



Programação 2

2º Semestre

Trabalho Prático P3A

Efectue as tarefas de programação descritas abaixo, usando a linguagem C++ em ambiente Linux.

Grupo 1

Pretende-se implementar uma classe `ListEx` em que seja possível determinar facilmente o elemento que se repete mais vezes:

```
class Erro{};
class ListElement
{
    int valor;
    int contador;
public:
    ListElement(int x) { valor = x; contador = 1;}
    friend class ListEx;
};

class ListEx
{
    list<ListElement> rol;
public:
    void imprimeLista (ostream& os)
    {
        list<ListElement>::iterator it;
        for (it = rol.begin(); it != rol.end(); it++)
        {
            os << (*it).valor << " Contador: " << (*it).contador
            << endl;
        }
    }
    /*...*/
};
```

a) Implemente o membro-função

```
int ListEx::insereOrdenado(int x);
```

Este membro-função deve inserir o elemento `x` na lista. Se `x` já existir na lista apenas deve ser incrementado o contador associado. Caso `x` não exista na lista, deve ser inserido mantendo SEMPRE a lista ordenada, por ordem crescente.

b) Implemente o membro-função

```
int ListEx::getModa() const;
```

Este membro-função retorna a moda da lista. Se a lista se encontrar vazia, deve lançar uma excepção do tipo `Erro`. A moda é o elemento com contador mais elevado. Caso exista mais que um elemento com o mesmo valor máximo de contador a função deve retornar um qualquer deles.

c) Implemente o membro-função

```
int ListEx::remove(int x);
```

Este membro-função deve remover o elemento `x` na lista. Remover corresponde a decrementar o contador associado. Caso o contador atinja o valor zero, o elemento deve ser removido da lista `rol`.



Programação 2

2º Semestre

Trabalho Prático P3A

Efectue as tarefas de programação descritas abaixo, usando a linguagem C++ em ambiente Linux.

Grupo 2

Pretende-se implementar uma classe `Viagem` para que seja possível analisar facilmente uma viagem:

```
class Erro{};
class Cidade
{
    string nome;
    double x;
    double y;
public:
    Cidade(string n, double posX, double posY) {nome = n; x = posX;
    y = posY;}
    friend class Viagem;
};
class Viagem
{
    list<Cidade> percurso;
public:
    void imprimeViagem (ostream& os)
    {
        list<Cidade>::iterator it;
        for (it = percurso.begin(); it != percurso.end(); it++)
        {
            os << (*it).nome << " (" << (*it).x << ";" << (*it).y
            <<") "<<endl;
        }
    }
    /*...*/
};
```

a) Implemente o membro-função

```
void insereCidade(string nome, double posX, double posY, string
nomeCidadeAnterior);
```

Este membro-função deve inserir a nova cidade a seguir à cidade com o nome `nomeCidadeAnterior`. Se não existir nenhuma cidade com o nome `nomeCidadeAnterior` a nova cidade deve ser inserida no fim da fila.

b) Implemente o membro-função

```
double distanciaPercorrida() const;
```

Este membro-função deve retornar a distância percorrida na viagem. As cidades foram visitadas pela ordem em que se encontram na lista. Se a lista se encontrar vazia deve ser lançada uma excepção do tipo `Erro`.

c) Implemente o membro-função

```
void Viagem::vimDeFuiPara(string nome);
```

Este membro-função deve imprimir a cidade visitada antes da cidade com o nome `nome` e a cidade visitada depois dessa cidade. Se não tiver passado por nenhuma cidade com o nome `nome` não imprime nada. Se tiver começado ou acabado a viagem na cidade com o nome `nome` apenas imprime a parte apropriada.



Programação 2

2º Semestre

Trabalho Prático P3A

Efectue as tarefas de programação descritas abaixo, usando a linguagem C++ em ambiente Linux.

Grupo 3

Pretende-se implementar uma classe para modelar o atendimento aos clientes numa mercearia. Existem duas filas: a fila para os clientes com prioridade e a fila normal.

```
class Erro{};
class Cliente
{
    string nome;
    int nItens; // numero de itens de compras
    double valorCompras;
public:
    Cliente(string n, int itens, double valor) { nome = n; nItens =
        itens, valorCompras = valor;}

friend class Mercearia;
friend ostream& operator<< (ostream& os, const Cliente& cl);
};

ostream& operator<< (ostream& os, const Cliente& cl)
{
    os << "nome: " << cl.nome << " nItens: " << cl.nItens << " total
    " << cl.valorCompras;
}

class Mercearia
{
    list<Cliente> filaComPrioridade;
    list<Cliente> filaNormal;

public:
    /*...*/

    bool clientesEmEspera();
    void insereCliente (string nome, int itens, double valor, string
        fila); // fila = "normal" ou "prioritaria"
    Cliente proximoClienteServir ();
};
```

a) Implemente o membro-função

```
void insereCliente (string nome, int itens, double valor, string
    fila);
```

Este membro-função insere o cliente na fila indicada (fila pode tomar dois valores: “normal” ou “prioritaria”). Contudo um cliente com mais de 10 itens de compras vai **sempre** para a fila normal.

Implemente ainda o membro-função

```
bool clientesEmEspera();
```

que retorna true se ainda existir algum cliente à espera de ser atendido.

b) Implemente o membro-função

```
Cliente Mercearia::proximoClienteServir ();
```

Este membro retorna o próximo cliente a ser servido e remove-o da fila respectiva.

Caso não existam clientes à espera deve ser lançada uma exceção do tipo `Erro`.

c) Implemente o membro-função

```
double valorNegocio();
```

Este membro retorna a soma dos valores das compras de todos os clientes que foram para as filas (mesmo que ainda não estejam atendidos). Altere convenientemente a classe `Mercearia` de forma a simplificar este método. No programa principal, leia sucessivamente do `standard input` a informação de um cliente e insira-o na fila. A leitura de clientes deve ser terminada com `ctrl-D`. Finalmente, imprima o valor do negócio.



Programação 2

2º Semestre

Trabalho Prático P3A

Efectue as tarefas de programação descritas abaixo, usando a linguagem C++ em ambiente Linux.

Grupo 4

Pretende-se implementar uma classe para modelar um contorno no plano 2D. O contorno será representado por uma lista de pontos.

```
class Erro{};
class Point2D
{
    double x, y;
public:
    Point2D(double xx, double yy)
    {x = xx, y=yy;}
    Point2D()
    {x = 0.0, y=0.0; }
    friend class SequenciaDePontos;
};

class SequenciaDePontos
{
    list<Point2D> contorno;

public:
    void imprimeContorno (ostream& os)
    {
        for (list<Point2D>::iterator it = contorno.begin(); it !=
            contorno.end(); it++)
        {
            os << "("<<(*it).x << ";" << (*it).y << ")"<<endl;
        }
    }
    /*...*/
};
```

a) Implemente o membro-função

```
void push(Point2D p);
```

da classe SequenciaDePontos para acrescentar ao fim da lista contorno o ponto p. Se o ponto a acrescentar não for vizinho do anterior na lista, deve ser lançada uma excepção do tipo Erro. Dois pontos são vizinhos se a distância entre eles é inferior a 2.

b) Implemente o membro-função

```
Point2D centroDeMassa ();
```

da classe SequenciaDePontos para determinar o centro de massa do contorno. Se o contorno se encontrar vazio deve ser lançada uma excepção do tipo Erro.

c) Implemente o membro-função

```
void suavizaContorno ();
```

da classe SequenciaDePontos para suavizar o contorno. Cada ponto na lista deve ser substituído pela média dele com o anterior.

