Exercício 6:
Classes Abstratas
Métodos Virtuais
Redefinição de métodos
Apontadores

Professor Miguel Melo

Programação Orientada a Objetos

Engenharia Informática

utad UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

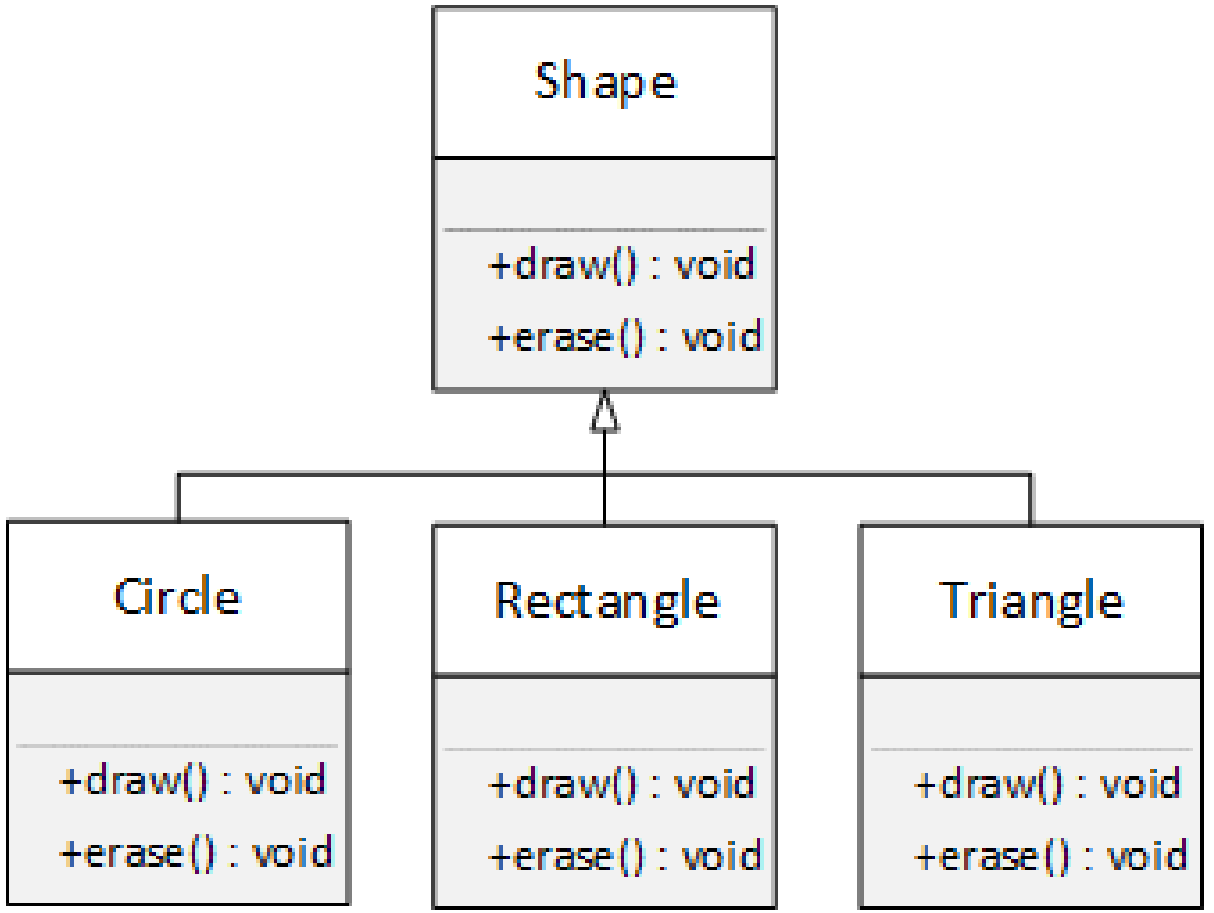
Classes abstratas e métodos virtuais

- Por vezes, uma classe pode servir de base a diferentes sub-classes
  - Problema: Por vezes, existem métodos que podem variar de sub-classe para sub-classe, sendo que uma implementação na classe mãe é inviável
  - Solução: criar um método base que possa ser estendido pelas sub-classes (**método virtual**)
  - Sintaxe:

```
virtual float Calcula_Ordenado();
```

Classes abstratas e métodos virtuais

- **Classe abstrata:** classe com pelo menos um método virtual
- **Método virtual:** método abstrato que pode ser estendido pelas sub-classes



Apontadores

- Até agora, temos trabalhado diretamente com o valor das variáveis(ex: *int i = 20;*)
- Por vezes, é necessário passar um argumento para dentro duma função. Para ultrapassar isso, usa-se apontadores (ex: *int* i = 20*)

```
int square1(int n)
{
    // Address of n in square1() is not the same as n1 in
    // main()

    cout << "address of n1 in square1(): "  << &n << "\n";

    // clone modified inside the function     return  n;
}
```

Apontadores

- Até agora, temos trabalhado diretamente com o valor das variáveis(ex: *int i = 20;*)
- Por vezes, é necessário passar um argumento para dentro duma função. Para ultrapassar isso, usa-se apontadores (ex: *int* i = 20*)

```
void square2(int* n)
{
   // Address of n in square2() is the same as n2 in main()
   cout << "address of n2 in square2(): " << n << "\n";
}
```

Criação de Menus

- **Criação de um ciclo while(true)**: ou seja, está sempre em loop
- **Dentro do ciclo while:**
  - Imprime todas as opções e indica qual o número que corresponde a cada uma dessas opções
  - espera que o utilizador introduza uma opção (imprime neste caso através de um número no teclado)
  - Usa um ciclo Switch que, consoante a opção selecionada, chama o método correspondente
  - Após executar esse método, o ciclo while é executado novamente e aguarda novo input do utilizador

Criação de Menus

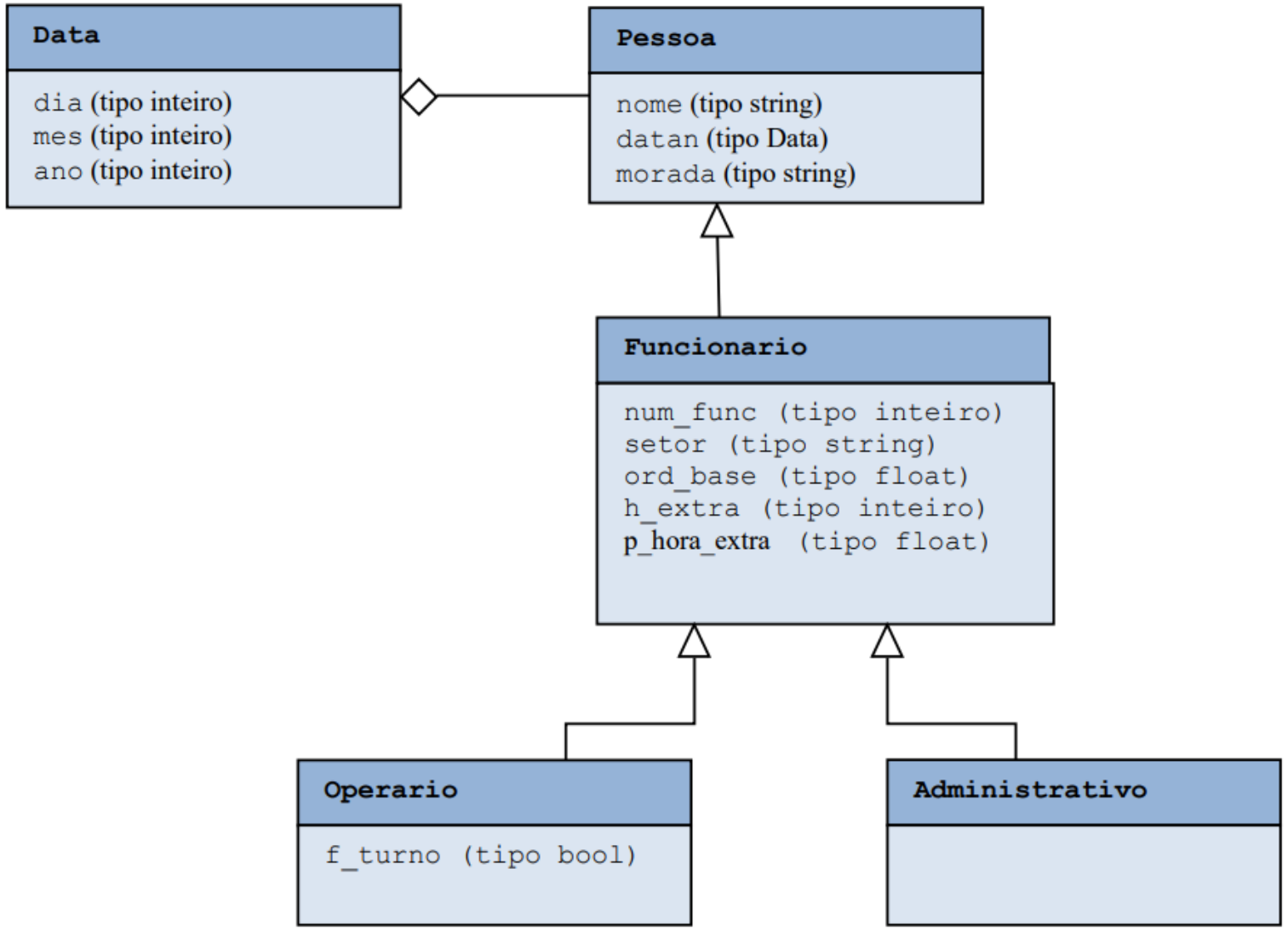
- **Exemplo Básico:**

```
int op;
while (true)
{
    cout << endl << endl << " ------    Bem Vindo ao Sistema XPTO     -----  " << endl
        << "\t 1 - Ler Ficheiro" << endl
        << "\t 0 - Sair" << endl
        << "Escolha a opção:";
    cin >> op;

    switch (op)
    {
    case 1:
        LerFicheiro(Lista);
        break;
    case 0:
        exit(0);
        break;
    }
}
```

Exercício 6

- 2 tipos de funcionários: Operários e Administrativos.
- Todos os funcionários têm um ordenado base (ord_base) ao qual podem ser acrescentados complementos de acordo com o seu trabalho
- As horas de trabalho extraordinário (h_extra) realizadas durantes o mês são pagas como complementos, ao valor de 10€/hora para os Administrativos e de 8€/hora para os Operários.
- Os Operários podem optar por trabalhar por turnos e nesse caso recebem sempre um complemento de 25%, relativamente ao ordenado base.



Exercício 6.1

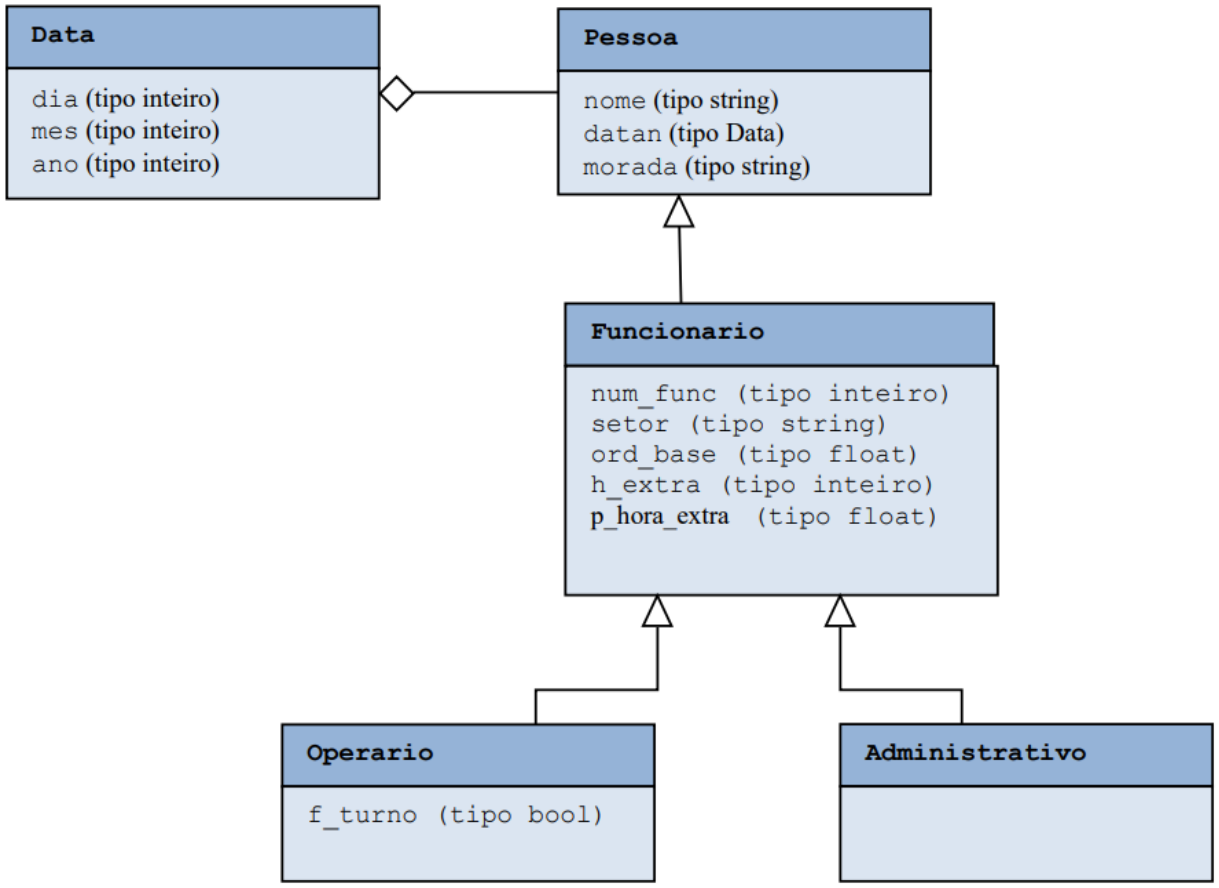
- Faça as devidas alterações à classe Funcionario, acrescentando um método virtual puro *Calcula_ordenado()*, tornando-a numa classe abstrata. Altere em conformidade os construtores com parâmetros da classe.

Exercício 6.1 - Proposta de resolução

1. Acrescenta as variáveis à classe Funcionario
2. Define os seus *getters* e *setters*
3. Atualiza os construtores para incluírem as novas variáveis
4. Acrescenta o método virtual *Calcula_Ordenado()*

Exercício 6.2 e 6.3

- Defina as classes *Operario* e *Administrativo* de acordo com o seguinte diagrama.
- Implemente os métodos construtores e de acesso em ambas as classes derivadas.



Exercício 6.2 e 6.3 - Proposta de resolução

.h

```cpp
#include "Operario.h"

Operario::Operario(int _num, string _setor, string _nome, string _morada, Data _data, float _ord, int _hextra, bool _turno)
    : Funcionario(_num, _setor, _nome, _morada, _data, _ord, _hextra, 8)
{
    f_turno = _turno;
}

Operario::Operario() :Funcionario()
{
    SetPrecoHorasExtra(8);
}
```

.cpp

```cpp
#pragma once
#include "Funcionario.h"
class Operario :public Funcionario
{
    private:
        bool f_turno;

    public:
        bool GetTurno() {
            return f_turno;
        }
        void SetTurno(bool t) {
            f_turno = t;
        }

        Operario();
        Operario(int _num, string _setor, string _nome, string _morada, Data _data, float _ord, int _hextra, bool _turno);

};
```

Exercício 6.4

- Faça a redefinição do método *Calcula_ordenado()* nas duas classes derivadas

Exercício 6.4 - Proposta de resolução (Operario.cpp)

- Faça a redefinição do método *Calcula_ordenado()* nas duas classes derivadas

```
float Operario::Calcula_Ordenado()
{
    if (f_turno)
    {
        return GetOrdBase() * 1.25f + GetHorasExtra() * GetPrecoHorasExtra();
    }
    else
    {
        return GetOrdBase()  + GetHorasExtra() * GetPrecoHorasExtra();

    }

}
```

Exercício 6.5 - alínea a

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
  - a) Leitura do ficheiro *"Funcionario.txt"* (ver exemplo), com os dados de ***N*** *Operarios* e *Administrativos*, para um vector de apontadores.O primeiro caractere da linha indica se é um *Operario (O)* ou *Admnistrativo (A)*.

```
O:Jonas Culatra; 20/9/1987; Rua da direita n 2; 25023; Pintura;1100;20;sim;
A:Joni Rato; 4/2/1990; Rua da esquerda n 3; 25024; Comercial;900;0;
O:Luis Carlos; 1/12/1985; Rua Central n22;10000; Ferragem; 1100;0;nao;
…
EOF
```

Exercício 6.5 - alínea a (Proposta de resolução)

- No método main(), cria o ciclo while(true) que te permita introduzir um valor para selecionar opções através de um switch (nota: para fins de teste, o primeiro pode ser apenas com a opção sair e depois vais acrescentando as opções pedidas)

```
int opcao;

while (true)
{
    cout << endl << endl << " ------    Bem Vindo ao Menu Inicial     -----  " << endl
        << "\t 1 - Ler Ficheiro" << endl
        << "Escolha a opção:";

    cin >> opcao;

    switch (opcao)
    {
    case 0:
        exit(0);
        break;
    }
}
```

Exercício 6.5 - alínea a (Proposta de resolução)

- Cria o método LerFicheiro(vector<Funcionario*> &Lista)
- Verifica se o ficheiro é aberto
- Faz um ciclo que:
  - Para casa linha, verifica até ao : se é A ou O
  - Consoante o resultado, cria um método que te permite ler do ficheiro
    - Nota: dependendo da forma como implementaste anteriormente o ReadFile(ifstream& is)
  - Caso chegues ao fim do ficheiro, fecha a stream de dados

Exercício 6.5 - alínea b

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
  - Calcular e apresentar o ordenado de todos os funcionários.

Exercício 6.5 - alínea b (proposta de resolução)

- Desenvolver a função de cálculo de salários:
  - Todos os funcionários têm um ordenado base (ord_base) ao qual podem ser acrescentados complementos de acordo com o seu trabalho
  - As horas de trabalho extraordinário (h_extra) realizadas durantes o mês são pagas como complementos, ao valor de 10€/hora para os Administrativos e de 8€/hora para os Operários.
  - Os Operários podem optar por trabalhar por turnos e nesse caso recebem sempre um complemento de 25%, relativamente ao ordenado base.

```
float Administrativo::Calcula_Ordenado()
{
    return GetOrdBase() + GetHorasExtra() * GetPrecoHorasExtra();
}
```

```
float Operario::Calcula_Ordenado()
{
    if (f_turno)
    {
        return GetOrdBase() * 1.25f + GetHorasExtra() * GetPrecoHorasExtra();
    }
    else
    {
        return GetOrdBase()  + GetHorasExtra() * GetPrecoHorasExtra();

    }

}
```

Exercício 6.5 - alínea b (proposta de resolução)

- Fazer o método correspondente à opção 2 para percorrer o vector preenchido e listar os salários dos diferentes funcionários:

```
void CalcularOrdenados(vector<Funcionario*> &Lista)
{
    cout << endl << endl << "-----    Lista de Funcionarios    -----" << endl;

    for (int i = 0; i < Lista.size(); i++)
    {
        cout << Lista[i]->GetNum() << "\t"
            << Lista[i]->GetNome() << "\t"
            << Lista[i]->Calcula_Ordenado() << endl;
    }

    cout << endl << endl;
}
```

Exercício 6.5 - alínea c

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
    - Pesquisar um funcionário pelo seu número.

Exercício 6.5 - alínea c (proposta de resolução)

- Criar a opção no menu e o respetivo método:
  1. Pede número de funcionário
  2. Percorre a lista e compara todos os nums. de funcionário com o número introduzido. Se for igual, mostra esse funcionário. Caso não encontre igual, apresenta a informação que não foi encontrado

```
void PesquisaFuncionario(vector<Funcionario*> &Lista)
{
    cout << endl << endl << "-----    Pesquisa de Funcionarios    -----" << endl;

    int numero;
    cout << "Insira o numero do Funcionario:" << endl;
    cin >> numero;

    //percorremos o vector a procura do utilizador com o numero inserido.
    for (int i = 0; i < Lista.size(); i++)
    {

        if (Lista[i]->GetNum() == numero)
        {
            Lista[i]->Show();
            return; //se encontrar termina aqui
        }

    }
    cout << "Não foi encontrado o utilizador" << endl;
}
```

Exercício 6.5 - alínea d

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
  - Alterar o ordenado base de um funcionário, identificado pelo seu número.

Exercício 6.5 - alínea d (proposta de resolução)

- Criar a opção no menu e o respetivo método:
    1. Pede número de funcionário
    2. Percorre a lista e compara todos os nums. de funcionário com o número introduzido.
        1. Se for encontrado, mostra o salario atual, pede input de novo salario e atualiza o salario desse funcionário
        2. Caso não encontre igual, apresenta a informação que não foi encontrado

Exercício 6.5 - alínea e

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
  - Adicionar ao vector um Operario ou Administrativo, à escolha do utilizador

Exercício 6.5 - alínea e (proposta de resolução)

1. Criar métodos para adicionar A e O através do teclado
2. Pedir para identificar tipo de funcionário
3. Consoante o tipo, de funcionário, chamar o método correspondente
4. Adicionar o objeto ao vetor

```
void CriarFuncionario(vector<Funcionario*> &Lista)
{
    cout << endl << endl << "-----    Criar Funcionario    -----" << endl;

    string tipo;
    //descartar o lixo do buffer
    cin.clear();
    fflush(stdin);

    cout << "Deseja inserir: " << endl
        << "O - Operario" << endl
        << "A - Administrativo" << endl;
    getline(cin, tipo, '\n');

    if (tipo == "O" || tipo == "o")
    {
        Operario *op = new Operario();
        op->ReadK(); //o metodo readK permite ler a informação a partir do teclado.
        Lista.push_back(op);
    }
    else if (tipo == "A" || tipo == "a")
    {
        Administrativo *adm = new Administrativo();
        adm->ReadK();
        Lista.push_back(adm);
    }

    cout << endl;

    cin.clear();
    fflush(stdin);
}
```

Exercício 6.5 - alínea f

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
  - Apresentar o nome dos operários que trabalham por turnos.

Exercício 6.5 - alínea f (proposta de resolução)

Apresentar o nome dos operários que trabalham por turnos.

```
void ApresentaTurnos(vector<Funcionario*> &Lista)
{
    cout << endl << endl << "-----    Lista de Operarios que trabalham em turno    -----" << endl;

    for (int i = 0; i < Lista.size(); i++) //percorremos cada um dos funcionarios
    {
        if (Operario *op = dynamic_cast<Operario*>(Lista[i])) // tentamos converter para operario, caso não seja possivel é porque nessa posição está um administrativo
        {
            if (op->GetTurno()) //se for um operario, vamos vereficar se ele trabalha por turno.
            {
                cout <<op->GetNum() << "\t " << op->GetNome() << endl;
            }
        }
    }
    cout << endl;
}
```

Exercício 6.5 - alínea g

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
  - Eliminar do vetor um funcionário, identificado pelo número.

Exercício 6.5 - alínea g

Eliminar do vetor um funcionário, identificado pelo número.

```
void RemoveFuncionarios(vector<Funcionario*> &Lista)
{
    int numero;
    cout << "insira o numero do Funcionario a remover:" << endl;
    cin >> numero;

    //procurar o funcionario na lista;
    for (int i = 0; i < Lista.size(); i++)
    {
        if (Lista[i]->GetNum() == numero) // se existir
        {
            //removemos do vector com o metodo erase, que recebe um "iterador" que aponta para a posição que queremos remover a partir do inicio.
            Lista.erase(Lista.begin() + i);

            cout << endl << "O funcionario " << numero << " foi removido com sucesso" << endl;
            return; //como foi removido, podemos terminar;
        }
    }

    cout << "Nenhum utilizador encontrado com esse numero!" << endl;
}
```

Exercício 6.5 - alínea h

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
  - Guardar, num ficheiro “Administrativos.txt”, os dados dos Administrativos.

Exercício 6.5 - alínea h

```
void GuardarAdministrativos(vector<Funcionario*> &Lista)
{
    ofstream file;
    file.open("Administrativos.txt");

    if (!file.is_open())
    {
        exit(-1);
    }

    for (int i = 0; i < Lista.size(); i++)
    {
        //como apenas queremos guardar administrativos, vamos tentar converter, se for possivel guardamos no ficheiro, se não for,
        // é um operario, então passamos a frente.
        if (Administrativo *adm = dynamic_cast<Administrativo*>(Lista[i]))
        {
            adm->SaveFile(file);
            file << endl;

        }

    }

    cout << "Ficheiro Administrativos.txt Guardado com sucesso" << endl;
    file.close();
}
```

Exercício 6.5 - alínea i

- Desenvolva uma aplicação que possua um menu com as seguintes opções:
  - Guardar, num ficheiro “Operarios.txt”, os dados dos Operários.

Exercício 6.5 - Proposta de resolução

```
void GuardarOperarios(vector<Funcionario*> &Lista)
{

    ofstream file;
    file.open("Operarios.txt");

    if (!file.is_open())
    {
        exit(-1);
    }

    for (int i = 0; i < Lista.size(); i++)
    {
        //como apenas queremos guardar Operarios, vamos tentar converter, se for possivel guardamos no ficheiro, se não for,
        // é um Administrativos, então passamos a frente.
        if (Operario *oper = dynamic_cast<Operario*>(Lista[i]))
        {
            oper->SaveFile(file);
            file << endl;
        }

    }

    cout << "Ficheiro Operarios.txt Guardado com sucesso" << endl;

    file.close();
}
```

- Nota: semelhante ao anterior mas estendemos o método SaveFile() para guardar também o valor da variável turno

Exercício 6:
Classes Abstratas
Métodos Virtuais
Redefinição de métodos
Apontadores

Professor Miguel Melo

Programação Orientada a Objetos

Engenharia Informática

utad UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO