNova);
        bool MoverDirectoria(const string& DirOld, const string& DirNew);
        string DataFicheiro(const string& ficheiro);
        void PesquisarAllDirectorias(list<string>& lres, const string& dir);
        void PesquisarAllFicheiros(list<string>& lres, const string& file);
        void RenomearFicheiros(const string& fich_old, const string& fich_new);
        bool VerificarExistenciaFicheiro(const string& NFich);
        bool VerificarExistenciaDiretoria(const string& NDir);
        bool FicheirosDuplicados();
        bool CopyBatch(const string& padrao, const string& DirOrigem, const string& DirDestino);
        virtual ~SistemaFicheiros();
};
```

Libs:

```cpp
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
#include <filesystem>   //Versão C++17 +
#include <cstdio>
#include <list>
#include <string>
#include <dirent.h>
#include <direct.h>
#include <limits.h>
#include <locale.h>
#include <sys/stat.h>
#include <time.h>

using namespace std;
namespace fs = std::filesystem;
```

3. Objeto geral

Nesta primeira função de “objeto geral” podemos verificar que a mesma tem a função de Class “Mãe”, ou seja, todos os outros ficheiros headers utilizam e acedem aos dados presentes neste mesmo ficheiro header, daí o seu nome. É partilhada informação como:

- Nomes e paths relativos a ficheiros/diretorias;
- Data de modificação de um ficheiro;

3.1 Construtor e destrutor:

```
#include "ObjetoGeral.h"

ObjetoGeral::ObjetoGeral(tm* _data, string _caminho, string _nome)
{
    DataCriacao = _data;
    Path = _caminho;
    Nome = _nome;
}

ObjetoGeral::~ObjetoGeral()
{
    //dtor
}
```

3.2 AddTabulaçõesXML:

```
void ObjetoGeral::AddTabulacoesXML(ofstream& Ficheiro, int Espacos)
{
    for (int i = 0; i < Espacos; ++i)
        Ficheiro << "\t";
}
```

Esta função tem como objetivo formar a tabulação do ficheiro XML que irá ser criado através da mesma

3.3 EscreverElementoXML:

```
void ObjetoGeral::EscreveElementoXML(ofstream& Ficheiro, int Espacos, string Elemento)
{
    AddTabulacoesXML(Ficheiro, Espacos);
    Ficheiro << "<" << Elemento << ">";
}
```

A função a cima tem o objetivo, inicializar cada um dos elementos e escreve-los no ficheiro XML

3.4 FechaElementoXML:

```
void ObjetoGeral::FechaElementoXML(ofstream& Ficheiro, int Espacos, string Elemento)
{
    AddTabulacoesXML(Ficheiro, Espacos);
    Ficheiro << "</" << Elemento << ">\n";
}
```

À semelhança da função anteriror, esta função tem como objetivo terminar os elementos

3.5 Função GetData:

```
tm* ObjetoGeral::GetData(const string& NFich)
{
    stat(NFich.c_str(), &attrib_file);
    DataCriacao = gmtime(&(attrib_file.st_mtime));
    return DataCriacao;
}
```

A função retorna a data de modificação de um determinado ficheiro dado o seu nome

4. Ficheiro

4.1 Função Construtor e destrutor:

```
#include "Ficheiro.h"

Ficheiro::Ficheiro(tm* _data, int _tamanho, string _caminho, string _nome)
    : ObjetoGeral(_data, _caminho, _nome)
{
    Tamanho = _tamanho;
}

Ficheiro::~Ficheiro()
{
    //dtor
}
```

4.2 Função Escreve XML

```
void Ficheiro::EscreverXML(ofstream& File, int Espacos)
{
    EscreveElementoXML(File, Espacos, "Ficheiro");
    File << "\n";

    EscreveElementoXML(File, Espacos + 1, "Nome");
    File << this->GetNome();
    FechaElementoXML(File, 0, "Nome");

    EscreveElementoXML(File, Espacos + 1, "Path");
    File << this->GetPath();
    FechaElementoXML(File, 0, "Path");

    EscreveElementoXML(File, Espacos + 1, "Tamanho");
    File << this->GetTamanho();
    FechaElementoXML(File, 0, "Tamanho");

    FechaElementoXML(File, Espacos, "Ficheiro");
}
```

Na função seguinte podemos ver a formatação de texto final da função falada anteriormente

5. Menus

Nas próximas é possível ver o menu principal, assim como todos os submenus.

```cpp
#include "libs.h"
#include "Menus.h"

int Menus::Menu()
{
    int opcao, opcao_invalida = 0;
    do
    {
        system("cls");
        cout << "#----------------------------------------------MENU----------------------------------------------#" << endl;
        cout << "|  1- Carregar a diretoria                                                                        |" << endl;
        cout << "|  2- Contar o numero de ficheiros guardados em memoria                                           |" << endl;
        cout << "|  3- Contar o numero de diretorias guardadas em memoria                                          |" << endl;
        cout << "|  4- Determinar toda a memoria ocupada                                                           |" << endl;
        cout << "|  5- Determinar qual a diretoria que tem mais elementos                                          |" << endl;
        cout << "|  6- Determinar qual a diretoria que tem menos elementos                                         |" << endl;
        cout << "|  7- Determinar qual o ficheiro que ocupa mais espaco                                            |" << endl;
        cout << "|  8- Determinar qual a diretoria que ocupa mais espaco                                           |" << endl;
        cout << "|  9- Pesquisar ficheiro ou diretoria                                                             |" << endl;
        cout << "| 10- Remover ficheiro ou diretoria                                                               |" << endl;
        cout << "| 11- Gravar os ficheiros em formato XML                                                          |" << endl;
        cout << "| 12- Ler XML (apaga todo o Sistema de Ficheiros criado anteriormente)                            |" << endl;
        cout << "| 13- Mover um ficheiro para outra diretoria                                                      |" << endl;
        cout << "| 14- Mover uma diretoria para outra diretoria (move ficheiros e sub-diretorias)                  |" << endl;
        cout << "| 15- Inserir o nome de um ficheiro e receber a data de modificacao                               |" << endl;
        cout << "| 16- Tree                                                                                        |" << endl;
        cout << "| 17- Pesquisar todas as diretorias                                                               |" << endl;
        cout << "| 18- Pesquisar todos os ficheiros                                                                |" << endl;
        cout << "| 19- Renomear ficheiros                                                                          |" << endl;
        cout << "| 20- Verificar se exitem ficheiros duplicados (com o mesmo nome)                                 |" << endl;
        cout << "| 21- Copiar os ficheiros de uma diretoria para outra, definida pelo utilizador                   |" << endl;
        cout << "| 0 - Sair                                                                                        |" << endl;
        cout << "#-------------------------------------------------------------------------------------------------#" << endl;
        cin >> opcao;
        opcao_invalida = ((opcao > 21) && (opcao < 0));

        if (opcao_invalida)
        {
            cout << "Opcao nao disponivel! Insira uma opcao valida!" << endl;
            system("pause");
        }
        fflush(stdin);
    } while (opcao_invalida);

    return opcao;
}
```

Função Menu Principal

```
#-----------------------------------------------------MENU-----------------------------------------------------#
|   1- Carregar a diretoria                                                                                     |
|   2- Contar o numero de ficheiros guardados em memoria                                                        |
|   3- Contar o numero de diretorias guardadas em memoria                                                       |
|   4- Determinar toda a memoria ocupada                                                                        |
|   5- Determinar qual a diretoria que tem mais elementos                                                       |
|   6- Determinar qual a diretoria que tem menos elementos                                                      |
|   7- Determinar qual o ficheiro que ocupa mais espaco                                                         |
|   8- Determinar qual a diretoria que ocupa mais espaco                                                        |
|   9- Pesquisar ficheiro ou diretoria                                                                          |
|  10- Remover todos os ficheiros ou todas as diretorias                                                        |
|  11- Gravar os ficheiros em formato XML                                                                       |
|  12- Ler XML (apaga todo o Sistema de Ficheiros criado anteriormente)                                         |
|  13- Mover um ficheiro para outra diretoria                                                                   |
|  14- Mover uma diretoria para outra diretoria (implica mover tudo o que esteja dentro dessa diretoria)        |
|  15- Inserir o nome de um ficheiro e receber a data de modificacao                                            |
|  16- Tree                                                                                                     |
|  17- Pesquisar diretorias                                                                                     |
|  18- Pesquisar ficheiros                                                                                      |
|  19- Renomear ficheiros                                                                                       |
|  20- Verificar se exitem ficheiros duplicados (com o mesmo nome)                                              |
|  21- Copiar os ficheiros de uma diretoria para outra, definida pelo utilizador                                |
|  0 - Sair                                                                                                     |
#-----------------------------------------------------------------------------------------------------------------#
```

Menu Principal

```cpp
int Menus::MenuSearch()
{
    int opcao, opcao_invalida;
    do
    {
        system("cls");
        cout << "#-----------------------------------------------------MENU-----------------------------------------------------#" << endl;
        cout << "|  0 - Procurar um ficheiro                                                                                    |" << endl;
        cout << "|  1 - Procurar uma diretoria                                                                                  |" << endl;
        cout << "#--------------------------------------------------------------------------------------------------------------#" << endl;
        cin >> opcao;
        opcao_invalida = !((opcao == 1) || (opcao == 0));

        if (opcao_invalida)
        {
            cout << "Opcao invalida! Insira uma das opcoes apresentadas!" << endl;
            system("pause");
        }
        fflush(stdin);
    } while (opcao_invalida);

    return opcao;
}
```

```cpp
string Menus::NomeSearch(int Op)
{
    string Nome;

    if (Op == 0)
        cout << "Qual o nome do ficheiro a pesquisar?" << endl;
    else
        cout << "Qual o nome da diretoria a pesquisar?" << endl;

    cin >> Nome;
    return Nome;
}
```

Funões Menus Search

```cpp
int Menus::MenuRemover()
{
    int opcao, opcao_invalida;
    do
    {
        system("cls");
        cout << "#----------------------------------------------------MENU----------------------------------------------------#" << endl;
        cout << "|  1 - Remover todos os Ficheiros de uma certa diretoria                                                     |" << endl;
        cout << "|  2 - Remover uma Diretoria (incluido as sub-diretorias e os respetivos ficheiros)                          |" << endl;
        cout << "|  0 - Voltar atras                                                                                          |" << endl;
        cout << "#------------------------------------------------------------------------------------------------------------#" << endl;
        cin >> opcao;
        opcao_invalida = !((opcao == 1) || (opcao == 0) || (opcao == 2));

        if (opcao_invalida)
        {
            cout << "Opcao invalida! Insira uma das opcoes apresentadas!" << endl;
            system("pause");
        }
        fflush(stdin);
    } while (opcao_invalida);
    return opcao;
}
```

```cpp
string Menus::RemoverDiretoria()
{
    string Nome;
    cout << "Qual o nome da diretoria que pretende eliminar?" << endl;
    cin >> Nome;
    system("cls");
    return Nome;
}
```

Função Menu Remover

```
#----------------------------------------------------MENU----------------------------------------------------#
|  0 - Remover todos os Ficheiros                                                                            |
|  1 - Remover uma Diretoria (incluido as sub-diretorias e os respetivos ficheiros                           |
#------------------------------------------------------------------------------------------------------------#
```

Menu Remover

```cpp
int Menus::MenuTree()
{
    int opcao, opcao_invalida;
    do
    {
        system("cls");
        cout << "#----------------------------------------------------MENU----------------------------------------------------#" << endl;
        cout << "|  0 - Mostrar Tree num ficheiro                                                                             |" << endl;
        cout << "|  1 - Mostrar Tree no ecra                                                                                  |" << endl;
        cout << "#------------------------------------------------------------------------------------------------------------#" << endl;
        cin >> opcao;
        opcao_invalida = !((opcao == 1) || (opcao == 0));

        if (opcao_invalida)
        {
            cout << "Opcao invalida! Insira uma das opcoes apresentadas!" << endl;
            system("pause");
        }
        fflush(stdin);
    } while (opcao_invalida);
    system("cls");
    return opcao;
}
```

Função Menu Tree

```
#----------------------------------------------------MENU----------------------------------------------------#
|  0 - Mostrar Tree num ficheiro                                                                             |
|  1 - Mostrar Tree no ecra                                                                                  |
#------------------------------------------------------------------------------------------------------------#
```

Menu Tree

```
void Menus::Select(SistemaFicheiros* P)
{
    int Opcao, OpAux, aux_loaded = 0;
    char buff[PATH_MAX];
    _getcwd(buff, PATH_MAX);    //faz o catch da diretoria atual do programa
    string current_dir(buff);
    string Str, StrAux, confirmar, del_dir, del_allfich, old_filename, new_filename, old_dirname, new_dirname;
    list<string> LResDir;
    list<string> LResFich;

    do
    {
        system("cls");
        Opcao = Menu();
        switch (Opcao)
        {
            do
            {
        case 1:
            if (!P->LoadRoot(current_dir))  //faz load da directoria onde o programa se encontra
            {
                cout << endl << "Nao foi possivel carregar a diretoria!" << endl;
                cout << endl << "Tente novamente por favor." << endl;
                system("pause");
            }
            else
            {
                cout << endl << "Diretoria carregada com exito!" << endl;
                cout << endl << "Foi carregada a seguinte diretoria: " << current_dir << endl << endl;
                aux_loaded = 1;
                system("pause");
            }
            break;

        case 2:
            if (aux_loaded == 0)
            {
                cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
                system("pause");
            }
            else
            {
                cout << endl << "Numero de ficheiros guardados em memoria: " << P->ContarFicheiros() << endl;
                system("pause");
            }
            break;
        case 3:
            if (aux_loaded == 0)
            {
                cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
                system("pause");
            }
            else
            {
                cout << endl << "Numero de diretorias guardadas em memoria: " << P->ContarDirectorias() << endl;
                system("pause");
            }
            break;
```

```
case 4:
    if (aux_loaded == 0)
    {
        cout << "ERRO! Não foi possível carregar a directoria" << endl;
        system("pause");
    }
    else
    {
        cout << "Estao alocados " << P->Memoria() << " bytes na memoria!" << endl;
        system("pause");
    }
    break;

case 5:
    if (aux_loaded == 0)
    {
        cout << "ERRO! Não foi possível carregar a directoria" << endl;
        system("pause");
    }
    else
    {
        cout << "A diretoria com mais elementos e: " << P->DiretoriaMaisElementos() << endl;
        system("pause");
    }
    break;
case 6:
    if (aux_loaded == 0)
    {
        cout << "ERRO! Não foi possível carregar a directoria" << endl;
        system("pause");
    }
    else
    {
        cout << "A diretoria com menos elementos e: " << P->DiretoriaMenosElementos() << endl;
        system("pause");
    }
    break;

case 7:
    if (aux_loaded == 0)
    {
        cout << "ERRO! Não foi possível carregar a directoria" << endl;
        system("pause");
    }
    else
    {
        cout << "O ficheiro maior e: " << P->FicheiroMaior() << endl;
        system("pause");
    }
    break;
```

```
        case 8:
            if (aux_loaded == 0)
            {
                cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
                system("pause");
            }
            else
            {
                cout << endl << "Diretoria que ocupa mais espaco: " << P->DiretoriaMaisEspaco() << endl;
                system("pause");
            }
            break;

        case 9:
            if (aux_loaded == 0)
            {
                cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
                system("pause");
            }
            else
            {
                OpAux = MenuSearch();
                if (OpAux == 0)
                {
                    break;
                }
                Str = NomeSearch(OpAux);
                cout << "Resultado da procura: " << P->Search(Str, OpAux) << endl;
                system("pause");
            }
            break;
```

```
        case 10:
            if (aux_loaded == 0)
            {
                cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
                system("pause");
            }
            else
            {
                OpAux = MenuRemover();
                if (OpAux == 2)
                {
                    del_dir = RemoverDiretoria();
                    P->RemoverAll(del_dir, "DIR");
                    P->LoadRoot(current_dir);   //atualiza a diretoria em memoria
                    system("pause");
                }
                else if (OpAux == 1)
                {
                    del_allfich = RemoverFicheiros();
                    if (P->RemoverAll(del_allfich, "Fich"))
                        cout << endl << "Ficheiros eliminados com sucesso!" << endl;
                    P->LoadRoot(current_dir);   //atualiza a diretoria em memoria
                    system("pause");
                }
                else
                    break;
            }
            break;

        case 11:
            if (aux_loaded == 0)
            {
                cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
                system("pause");
            }
            else
            {
                cout << endl << "Qual o nome do ficheiro para o qual pretende guardar o conteudo em XML?" << endl;
                cin >> Str;
                P->Escrever_XML(Str);
                cout << endl << "Operacao concluida com sucesso!" << endl;
                system("pause");
            }
            break;
```

```
        case 12:
            //Ler_XML();
            system("pause");
            break;
```

```
    case 13:
        if (aux_loaded == 0)
        {
            cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
            system("pause");
        }
        else
        {
            system("cls");
            cout << endl << "Qual o nome do ficheiro que pretende mover?" << endl;
            cin >> old_filename;
            cin.ignore();
            if (P->VerificarExistenciaFicheiro(old_filename))   //Verifica se o ficheiro especificado pelo utilizador existe na diretoria carregada em memória ("current_dir" no caso)
            {
                cout << endl << "A mover o ficheiro: <" << old_filename << "> da diretoria: " << current_dir << endl;
                cout << endl << "Qual o caminho da diretoria para a qual pretende mover?" << endl;
                cin >> new_dirname;
                if (P->VerificarExistenciaDiretoria(new_dirname))       //Verificar se existe a diretoria especificada pelo utilizador
                {
                    cout << endl << "Deseja fazer a seguinte operacao? [Sin/Nao] :" << endl << "Mover o ficheiro <" << old_filename << "> da diretoria " << current_dir << "para a diretoria " << new_dirname << endl;
                    cin >> confirmar;
                    if (confirmar == "s" || confirmar == "S" || confirmar == "sim" || confirmar == "Sim" || confirmar == "SIM" || confirmar == "y" || confirmar == "Y" || confirmar == "yes" || confirmar == "Yes" || confirmar == "YES")
                    {
                        if (P->MoveFicheiro(old_filename, current_dir, new_dirname))   //move o ficheiro indicado pelo utilizar da diretoria "old_dirname" para a diretoria "new_dirname"
                        {
                            cout << endl << "Operacao concluida com sucesso" << endl << endl;
                            cout << endl << "Mostrar a tree? [Sin/Nao] :" << endl;  //(extra) Mostra a tree para verificar mudanças
                            cin >> confirmar;
                            if (confirmar == "s" || confirmar == "S" || confirmar == "sim" || confirmar == "Sim" || confirmar == "SIM" || confirmar == "y" || confirmar == "Y" || confirmar == "yes" || confirmar == "Yes" || confirmar == "YES")
                            {
                                cout << endl << endl;
                                system("tree /f");
                            }
                        }
                        system("pause");
                    }
                    else
                    {
                        cout << "Operacao cancelada" << endl << endl << "A sair....." << endl;
                        system("pause");
                        break;
                    }
                }
                P->LoadRoot(current_dir);   //atualiza a diretoria em memoria
            }
```

```
            else if (!P->VerificarExistenciaFicheiro(old_filename))  //Caso o ficheiro especificado pelo utilizador não exista na diretoria carregada em memória ("current_dir" no caso)
            {
                cout << endl << "O ficheiro que esta a mover nao se encontra na mesma diretoria do programa" << endl;
                cout << "Por favor indique o caminho da diretoria do ficheiro que pretende mover:" << endl; //O utilizador tem de especificar para que diretoria pretende mover o ficheiro
                cin >> old_dirname;
                if (P->VerificarExistenciaDiretoria(old_dirname))   //Verificar se existe a diretoria especificada pelo utilizador
                {
                    cout << endl << "A mover o ficheiro: <" << old_filename << "> da diretoria: " << old_dirname << endl;
                    cout << endl << "Qual o caminho da diretoria para a qual pretende mover?" << endl;
                    cin >> new_dirname;
                    if (P->VerificarExistenciaDiretoria(new_dirname))   //Verificar se existe a diretoria especificada pelo utilizador
                    {
                        cout << endl << "Deseja fazer a seguinte operacao? [Sim/Nao] :" << endl << "Mover o ficheiro <" << old_filename << "> da diretoria " << old_dirname << "para a diretoria " << new_dirname << endl;
                        cin >> confirmar;
                        if (confirmar == "s" || confirmar == "S" || confirmar == "sim" || confirmar == "Sim" || confirmar == "SIM" || confirmar == "y" || confirmar == "Y" || confirmar == "yes" || confirmar == "Yes" || confirmar == "YES")
                        {
                            if (P->MoveFicheiro(old_filename, old_dirname, new_dirname))   //move o ficheiro indicado pelo utilizador da diretoria "old_dirname" para a diretoria "new_dirname"
                            {
                                cout << endl << "Operacao concluida com sucesso" << endl << endl;
                                cout << endl << "Mostrar a tree? [Sim/Nao] :" << endl;  //(extra) Mostra a tree para verificar mudanças
                                cin >> confirmar;
                                if (confirmar == "s" || confirmar == "S" || confirmar == "sim" || confirmar == "Sim" || confirmar == "SIM" || confirmar == "y" || confirmar == "Y" || confirmar == "yes" || confirmar == "Yes" || confirmar == "YES")
                                {
                                    cout << endl << endl;
                                    system("tree /f");
                                }
                                system("pause");
                            }
                            else
                            {
                                cout << "Operacao cancelada" << endl << endl << "A sair......." << endl;
                                system("pause");
                                break;
                            }
                        }
                    }
                    P->LoadRoot(current_dir);   //atualiza a diretoria em memoria
                }
            }
            break;

        case 14:
            if (aux_loaded == 0)
            {
                cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
                system("pause");
            }
            else
            {
                system("cls");
                cout << endl << "Indique o caminho da diretoria que pretende mover" << endl;
                cin >> old_dirname;
                cin.ignore();
                if (P->VerificarExistenciaDiretoria(old_dirname))   //Verificar se existe a diretoria especificada pelo utilizador
                {
                    cout << endl << "A mover a diretoria: " << old_dirname << endl;
                    cout << endl << "Indique o novo caminho para onde pretende mover a diretoria" << endl;
                    cin >> new_dirname;
                    if (P->VerificarExistenciaDiretoria(new_dirname))   //Verificar se existe a diretoria especificada pelo utilizador
                    {
                        cout << endl << "Deseja fazer a seguinte operacao? [Sim/Nao] :" << endl << "Mover a diretoria " << old_dirname << " para a diretoria " << new_dirname << endl;
                        cin >> confirmar;
                        if (confirmar == "s" || confirmar == "S" || confirmar == "sim" || confirmar == "Sim" || confirmar == "SIM" || confirmar == "y" || confirmar == "Y" || confirmar == "yes" || confirmar == "Yes" || confirmar == "YES")
                        {
                            if (P->MoverDirectoria(old_dirname, new_dirname))   //Move a diretoria indicada pelo utilizador "old_dirname" para uma outra diretoria "new_dirname"
                            {
                                cout << endl << "Operacao concluida com sucesso" << endl << endl;
                                cout << endl << "Mostrar a tree? [Sim/Nao] :" << endl;  //(extra) Mostra a tree para verificar mudanças
                                cin >> confirmar;
                                if (confirmar == "s" || confirmar == "S" || confirmar == "sim" || confirmar == "Sim" || confirmar == "SIM" || confirmar == "y" || confirmar == "Y" || confirmar == "yes" || confirmar == "Yes" || confirmar == "YES")
                                {
                                    cout << endl << endl;
                                    system("tree /f");
                                }
                            }
                            system("pause");
                        }
                        else
                        {
                            cout << "Operacao cancelada" << endl << endl << "A sair......." << endl;
                            system("pause");
                            break;
                        }
                    }
                    P->LoadRoot(current_dir);   //atualiza a diretoria em memoria
                }
            }
            break;
```

```
    case 15:
        if (aux_loaded == 0)
        {
            cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
            system("pause");
        }
        else
        {
            cout << endl << "Qual o nome do ficheiro do qual pretende saber a data?" << endl;
            cin >> StrAux;
            cout << P->DataFicheiro(StrAux) << endl;
            system("pause");
        }
        break;

    case 16:
        if (aux_loaded == 0)
        {
            cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
            system("pause");
        }
        else
        {
            OpAux = MenuTree();
            if (OpAux == 1)
            {
                P->Tree();
                system("pause");
            }
            else if (OpAux == 2)
            {
                system("tree /f /a > tree.txt");    //guarda a tree do path atual para um ficheiro (tree.txt)
                cout << endl << "Tree guardada no ficheiro: <tree.txt>" << endl << endl;
                system("pause");
            }
            else
                break;
        }
        break;
```

```
    case 17:
        if (aux_loaded == 0)
        {
            cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
            system("pause");
        }
        else
        {
            cout << endl << "Qual o nome da(s) diretoria(s) a pesquisar?" << endl;
            cin >> Str;
            P->PesquisarAllDirectorias(LResDir, Str);
            cout << "Resultados obtidos:" << endl;
            for (list<string>::iterator it = LResDir.begin(); it != LResDir.end(); ++it)
                cout << (*it) << endl;
            LResDir.clear();
            system("pause");
        }
        break;

    case 18:
        if (aux_loaded == 0)
        {
            cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
            system("pause");
        }
        else
        {
            cout << endl << "Qual o nome do(s) ficheiro(s) a pesquisar?" << endl;
            cin >> Str;
            P->PesquisarAllFicheiros(LResFich, Str);
            cout << "Resultados obtidos:" << endl;
            for (list<string>::iterator it = LResFich.begin(); it != LResFich.end(); ++it)
                cout << (*it) << endl;
            LResFich.clear();
            system("pause");
        }
        break;
```

```
    case 19:
        if (aux_loaded == 0)
        {
            cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
            system("pause");
        }
        else
        {
            cout << endl << "Qual o nome do(s) ficheiro(s) a renomear?" << endl;
            cin.ignore();
            cin >> old_filename;
            if (P->VerificarExistenciaFicheiro(old_filename))
            {
                cout << "Qual o novo nome a inserir?" << endl;
                cin >> new_filename;
                P->RenomearFicheiros(old_filename, new_filename);   //renomeia em "memória" (para efeitos de pesquisa através do uso das listas)
                rename(old_filename.c_str(), new_filename.c_str()); //renomeia o ficheiro "fisicamente"
                P->LoadRoot(current_dir);
                system("pause");
            }
            else
            {
                cout << endl << "[ERRO!!]: Nao foi possivel renomear o ficheiro <" << old_filename << ">. O ficheiro nao existe ou pertence a outra diretoria." << endl;
                cout << endl << "Tente novamente por favor." << endl;
                system("pause");
                break;
            }
        }
        break;

    case 20:
        if (aux_loaded == 0)
        {
            cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
            system("pause");
        }
        else
        {
            if (P->FicheirosDuplicados())
                cout << endl << "Existem ficheiros duplicados!" << endl;
            else
                cout << endl << "Nao existem ficheiros duplicados!" << endl;
            system("pause");
        }
        break;
case 21:
    if (aux_loaded == 0)
    {
        cout << endl << "ERRO! Não foi possível carregar a directoria" << endl;
        system("pause");
    }
    else
    {
        system("cls");
        cout << endl << "Indique uma diretoria a copiar" << endl;
        cin >> old_dirname;
        cin.ignore();
        if (P->VerificarExistenciaDiretoria(old_dirname))   //Verificar se existe a diretoria especificada pelo utilizador
        {
            cout << endl << "A copiar a diretoria: " << old_dirname << endl;
            cout << endl << "Indique o novo caminho para onde pretende copiar a diretoria" << endl;
            cin >> new_dirname;
            if (P->VerificarExistenciaDiretoria(new_dirname))   //Verificar se existe a diretoria especificada pelo utilizador
            {
                cout << endl << "Deseja fazer a seguinte operacao? [Sim/Nao] :" << endl << "Copiar todos os ficheiros (e respetiva estrutura de sub-directorias) da diretoria " << old_dirname << " para a diretoria " << new_dirname << endl;
                cin >> confirmar;
                if (confirmar == "s" || confirmar == "S" || confirmar == "sim" || confirmar == "Sim" || confirmar == "SIM" || confirmar == "y" || confirmar == "Y" || confirmar == "yes" || confirmar == "Yes" || confirmar == "YES")
                {
                    P->CopyBatch("*.*", old_dirname, new_dirname);   //Copia todos os ficheiros(e respetiva estrutura de sub - directorias) da diretoria "old_dirname" para a diretoria "new_dirname"
                    cout << endl << "Operacao concluida com sucesso" << endl << endl;
                    cout << endl << "Mostrar a tree? [Sim/Nao] :" << endl;  //(extra) Mostra a tree para verificar mudanças
                    cin >> confirmar;
                    if (confirmar == "s" || confirmar == "S" || confirmar == "sim" || confirmar == "Sim" || confirmar == "SIM" || confirmar == "y" || confirmar == "Y" || confirmar == "yes" || confirmar == "Yes" || confirmar == "YES")
                    {
                        cout << endl << endl;
                        system("tree /f");
                    }
                    system("pause");
                }
                else
                {
                    cout << "Operacao cancelada" << endl << endl << "A sair......." << endl;
                    system("pause");
                    break;
                }
            }
            P->LoadRoot(current_dir);   //atualiza a diretoria em memoria
        }
    }
    break;
```

Nas prints acima apresentamos a função select e todos os seus componentes, mas gostariamos de aprofundar algumas dessas componentes:

“case 13”: Esta função, tal como pedido no enunciado, move um ficheiro para outra diretoria. A solução que decidimos adotar passa pelo uso de funções da biblioteca “filesystem”. Inicialmente fazemos uma verificação para confirmar que a directoria foi carregada (Linha 348). De seguida, se isto se verificar é pedido ao utilizador para especificar o nome do ficheiro a mover. Nesta fase pode haver 2 outputs:

- Se o ficheiro existir na mesma diretoria da diretoria onde o programa se encontra (diretoria carregada em memoria), este continua e é pedido ao utilizador para que directoria o pretende mover. Caso todas as condições tenham sido cumpridas, o ficheiro é movido para a directoria destino.

- Se o ficheiro não existir na mesma diretoria da diretoria onde o programa se encontra (diretoria carregada em memoria), o utilizador depara-se com uma ”mensagem de erro” pedindo para o mesmo especificar o caminho onde o ficheiro que pretende mover se encontra. Caso exista o ficheiro na diretoria que o utilizador especificou, o programa continua e, a semelhança do output anterior, é pedido ao utilizador a diretoria para onde pretende mover o ficheiro.

Adicionalmente apo o ficheiro ter sido movido é dada a opção ao utilizador de ver a tree da diretoria carregada em memoria para confirmar que o ficheiro foi movido.
Finalmente, no final da função, para efeitos de pesquisa, é novamente dado o load em memoria da diretoria onde o programa se encontra.

“case 14”: Esta função usa a mesma logica da função”case 13”, mas em vez de mover um ficheiro, move uma diretoria e todo o seu conteudo adjacente (ficheiros e subdiretorias).

“case 21”: Em semelhança as funções anteriores, esta usa a mesma logica. Em vez de mover um ficheiro ou diretoria, faz a copia integral de uma diretoria para outra.

Nota: No “case 21” por dificuldade não se conseguiu verificar se na diretoria destino já existiem ficheiros com nomes iguais a ficheiros da diretoria origem. Em alternativa os ficheiros da diretoria origem com o nome igual a ficheiros da diretoria destino são ignorados, não fazendo assim a cópia dos mesmos.

6. Sistema ficheiros

6.1 Função LoadRoot

```
bool SistemaFicheiros::LoadRoot(const string& path)
{
    bool Resultado = false;
    DIR* fich = NULL;
    if (((fich = opendir(path.c_str())) != NULL))
    {
        struct dirent* abrir = readdir(fich);
        this->Path = path;
        Diretoria* D = new Diretoria(NULL, path, abrir->d_name);
        this->Raiz = D;

        if (!Load(path, D))
            Resultado = false;
        else
            Resultado = true;

        closedir(fich);
    }
    return Resultado;
}
```

Esta função é responsável pelo carregamento da diretoria de raiz do programa

6.2 Função Load

```
bool SistemaFicheiros::Load(const string& path, Diretoria* Dir)
{
    bool Resultado = false;
    DIR* fich = NULL;
    if (((fich = opendir(path.c_str())) != NULL))
    {
        struct tm* DataCriacao;
        struct stat attrib_file;
        struct dirent* abrir;
        while ((abrir = readdir(fich)))
        {
            if ((strcmp(abrir->d_name, ".") != 0) && (strcmp(abrir->d_name, "..") != 0))
            {
                string p = path;
                p.append("\\");
                p.append(abrir->d_name);
                stat(p.c_str(), &attrib_file);
                if (S_ISDIR(attrib_file.st_mode)) // Diretoria
                {
                    DataCriacao = gmtime(&attrib_file.st_ctime);
                    Diretoria* D = new Diretoria(DataCriacao, path, abrir->d_name);
                    Dir->AddDir(D);
                    Load(p, D);
                }
                else if (S_ISREG(attrib_file.st_mode)) //Ficheiro
                {
                    DataCriacao = gmtime(&attrib_file.st_ctime);
                    Ficheiro* F = new Ficheiro(DataCriacao, attrib_file.st_size, path, abrir->d_name);
                    Dir->AddFich(F);
                }
            }
            Resultado = true;
        }

        closedir(fich);
    }
    return Resultado;
}
```

Esta função é responsável pelo carregamento de todo o conteúdo presente na diretoria que o utilizador definir

6.3 Funções Contar

```
int SistemaFicheiros::ContarFicheiros()
{
    int NumFich = 0;
    this->Raiz->ContarFicheiros(&NumFich);
    return NumFich;
}

int SistemaFicheiros::ContarDirectorias()
{
    int NumDir = 0;
    this->Raiz->ContarDiretorias(&NumDir);
    return NumDir;
}
```

As funções acima possuem a mesma utilidade para itens diferentes. Enquanto a primeira função é capaz de contar o número de ficheiros, a segunda é responsável por contar o numero de diretorias

6.4. Função Memória

```
int SistemaFicheiros::Memoria()
{
    int MemoriaTotal = 0;
    MemoriaTotal = MemoriaTotal + sizeof(SistemaFicheiros);
    this->Raiz->Memoria(&MemoriaTotal);
    return MemoriaTotal;
}
```

A função memoria tem como objetivo mostrar a quantidade de memoria carregada que esta a a ser utilizada pelo programa

6.5 Funções Número de Elementos

```
string SistemaFicheiros::DiretoriaMaisElementos()
{
    string DirPath;
    string DirName;
    int NumElementos = 0;
    this->Raiz->DirMaisElementosRoot(&NumElementos, &DirPath, &DirName);
    cout << DirName << endl << "Tem: " << NumElementos << " elementos" << endl << "Caminho: ";
    return(DirPath);
}

string SistemaFicheiros::DiretoriaMenosElementos()
{
    string DirPath;
    string DirName;
    int NumElementos = 10000;
    this->Raiz->DirMenosElementosRoot(&NumElementos, &DirPath, &DirName);
    cout << DirName << endl << "Tem: " << NumElementos << " elementos" << endl << "Caminho: ";
    return(DirPath);
}
```

Com a utilização desta função o utilizador é capaz de verificar que diretoria tem mais e menos elementos respetivamente

6.6 Funções Relativas a Memória

```
string SistemaFicheiros::FicheiroMaior()
{
    string FilePath;
    string FileName;
    int Tamanho = 0;
    this->Raiz->FichMaior(&Tamanho, &FilePath, &FileName);
    cout << FileName << endl << "Ocupa: " << Tamanho << " bytes." << endl << "Caminho: ";
    return(FilePath);
}

string SistemaFicheiros::DiretoriaMaisEspaco()
{
    string DirPath;
    string DirName;
    int Tamanho = 0;
    this->Raiz->DiretoriaMaiorRoot(&Tamanho, &DirPath, &DirName);
    cout << DirName << endl << "Ocupa: " << Tamanho << " bytes." << endl << "Caminho: ";
    return(DirPath);
}
```

A função apresentada permite o utilizador verificar qual é o ficheiro mais pesado de todos assim como a diretoria que ocupa o maior espaço na memoria

6.7 Função Search

```
string SistemaFicheiros::Search(const string& s, int Tipo)
{
    string Caminho = "Nao encontrado";

    if (Tipo == 1)
        this->Raiz->SearchFicheiros(s, Caminho);

    if (Tipo == 2)
        this->Raiz->Search(s, Caminho);

    return Caminho;
}
```

A função como o nome indica, permite que o utilizador faça a pesquisa tanto de uma diretoria como de um ficheiro atravez do seu nome

6.8 Função RemoverALL

```
bool SistemaFicheiros::RemoverAll(const string& s, const string& tipo)
{
    bool Resultado = false;

    if (tipo.compare("DIR") == 0)
        Resultado = this->Raiz->RemoveDiretoria(s);
    else
        Resultado = this->Raiz->RemoveFicheiros(s);

    return Resultado;
}
```

A função “RemoverALL”, permite o utilizador remover todas as diretorias ou todos os ficheiros presentes no caminho que o mesmo desejar, mas nunca fazendo os 2 em simultâneo

6.9 Funções Relativas ao XML

```
void SistemaFicheiros::Escrever_XML(const string& s)
{
    ofstream File;
    File.open(s);
    if (!File)
    {
        cout << "Erro na abertura do ficheiro em " << __FUNCTION__ << endl;
        return;
    }

    File << "<SistemaFicheiros>\n" << endl;
    this->Raiz->EscreverXML(File, 0);
    File << "</SistemaFicheiros>\n" << endl;
    File.close();


}

bool SistemaFicheiros::Ler_XML(const string& s)
{
    return false;
}
```

Ambas as funções permitem o utilizador utilizar as funcionalidades do XML uma permitindo o programa escrever, e a outra ler

Nota: Devido a algumas dificuldades encontradas não fomos capazes de concluir a função “Ler_XML”.

6.10 Funções de Verificação de Existencia

```
bool SistemaFicheiros::VerificarExistenciaFicheiro(const string& NFich)
{
    if (FILE* fich = fopen(NFich.c_str(), "r"))
    {
        fclose(fich);
        return true;
    }
}

bool SistemaFicheiros::VerificarExistenciaDiretoria(const string& NDir)
{
    struct stat buffer;

    if (stat(NDir.c_str(), &buffer) != 0)
    {
        cout << endl << "[ERRO!!]: A diretoria com o nome " << NDir << " nao foi encontrada" << endl << endl;
        cout << "Insira uma diretoria valida!" << endl << "Usage (Exemplo): C:\\Users\\user\\...\\...\\...\\...\\DirectoriaTeste" << endl << endl;
        system("pause");
        return false;
    }
    else
    {
        return true;
    }
}
```

Ambas as funções permitem ao utilizador verificar a existência de uma diretoria ou de um ficheiro no path que lhes for fornecido

6.11 Funções para Mover

```
bool SistemaFicheiros::MoveFicheiro(const string& Fich, string DirAntiga, string DirNova)
{
    return this->Raiz->MoveFicheiro(Fich, DirAntiga, DirNova);
}

bool SistemaFicheiros::MoverDirectoria(const string& DirOld, const string& DirNew)
{
    return this->Raiz->MoverDirectoria(DirOld, DirNew);
}
```

As funções permitem mover ficheiros e Diretorias, respetivamente, para o path que o utilizador desejar, caso a diretoria do mesmo já esteja carregada em memoria

6.12 Função DataFicheiro

```
string SistemaFicheiros::DataFicheiro(const string& ficheiro)
{
    tm* Data = this->Raiz->DataFicheiro(ficheiro);
    string resultado;
    if (Data != NULL)
    {
        // years since 1900         // months since January - [0, 11]
        cout << "Data de modificacao do ficheiro <" << ficheiro << ">: " << Data->tm_year + 1900 << "|" << Data->tm_mon + 1 << "|" << Data->tm_mday;
        return "";
    }
    else
        return "Ficheiro nao encontrado!";
}
```

A função apresentada em cima tem o objetivo mostrar a data de modificação do ficheiro que o utilizador inserir

6.13 Função Tree

```
void SistemaFicheiros::Tree()
{
    system("tree /f");  //mostra a tree do path atual no ecrã
}
```

A função, assim como o seu nome indica, mostra a tree (no ecrã) do path onde o programa se encontra naquele momento.

6.14 Funções de Pesquisa

```
void SistemaFicheiros::PesquisarAllDirectorias(list<string>& lres, const string& dir)
{
    this->Raiz->PesquisarAllDiretoriasRoot(lres, dir);
}

void SistemaFicheiros::PesquisarAllFicheiros(list<string>& lres, const string& file)
{
    this->Raiz->PesquisarAllFicheiros(lres, file);
}
```

As funções seguintes tem como sua funcionalidade pesquisar por uma diretoria ou por um ficheiro que o utilizador queira e especifique

6.15 Função RenomearFicheiros

```
void SistemaFicheiros::RenomearFicheiros(const string& fich_old, const string& fich_new)
{
    this->Raiz->RenomearFicheiros(fich_old, fich_new);
    cout << "O(s) ficheiro(s) com o nome <" << fich_old << "> foi/foram renomeado(s) para: <" << fich_new << ">" << endl;
}
```

A função “RenomearFicheiros”, é utilizada para alterar o nome de um ficheiro

6.16 Função FicheirosDuplicados

```
bool SistemaFicheiros::FicheirosDuplicados()
{
    list<string> LNomes;
    bool Resultado = false;
    Resultado = this->Raiz->FicheirosDuplicados(LNomes);
    return Resultado;
}
```

A função apresentada em cima, é responsavel por verificar se existem ficheiros iguais na diretoria carregada em memoria

6.17 Função CopyBatch

```
bool SistemaFicheiros::CopyBatch(const string& padrao, const string& DirOrigem, const string& DirDestino)
{
    this->Raiz->CopyBatch(padrao, DirOrigem.c_str(), DirDestino);
    return false;
}
```

Esta ultima função deste projeto, tem como objetivo copiar todo o conteudo de uma diretoria para outra a escolha

7 Diretoria

```
#include "Diretoria.h"

Diretoria::Diretoria(tm* _data, string _caminho, string _nome)
    : ObjetoGeral(_data, _caminho, _nome)
{
    //ctor
}

Diretoria::~Diretoria()
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
        delete* it;

    for (list<Ficheiro*>::iterator it = LFich.begin(); it != LFich.end(); ++it)
        delete* it;
}

void Diretoria::ContarFicheiros(int* NumFich)
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
        (*it)->ContarFicheiros(NumFich);

    for (list<Ficheiro*>::iterator it2 = LFich.begin(); it2 != LFich.end(); ++it2)
        *NumFich = *NumFich + 1;
}
```

```
void Diretoria::ContarDiretorias(int* NumDir)
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
    {
        *NumDir = *NumDir + 1;
        (*it)->ContarDiretorias(NumDir);
    }
}

void Diretoria::Memoria(int* MemoriaTotal)
{
    *MemoriaTotal = *MemoriaTotal + sizeof(Diretoria) + sizeof(Ficheiro) * this->GetNumFich();
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
        (*it)->Memoria(MemoriaTotal);
}

void Diretoria::DirMaisElementos(int* NumElementos, string* DirPath, string* DirName)
{
    if (*NumElementos < this->GetNumElem())
    {
        *NumElementos = this->GetNumElem();
        *DirPath = this->GetPath();
        *DirName = this->GetNome();
    }

    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
    {
        *NumElementos = *NumElementos + this->GetNumElem();
        (*it)->DirMaisElementos(NumElementos, DirPath, DirName);
    }
}
```

```
void Diretoria::DirMaisElementosRoot(int* NumElementos, string* DirPath, string* DirName)
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
    {
        (*it)->DirMaisElementos(NumElementos, DirPath, DirName);
    }
}

void Diretoria::DirMenosElementos(int* NumElementos, string* DirPath, string* DirName)
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
    {
        *NumElementos = *NumElementos + this->GetNumElem();
        (*it)->DirMenosElementos(NumElementos, DirPath, DirName);
    }

    if (*NumElementos > this->GetNumElem())
    {
        *NumElementos = this->GetNumElem();
        *DirPath = this->GetPath();
        *DirName = this->GetNome();
    }
}

void Diretoria::DirMenosElementosRoot(int* NumElementos, string* DirPath, string* DirName)
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
    {
        (*it)->DirMenosElementos(NumElementos, DirPath, DirName);
    }
}

void Diretoria::FichMaior(int* Tamanho, string* FichPath, string* NomeFich)
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
        (*it)->FichMaior(Tamanho, FichPath, NomeFich);

    for (list<Ficheiro*>::iterator it2 = LFich.begin(); it2 != LFich.end(); ++it2)
    {
        if ((*it2)->GetTamanho() > *Tamanho)
        {
            *Tamanho = (*it2)->GetTamanho();
            *FichPath = (*it2)->GetPath();
            *NomeFich = (*it2)->GetNome();
        }
    }
}

int Diretoria::GetTamanhoDir()
{
    int Tamanho_Total = 0;
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
        Tamanho_Total = (*it)->GetTamanhoDir();

    for (list<Ficheiro*>::iterator it2 = LFich.begin(); it2 != LFich.end(); ++it2)
        Tamanho_Total = Tamanho_Total + (*it2)->GetTamanho();

    return Tamanho_Total;
}
```

```
void Diretoria::DiretoriaMaior(int* Tamanho, string* DirPath, string* NomeDir)
{
    if (*Tamanho < this->GetTamanhoDir())
    {
        *Tamanho = this->GetTamanhoDir();
        *DirPath = this->GetPath();
        *NomeDir = this->GetNome();
    }

    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
    {
        (*it)->DiretoriaMaior(Tamanho, DirPath, NomeDir);
    }
}

void Diretoria::DiretoriaMaiorRoot(int* Tamanho, string* DirPath, string* NomeDir)
{
    for (list<Ficheiro*>::iterator it = LFich.begin(); it != LFich.end(); ++it)
        (*it)->GetTamanho();

    for (list<Diretoria*>::iterator it2 = LDir.begin(); it2 != LDir.end(); ++it2)
    {
        if ((*it2)->GetTamanhoDir() > *Tamanho)
        {
            *Tamanho = (*it2)->GetTamanhoDir();
            *DirPath = (*it2)->GetPath();
            *NomeDir = (*it2)->GetNome();
        }
    }
}

void Diretoria::SearchFicheiros(const string& FileName, string& Caminho)
{
    for (list<Ficheiro*>::iterator it = LFich.begin(); it != LFich.end(); ++it)
    {
        if ((*it)->GetNome().compare(FileName) == 0)
        {
            Caminho = (*it)->GetPath();
            return;
        }
    }
    for (list<Diretoria*>::iterator it2 = LDir.begin(); it2 != LDir.end(); ++it2)
        (*it2)->SearchFicheiros(FileName, Caminho);
}

void Diretoria::Search(const string& DirName, string& Caminho)
{
    if (this->GetNome().compare(DirName) == 0)
    {
        Caminho = this->GetPath();
        return;
    }
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
        (*it)->Search(DirName, Caminho);
}

bool Diretoria::RemoveDiretoria(const string& DirName)
{
    if (fs::remove_all(DirName.c_str())) //Apaga todo conteúdo da diretoria e de todas as suas subdiretorias, recursivamente, e no final apaga-se a si mesmo(diretoria "mãe")
    {
        if (LFich.size() != 0)
            this->LFich.erase(LFich.begin(), LFich.end());
        if (LDir.size() != 0)
            this->LDir.erase(LDir.begin(), LDir.end());
        delete this;

        cout << "Diretoria " << DirName << " eliminada com sucesso!" << endl;
        return true;
    }
    else
    {
        cout << "Nao foi possivel eliminar a diretoria com o nome " << DirName << endl;
        return false;
    }
}
```

```cpp
bool Diretoria::RemoveFicheiros(const string& Path)
{
    if (fs::exists(Path.c_str()) && fs::is_directory(Path.c_str()))
    {
        fs::directory_iterator end;
        for (fs::directory_iterator it(Path.c_str()); it != end; ++it)
        {
            if (fs::is_regular_file(it->status()) && (it->path().extension() == ".txt") || (it->path().extension() == ".pdf") || (it->path().extension() == ".docx"))      //adicionar com "ou" as extensões dos ficheiros a eliminar
            {
                for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
                    (*it)->RemoveFicheiros(Path);

                if (this->LFich.size() != 0)
                    this->LFich.erase(LFich.begin(), LFich.end());

                fs::remove(it->path());
            }
        }
        return true;
    }
    else
    {
        cout << endl << "[ERRO!!]: Nao foi possivel eliminar os ficheiros da diretoria com o nome " << Path << endl;
        return false;
    }
}
```

```cpp
void Diretoria::EscreverXML(ofstream& File, int Espacos)
{
    EscreveElementoXML(File, Espacos, "Diretoria");
    File << "\n";

    EscreveElementoXML(File, Espacos + 1, "Nome");
    File << this->GetNome();
    FechaElementoXML(File, 0, "Nome");

    EscreveElementoXML(File, Espacos + 1, "Path");
    File << this->GetPath();
    FechaElementoXML(File, 0, "Path");

    if (this->GetNumDir() != 0)
    {
        EscreveElementoXML(File, Espacos + 1, "LDir");
        File << "\n";
        for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
            (*it)->EscreverXML(File, Espacos + 1);
        FechaElementoXML(File, Espacos + 1, "LDir");
    }

    if (this->GetNumFich() != 0)
    {
        EscreveElementoXML(File, Espacos + 1, "LFich");
        File << "\n";
        for (list<Ficheiro*>::iterator it = LFich.begin(); it != LFich.end(); ++it)
            (*it)->EscreverXML(File, Espacos + 2);
        FechaElementoXML(File, Espacos + 1, "LFich");
    }

    FechaElementoXML(File, Espacos, "Diretoria");
}
```

```cpp
    }
    else
    {
        cout << "[ERRO!!]" << endl;
        cout << "Nao foi possivel localizar o ficheiro: " << DirAntiga << endl;
        cout << "Por favor insira uma diretoria valida." << endl;
        return false;
    }
}
```

```
    if (DirAntigaFich)
    {
        ifstream iDirNova(DirNova.c_str());
        if (iDirNova)
        {
            cout << Fich << " O ficheiro que esta a mover ja existe na diretoria nova." << endl << endl << "A sair......." << endl;
            system("pause");
            iDirNova.close();
            return false;
        }
        else
        {
            iDirNova.close();
            ofstream DirNovaFich(DirNova.c_str(), ios::binary);
            string line;
            while (getline(DirAntigaFich, line))
            {
                DirNovaFich << line << endl;;
            }
            DirNovaFich.flush();
            DirNovaFich.close();
            DirAntigaFich.close();
            int old_file = remove(DirAntiga.c_str());
            if (old_file == 0)
            {
                return true;
            }
            else
            {
                cout << "[ERRO!!]: Nao foi possivel mover o ficheiro. Tente novamente por favor." << endl;
                system("pause");
                return false;
            }
        }
        DirAntigaFich.close();
```

```
bool Diretoria::MoveFicheiro(const string& Fich, string DirAntiga, string DirNova)
{
    for (int i = 0; i < DirAntiga.size(); i++)
    {
        if (DirAntiga[i] == '\\')
            DirAntiga[i] = '/';
    }

    for (int i = 0; i < DirNova.size(); i++)
    {
        if (DirNova[i] == '\\')
            DirNova[i] = '/';
    }

    if (DirAntiga[DirAntiga.size() - 1] != '/')
        DirAntiga.push_back('/');

    DirAntiga = DirAntiga + Fich;

    if (DirNova[DirNova.size() - 1] != '/')
        DirNova.push_back('/');

    DirNova = DirNova + Fich;
    ifstream DirAntigaFich(DirAntiga.c_str(), ios::binary);
```

```
bool Diretoria::MoverDirectoria(const string& DirOld, const string& DirNew)
{
    const auto copyOptions = fs::copy_options::overwrite_existing | fs::copy_options::recursive;   //opções adicionais que controlam o comportamento da funcao copy()
                             //overwrite_existing -> Substitui o arquivo existente caso o mesmo exista na diretoria destino
                            //recursive -> Conta recursivamente sub-diretorias e o seu conteudo
    if (rename(DirOld.c_str(), DirNew.c_str()))     //Atualiza a diretoria antiga para a diretoria nova
    {
        fs::copy(DirOld.c_str(), DirNew.c_str(), copyOptions);      //Copia todo o conteúdo presente na diretoria antiga para a diretoria nova, incluindo sub-diretorias (recursivamente)
        fs::remove_all(DirOld.c_str());                             //Elimina todo o conteúdo presente na diretoria antiga, incluindo sub-diretorias (recursivamente)
    }
    else
    {
        cout << "[ERRO!!]" << endl;
        cout << "Nao foi possivel localizar a diretoria: " << DirOld << endl;
        cout << "For favor insira uma diretoria valida." << endl;
        return false;
    }
}
```

```
tm* Diretoria::DataFicheiro(const string& NomeFich)
{
    for (list<Ficheiro*>::iterator it = LFich.begin(); it != LFich.end(); ++it)
    {
        if ((*it)->GetNome().compare(NomeFich) == 0)
            return ((*it)->GetData(NomeFich));
    }
    for (list<Diretoria*>::iterator it2 = LDir.begin(); it2 != LDir.end(); ++it2)
        (*it2)->DataFicheiro(NomeFich);

    return NULL;
}
```

```
void Diretoria::PesquisarAllDiretorias(list<string>& LPath, const string& NomeDir)
{
    if (this->GetNome().compare(NomeDir) == 0)
    {
        LPath.push_back(this->GetPath());
    }
    for (list<Diretoria*>::iterator it2 = LDir.begin(); it2 != LDir.end(); ++it2)
        (*it2)->PesquisarAllDiretorias(LPath, NomeDir);
}

void Diretoria::PesquisarAllDiretoriasRoot(list<string>& LPath, const string& NomeDir)
{
    for (list<Diretoria*>::iterator it2 = LDir.begin(); it2 != LDir.end(); ++it2)
        (*it2)->PesquisarAllDiretorias(LPath, NomeDir);
}

void Diretoria::PesquisarAllFicheiros(list<string>& LPath, const string& NomeFicheiro)
{
    for (list<Ficheiro*>::iterator it = LFich.begin(); it != LFich.end(); ++it)
    {
        if ((*it)->GetNome().compare(NomeFicheiro) == 0)
            LPath.push_back((*it)->GetPath());
    }
    for (list<Diretoria*>::iterator it2 = LDir.begin(); it2 != LDir.end(); ++it2)
        (*it2)->PesquisarAllFicheiros(LPath, NomeFicheiro);
}

void Diretoria::RenomearFicheiros(const string& NFich, const string& NNovo)
{
    for (list<Ficheiro*>::iterator it = LFich.begin(); it != LFich.end(); ++it)
    {
        if ((*it)->GetNome().compare(NFich) == 0)
            (*it)->Renomear(NNovo);
    }
    for (list<Diretoria*>::iterator it2 = LDir.begin(); it2 != LDir.end(); ++it2)
        (*it2)->RenomearFicheiros(NFich, NNovo);
}

void Diretoria::RenomearFicheirosRoot(const string& NFich, const string& NNovo)
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
        (*it)->RenomearFicheiros(NFich, NNovo);
}

bool Diretoria::VerificaNomes(list<string>& LNomes, string NomeVerificar)
{
    if (LNomes.size() == 0)
        return false;

    for (list<string>::iterator it = LNomes.begin(); it != LNomes.end(); ++it)
    {
        if (NomeVerificar.compare((*it)) == 0)
            return true;
    }
    return false;
}
```

```cpp
bool Diretoria::FicheirosDuplicados(list<string>& LNomes)
{
    for (list<Diretoria*>::iterator it = LDir.begin(); it != LDir.end(); ++it)
    {
        int Aux = (*it)->FicheirosDuplicados(LNomes);
        if (Aux)
            return true;
    }

    for (list<Ficheiro*>::iterator it2 = LFich.begin(); it2 != LFich.end(); ++it2)
    {
        if (VerificaNomes(LNomes, (*it2)->GetNome()))
            return true;
        LNomes.push_back((*it2)->GetNome());
    }
}
```

```cpp
bool Diretoria::CopyBatch(const string& padrao, const string& DirOrigem, const string& DirDestino)
{
    const auto copyOptions = fs::copy_options::skip_existing | fs::copy_options::recursive;   //opções adicionais que controlam o comportamento da funçao copy()
                             //skip_existing -> Ignora o arquivo caso o mesmo exista na diretoria destino (com o mesmo nome)
                            //recursive -> Copia recursivamente sub-diretorias e o seu conteudo
    if (rename(DirOrigem.c_str(), DirDestino.c_str()))     //Atualiza a diretoria antiga para a diretoria nova
    {
        fs::copy(DirOrigem.c_str(), DirDestino.c_str(), copyOptions);      //Copia todo o conteúdo presente na diretoria antiga para a diretoria nova, incluindo sub-diretorias (recursivamente)
        return true;
    }
    else
    {
        cout << "[ERRO!!]" << endl;
        cout << "Nao foi possivel localizar a diretoria: " << DirOrigem << endl;
        cout << "Por favor insira uma diretoria valida." << endl;
        return false;
    }
    return true;
}
```

8 Main

Finalmente iremos mostrar o nosso Main

```cpp
#include "libs.h"
#include "SistemaFicheiros.h"
#include "Menus.h"

int main()
{
    setlocale(LC_ALL, "Portuguese");
    SistemaFicheiros* F = new SistemaFicheiros();
    Menus::Select(F);

    return 0;
}
```

Como é possível verificar no caso do nosso programa, o main encontra-se com poucos itens, apenas mostrando o necessário. Assim que o programa inicia é automaticamente redirecionado para o menu principal onde depois se fará a seleção da funcionalidade que o utilizador pretender executar.

9. Conclusão

Para concluir, é importante fazer uma reflexão sobre o projeto e o processo de escrita do programa.

A elaboração deste programa permitiu-nos por em prática tudo o que foi lecionado ao decorrer de todas as aulas que tivemos. Serviu também para pôr em prática todo o tipo de funções, familiares ou não, assim como novas funções com que pouco trabalhamos.

Numa reflexão mais geral, a realização de um projeto como este permitiu-nos ganhar um grau elevado de conhecimento sobre a linguagem C++, sobretudo na área relacionada com o tratamento de ficheiros e diretorias. Apesar de certa forma se ter tornado um trabalho desafiante e interessante, muitas vezes acabou por ser frustrante devido aos erros e ao tempo necessário a dedicar ao trabalho. No entanto, após a sua conclusão, foi satisfatório ver o programa a funcionar com as funcionalidades implementadas.