FEITO POR : Raiky Prezotto Pereira Sahb Novaes

QUESTÃO NÚMERO 1 :

Primeiro grupo:

A. four - Pode ser usado. É um nome válido para variáveis, classes, atributos e métodos.

B. for - Não pode ser usado. "for" é uma palavra-chave reservada em muitas linguagens de programação e não pode ser usada como um identificador.

C. from - Não pode ser usado. "from" é outra palavra-chave reservada em muitas linguagens de programação e não pode ser usada como um identificador.

D. 4 - Não pode ser usado. Os identificadores não podem começar com números em muitas linguagens de programação.

E. FOR - Pode ser usado, mas não é uma prática recomendada usar identificadores em maiúsculas para variáveis, atributos ou métodos. É mais comum usar maiúsculas para nomes de classes.

Segundo grupo:

A. dia&noite - Não pode ser usado. O símbolo "&" geralmente não é permitido em identificadores em muitas linguagens de programação.

B. diaENoite - Pode ser usado. É um nome válido para variáveis, classes, atributos e métodos.

C. dia & noite - Não pode ser usado. O espaço em branco não é permitido em identificadores.

D. dia E noite - Não pode ser usado. O espaço em branco e o "E" maiúsculo não são permitidos em identificadores.

E. dia\_e\_noite - Pode ser usado. É um nome válido para variáveis, classes, atributos e métodos

Terceiro grupo:

A. contador - Pode ser usado. É um nome válido para variáveis, classes, atributos e métodos.

B. 1contador - Não pode ser usado. Os identificadores não podem começar com números em muitas linguagens de programação.

C. contador de linhas - Não pode ser usado. Geralmente, espaços em branco e caracteres especiais, como espaços, não são permitidos em identificadores.

D. Contador - Pode ser usado. É um nome válido para variáveis, classes, atributos e métodos, mas lembre-se de que a maiúscula inicial é diferente de "contador" com letra minúscula.

E. count - Pode ser usado. É um nome válido para variáveis, classes, atributos e métodos.

/----------------------------------------------------------/

QUESTÃO NÚMERO 2 :

O número de municípios de um estado do Brasil.

Tipo de dado: int

O nome de um estado do Brasil.

Tipo de dado: string

A população de um estado do Brasil.

Tipo de dado: long long int

A área do Brasil em quilômetros quadrados.

Tipo de dado: double

A população total do mundo.

Tipo de dado: long long int

O CEP de um endereço no Brasil.

Tipo de dado: string

O nome de uma rua em um endereço no Brasil.

Tipo de dado: string

A altura de uma pessoa em metros.

Tipo de dado: float ou double

O peso de uma pessoa em quilos.

Tipo de dado: float ou double

A temperatura corporal de uma pessoa.

Tipo de dado: float ou double

O sexo de uma pessoa.

Tipo de dado: char

A altura de uma pessoa em milímetros.

Tipo de dado: int

/----------------------------------------------------------/

QUESTÃO NÚMERO 3 :

A. Um valor do tipo boolean pode receber o valor numérico zero.

Verdadeiro. Em C++, o valor false é representado como zero e o valor true é representado como um valor diferente de zero.

B. Um valor do tipo float pode armazenar valores maiores do que os que podem ser armazenados por um valor do tipo long.

Falso. O tipo long tem maior capacidade de armazenamento em comparação com o tipo float. Um long pode armazenar inteiros maiores do que um float pode representar com precisão.

C. Podemos ter caracteres cujos valores sejam negativos.

Verdadeiro. Em C++, o tipo char pode representar valores de -128 a 127 (em um sistema que utiliza codificação ASCII). Portanto, valores negativos são possíveis.

D. O número de bytes ocupados por uma variável do tipo float depende do computador e do sistema operacional sendo usado.

Verdadeiro. O tamanho de um tipo de dado em C++ pode variar dependendo da plataforma e do compilador. Geralmente, em muitos sistemas, um float é representado com 4 bytes.

E. O tipo char pode ser usado para representar pares de caracteres, uma vez que variáveis desse tipo ocupam dois bytes na memória.

Falso. Em C++, o tipo char ocupa normalmente 1 byte na memória e é usado para representar um único caractere.

F. Os tipos de dados double e long não são equivalentes, apesar de variáveis desses tipos ocuparem o mesmo espaço na memória.

Verdadeiro. Embora ambos possam ocupar o mesmo espaço na memória (geralmente 8 bytes), double e long são tipos diferentes. double é um tipo de ponto flutuante que tem maior precisão do que long, que é um tipo inteiro longo. Portanto, eles têm propósitos diferentes e não são equivalentes.

/--------------------------------------------------------/

QUESTÃO NÚMERO 7 :

1 => O erro esta na sintaxe do nome da Classe , que não pode haver espaços em branco.

2 => O erro esta na nos tipos de retorno dos métodos 'maior' e 'menor', pois uma função int só pode retornar int e uma função void não pode retornar nada.

3 => O erro está no retorno do método valor, pois um retorno só pode retornar apenas um valor.

4 => O erro está na declaração dos atributos , pois não se pode declarar variáveis com nomes de valores numéricos como 1 , 2 e 3 .

5 => O erro está na no método amplitude , pois não se pode declarar um método dentro de outro;

/--------------------------------------------------------/

questões 04 , 05 , 06 , 20 :

#include <iostream>

#include <random>

#include<vector>

#include "L02Ex15.cpp"

using namespace std;

class Lampada

{

protected:

    string esgtadoDaLampada;

    int watts;

    Contador VezesDeLuzAcesa;

    // Pode ser usado string ou booleano para representar o estado Da Lampada

public:

    Lampada()

    {

        watts = (rand() % 40 ) + 10 ;

    }

    void Acende()

    {

        esgtadoDaLampada = "aceso";

        VezesDeLuzAcesa.incrementar();

    }

    void Apaga()

    {

        esgtadoDaLampada = "apagado";

    }

    void MostraEstado()

    {

        if (esgtadoDaLampada == "aceso")

            cout << "LIGADA " << endl;

        else

            cout << "DESLIGADA " << endl;

    }

    bool EstaLigada()

    {

        if (esgtadoDaLampada == "aceso")

            return true;

        return false;

    }

    bool Economia()

    {

        if(watts >= 40)

            return true;

        return false;

    }

    void VezesLigada()

    {

        VezesDeLuzAcesa.imprimirValor();

    }

};

int main()

{

    vector <Lampada> Lista;

    for(int i =0 ; i<10; i ++)

    {

        Lampada P;

        Lista.push\_back(P);

    }

    Lista[5].Acende();

    Lista[4].Acende();

    for(int i =0 ; i<10; i ++)

    {

        Lista[i].MostraEstado();

        cout<<" EH ECONOMICA : ";

        if(Lista[i].Economia())

        {

            cout<<" sim "<<endl;

        }

        else{

         cout<<" nao "<<endl;

        }

    }

}

/----------------------------------/

Questão 08 :

#include <iostream>

#include <vector>

using namespace std;

class *HoraAproximda*

{

protected:

    int hora,minutos;

public:

    void InicializaData(int *umaHora*, int *umMinuto*)

    {

        if (HoraValiada(*umaHora*,*umMinuto*))

        {

            hora = *umaHora*;

            minutos = *umMinuto*;

        }

        else

        {

            hora = 0;

            minutos = 0;

        }

    }

    bool HoraValiada(int *umaHora*, int *umMinuto*)

    {

        if ((*umaHora* >= 0) && (*umaHora* <= 23) && (*umMinuto* >= 1) && (*umMinuto* <= 59))

            return true;

        return false;

    }

    void MostraHora()

    {

        if(hora>9)

        {

                cout << "HORA EH : " << hora << ':' << minutos<<" h"<< endl;

        }

        else{

               cout << "HORA EH : 0" << hora << ':' << minutos<<" h"<< endl;

        }

    }

};

int main()

{

*HoraAproximda* hora;

    hora.InicializaData(10,35);

    hora.MostraHora();

}

/-----------------------------------------/

questão 09 :

#include <iostream>

#include <vector>

using namespace std;

class HoraPrecisa

{

protected:

    int hora, minutos, segundo, centesimoDeSegundo;

public:

    void InicializaData(int *umaHora*, int *umMinuto*, int *umSegundo*, int *umCentesimo*)

    {

        if (HoraValiada(umaHora,umMinuto,umSegundo,umCentesimo))

        {

            hora = umaHora;

            minutos = umMinuto;

            segundo = umSegundo;

            centesimoDeSegundo = umCentesimo;

        }

        else

        {

            hora = 0;

            minutos = 0;

            segundo = 0 ;

            centesimoDeSegundo = 0;

        }

    }

    bool HoraValiada(int *umaHora*, int *umMinuto*, int *umSegundo*, int *umCentesimo*)

    {

        if ((umaHora >= 0) && (umaHora <= 23) && (umMinuto >= 1) && (umMinuto <= 59) && (umSegundo >= 0 && umSegundo <= 59) && (umCentesimo >= 0 && umCentesimo <= 999))

            return true;

        return false;

    }

    void MostraHora()

    {

        if (hora > 9)

        {

            cout << "HORA EH : " << hora << ':' << minutos << ':' << segundo << ':' << centesimoDeSegundo << " h" << endl;

        }

        else

        {

            cout << "HORA EH : 0" << hora << ':' << minutos << " h" << endl;

        }

    }

};

int main()

{

    HoraPrecisa hora;

    hora.InicializaData(10, 35, 50, 150);

    hora.MostraHora();

}

/---------------------------------------------/

questão 10 :

#include <iostream>

#include <vector>

using namespace std;

struct *Pessoa*

{

*string* nome;

*string* telefone;

*string* email;

*string* endereco;

};

class *CadernoDeEnderecos*

{

private:

*Pessoa* x;

    vector<*Pessoa*> Lista;

public:

    // Construtor

    CadernoDeEnderecos(*string* *name*, *string* *fone*, *string* *e*, *string* *end*)

    {

        x.nome = *name*;

        x.telefone = *fone*;

        x.email = *e*;

        x.endereco = *end*;

        Lista.push\_back(x);

    }

    // Métodos para obter os dados

*string* getNome(int *idDaPessoaDoCaderno*)

    {

        return Lista[*idDaPessoaDoCaderno*].nome;

    }

*string* getTelefone(int *idDaPessoaDoCaderno*)

    {

        return Lista[*idDaPessoaDoCaderno*].telefone;

    }

*string* getEmail(int *idDaPessoaDoCaderno*)

    {

        return Lista[*idDaPessoaDoCaderno*].email;

    }

*string* getEndereco(int *idDaPessoaDoCaderno*)

    {

        return Lista[*idDaPessoaDoCaderno*].endereco;

    }

    // Método para imprimir os dados

    void imprimirDados()

    {

        for (int i = 0; i < Lista.size(); i++)

        {

            cout << "Nome: " << Lista[i].nome << endl;

            cout << "Telefone: " << Lista[i].telefone << endl;

            cout << "Email: " << Lista[i].email << endl;

            cout << "Endereço: " << Lista[i].endereco << endl;

        }

    }

};

int main()

{

*CadernoDeEnderecos* pessoa("João da Silva", "123456789", "joao@example.com", "Rua Principal, 123");

    pessoa.imprimirDados();

    return 0;

}

/----------------------------------------------/

Questão 12 ,13 ,14 :

#include <iostream>

#include <string>

using namespace std;

class *Livro* {

public:

*string* titulo;

*string* autor;

*string* editora;

    int anoPublicacao;

    int numPaginas;

};

class *LivroDeLivraria* : public *Livro* {

public:

    double preco;

    int estoque;

};

class *LivroDeBiblioteca* : public *Livro* {

public:

    int numCopiasDisponiveis;

    int dias\_emprestimo;

     // Método para emprestar livro

    bool emprestarLivro() {

        if (numCopiasDisponiveis > 0) {

            numCopiasDisponiveis--;

            return true;

        } else {

            return false;

        }

    }

};

int main()

{

}

/-------------------------------------------/

questão 15 :

#include <iostream>

using namespace std;

class *Contador*

{

private:

    int valor;

public:

    Contador() { valor = 0; }

    void zerar()

    {

        valor = 0;

    }

    void incrementar()

    {

        valor++;

    }

    void imprimirValor()

    {

        cout << "O valor do contador é " << valor << endl;

    }

};

/----------------------------------/

questão 16 ,22 ,23 :

#include <iostream>

#include <vector>

using namespace std;

enum *Meses*

{

    ZERO,

    JANEIRO,

    FEVEREIRO,

    MARCO,

    ABRIL,

    MAIO,

    JUNHO,

    JULHO,

    AGOSTO,

    SETEMBRO,

    OUTUBRO,

    NOVEMBRO,

    DEZEMBRO

};

class *Data*

{

protected:

    int dia, mes, ano;

public:

    void InicializaData(int *umDia*, int *umMes*, int *umAno*)

    {

        if (DataEhValida(*umDia*, *umMes*, *umAno*))

        {

            dia = *umDia*;

            mes = *umMes*;

            ano = *umAno*;

        }

        else

        {

            dia = 0;

            mes = 0;

            ano = 0;

        }

    }

    int getDia() { return dia; }

    int getMes() { return mes; }

    int getAno() { return ano; }

    bool DataEhValida(int *UmDia*, int *UmMes*, int *UmAno*)

    {

        bool pBissexto, pData;

        pBissexto = pData = false;

        if (*UmAno* % 4 == 0 || *UmAno* % 100 == 0 || *UmAno* % 400 == 0)

            pBissexto = true;

        if (*UmAno* > 0 && *UmAno* < 2023)

        {

            switch (*UmMes*)

            {

            case JANEIRO:

                if (*UmDia* > 0 && *UmDia* <= 31)

                    pData = true;

                break;

            case FEVEREIRO:

                if ((*UmDia* > 0 && *UmDia* <= 29 && pBissexto) || (*UmDia* > 0 && *UmDia* <= 28 && !pBissexto))

                    pData = true;

                break;

            case MARCO:

                if (*UmDia* > 0 && *UmDia* <= 31)

                    pData = true;

                break;

            case ABRIL:

                if (*UmDia* > 0 && *UmDia* <= 30)

                    pData = true;

                break;

            case MAIO:

                if (*UmDia* > 0 && *UmDia* <= 31)

                    pData = true;

                break;

            case JUNHO:

                if (*UmDia* > 0 && *UmDia* <= 30)

                    pData = true;

                break;

            case JULHO:

                if (*UmDia* > 0 && *UmDia* <= 31)

                    pData = true;

                break;

            case AGOSTO:

                if (*UmDia* > 0 && *UmDia* <= 31)

                    pData = true;

                break;

            case SETEMBRO:

                if (*UmDia* > 0 && *UmDia* <= 30)

                    pData = true;

                break;

            case OUTUBRO:

                if (*UmDia* > 0 && *UmDia* <= 31)

                    pData = true;

                break;

            case NOVEMBRO:

                if (*UmDia* > 0 && *UmDia* <= 30)

                    pData = true;

                break;

            case DEZEMBRO:

                if (*UmDia* > 0 && *UmDia* <= 31)

                    pData = true;

                break;

            default:

                break;

            }

        }

        if (pData)

            return true;

        return false;

    }

    void MostraData()

    {

        cout << "DATA EH : " << dia << '/' << mes << '/' << ano << endl;

    }

    static *Data* DuplicaData(*Data* *date*, *Data* *outra*)

    {

*outra*.InicializaData(*date*.dia, *date*.mes, *date*.ano);

        return *outra*;

    }

    bool VemAntes(*Data* *data1*, *Data* *data2*)

    {

        if (*data1*.ano < *data2*.ano)

        {

            return true;

        }

        else if (*data1*.ano > *data2*.ano)

        {

            return false;

        }

        else if ((*data1*.ano == *data2*.ano) && *data1*.mes < *data2*.mes)

        {

            return true;

        }

        else if ((*data1*.ano == *data2*.ano) && *data1*.mes > *data2*.mes)

        {

            return false;

        }

        else if ((*data1*.ano == *data2*.ano) && (*data1*.mes == *data2*.mes) && *data1*.dia < *data2*.dia)

        {

            return true;

        }

        else if ((*data1*.ano == *data2*.ano) && (*data1*.mes == *data2*.mes) && *data1*.dia > *data2*.dia)

        {

            return false;

        }

        else

        {

            return true;

        }

    }

};

int main()

{

*Data* aniversario;

    aniversario.InicializaData(11, 9, 2003);

    aniversario.MostraData();

*Data* outroAniversario;

    outroAniversario = *Data* ::DuplicaData(aniversario, outroAniversario);

    outroAniversario.MostraData();

}

/----------------------------------------------------/

questão 17 :

#include <iostream>

#include <string>

using namespace std;

class *PoligonoRegular* {

private:

    int numLados;

public:

    PoligonoRegular(int *numLados*) : numLados(*numLados*) {}

    void setNumLados(int *Lados*) {

        numLados = numLados;

    }

    int getNumLados() const {

        return numLados;

    }

*string* getNome() const {

        switch (numLados) {

            case 3:

                return "Triângulo";

            case 4:

                return "Quadrado";

            case 5:

                return "Pentágono";

            case 6:

                return "Hexágono";

            case 7:

                return "Heptágono";

            case 8:

                return "Octógono";

            case 9:

                return "Eneágono";

            case 10:

                return "Decágono";

            default:

                return "Polígono com número de lados inválido.";

        }

    }

};

/-------------------------------------------/

questão 18, 11 ,19 :

#include <iostream>

#include <vector>

using namespace std;

class Ponto2D {

private:

    double x;

    double y;

public:

    Ponto2D(double *x*, double *y*) : x(x), y(y) {}

    double getX() const {

        return x;

    }

    double getY() const {

        return y;

    }

    void desenhar(*vector*<*vector*<char>>& *matriz*) {

        int i = static\_cast<int>(y);

        int j = static\_cast<int>(x);

        matriz[i][j] = 'o';

    }

};

class Linha2D {

private:

    Ponto2D ponto1;

    Ponto2D ponto2;

public:

    Linha2D(const *Ponto2D*& *ponto1*, const *Ponto2D*& *ponto2*) : ponto1(ponto1), ponto2(ponto2) {}

    void desenhar(*vector*<*vector*<char>>& *matriz*) {

        ponto1.desenhar(matriz);

        ponto2.desenhar(matriz);

        double x1 = ponto1.getX();

        double y1 = ponto1.getY();

        double x2 = ponto2.getX();

        double y2 = ponto2.getY();

        int i1 = static\_cast<int>(y1);

        int j1 = static\_cast<int>(x1);

        int i2 = static\_cast<int>(y2);

        int j2 = static\_cast<int>(x2);

        // Desenhar a linha usando o algoritmo de Bresenham

        int deltaX = abs(j2 - j1);

        int deltaY = abs(i2 - i1);

        int sX = j1 < j2 ? 1 : -1;

        int sY = i1 < i2 ? 1 : -1;

        int erro = deltaX - deltaY;

        while (i1 != i2 || j1 != j2) {

            matriz[i1][j1] = 'x';

            int erroDuplo = erro \* 2;

            if (erroDuplo > -deltaY) {

                erro -= deltaY;

                j1 += sX;

            }

            if (erroDuplo < deltaX) {

                erro += deltaX;

                i1 += sY;

            }

        }

    }

};

class Retangulo {

private:

    Ponto2D ponto1;

    Ponto2D ponto2;

public:

    Retangulo(const *Ponto2D*& *ponto1*, const *Ponto2D*& *ponto2*) : ponto1(ponto1), ponto2(ponto2) {}

    void desenhar(*vector*<*vector*<char>>& *matriz*) {

        ponto1.desenhar(matriz);

        ponto2.desenhar(matriz);

        double x1 = ponto1.getX();

        double y1 = ponto1.getY();

        double x2 = ponto2.getX();

        double y2 = ponto2.getY();

        int i1 = static\_cast<int>(y1);

        int j1 = static\_cast<int>(x1);

        int i2 = static\_cast<int>(y2);

        int j2 = static\_cast<int>(x2);

        // Desenhar linhas horizontais

        for (int j = j1; j <= j2; ++j) {

            matriz[i1][j] = 'x';

            matriz[i2][j] = 'x';

        }

        // Desenhar linhas verticais

        for (int i = i1; i <= i2; ++i) {

            matriz[i][j1] = 'x';

            matriz[i][j2] = 'x';

        }

    }

};

int main() {

    const int altura = 10;

    const int largura = 10;

    vector<vector<char>> matriz(altura, vector<char>(largura, '.'));

    Ponto2D pontoA(2.5, 3.5);

    Ponto2D pontoB(8.5, 6.5);

    Linha2D linha(pontoA, pontoB);

    Ponto2D pontoC(4.5, 2.5);

    Ponto2D pontoD(7.5, 8.5);

    Retangulo retangulo(pontoC, pontoD);

    linha.desenhar(matriz);

    retangulo.desenhar(matriz);

    for (int i = 0; i < altura; ++i) {

        for (int j = 0; j < largura; ++j) {

            cout << matriz[i][j] << ' ';

        }

        cout << '\n';

    }

    return 0;

}

/------------------------------------------------/

questão 21 , 24 :

#include <iostream>

#include <iomanip>

using namespace std;

class ContaBancariaSimplificada

{

protected:

    string nomeDoCorrentista;

    float saldo;

    bool contaEhEspecial;

public:

    void AbreConta(*string* *name*, float *deposito*, bool *especial*)

    {

        if ((name.size() > 0 && name.size() <= 120) && deposito > 0.0)

        {

            nomeDoCorrentista = name;

            saldo = deposito;

            contaEhEspecial = especial;

            cout << "Conta Criada" << endl;

        }

        else

        {

            cout << "Falha ao criar conta! Dados errados!" << endl;

        }

    }

    void AbreContaSimples(*string* *name*)

    {

        if ((name.size() > 0 && name.size() <= 120))

        {

            nomeDoCorrentista = name;

            saldo = 0.00;

            contaEhEspecial = false;

            cout << "Conta Criada" << endl;

        }

        else

        {

            cout << "Falha ao criar conta! Dados errados!" << endl;

        }

    }

    void Depositar(float *deposito*)

    {

        if (deposito > 0.00)

        {

            saldo += deposito;

        }

        else

        {

            cout << "Valor incorreto" << endl;

        }

    }

    void Retira(float *valor*)

    {

        if (!contaEhEspecial)

        {

            if (valor > 0.00 && valor<=saldo)

            {

                saldo -= valor;

            }

            else

            {

                cout << "Valor incorreto" << endl;

            }

        }

        else

        {

            if(valor > 0.00)

            {

                saldo-=valor;

            }

            else

            {

                cout << "Valor incorreto" << endl;

            }

        }

    }

    void MostraDados()

    {

        cout<<"O nome do correntista eh : "<<nomeDoCorrentista<<endl;

        cout<<"O Saldo eh : "<<fixed<<setprecision(2)<<saldo<<endl;

        if(contaEhEspecial)cout<<"A conta eh espacial"<<endl;

        else cout<<"A conta nao eh espacial"<<endl;

    }

};

int main()

{

    ContaBancariaSimplificada conta;

    conta.AbreConta("Raiky",1500.00,1);

    conta.MostraDados();

    conta.Retira(500);

    conta.MostraDados();

}

/---------------------------------------/

Questão 25 :

#include <iostream>

#include <vector>

using namespace std;

class MesaDeRestaurante {

private:

    vector<int> pedidos;

public:

    void adicionaAoPedido(int *quantidade*, int *item*) {

        if (item >= 0 && item < pedidos.size()) {

            pedidos[item] += quantidade;

        } else {

            pedidos.push\_back(quantidade);

        }

    }

    void zeraPedidos() {

        for (int i = 0; i < pedidos.size(); i++) {

            pedidos[i] = 0;

        }

    }

    float calculaTotal(*vector*<float> *precos*) {

        float total = 0.0;

        for (int i = 0; i < pedidos.size(); i++) {

            total += pedidos[i] \* precos[i];

        }

        return total;

    }

};

class RestauranteCaseiro {

private:

    vector<MesaDeRestaurante> mesas;

public:

    RestauranteCaseiro(int *numMesas*) : mesas(numMesas) {}

    void adicionaAoPedido(int *quantidade*, int *item*, int *mesa*) {

        if (mesa >= 0 && mesa < mesas.size()) {

            mesas[mesa].adicionaAoPedido(quantidade, item);

        } else {

            cout << "Mesa inválida." << endl;

        }

    }

    float calculaTotalMesa(int *mesa*, *vector*<float> *precos*) {

        if (mesa >= 0 && mesa < mesas.size()) {

            return mesas[mesa].calculaTotal(precos);

        } else {

            cout << "Mesa inválida." << endl;

            return 0.0;

        }

    }

};

int main() {

    vector<float> precos = {10.0, 20.0, 15.0}; // Preços dos itens

    RestauranteCaseiro restaurante(5);

    restaurante.adicionaAoPedido(2, 0, 0); // Adiciona 2 itens do tipo 0 à mesa 0

    restaurante.adicionaAoPedido(1, 1, 0); // Adiciona 1 item do tipo 1 à mesa 0

    restaurante.adicionaAoPedido(3, 1, 1); // Adiciona 3 itens do tipo 1 à mesa 1

    restaurante.adicionaAoPedido(1, 2, 1); // Adiciona 1 item do tipo 2 à mesa 1

    cout << "Total da mesa 0: " << restaurante.calculaTotalMesa(0, precos) << endl;

    cout << "Total da mesa 1: " << restaurante.calculaTotalMesa(1, precos) << endl;

    return 0;

}

/\*\*Vantagens de criar classes desta forma:

Organização: As classes MesaDeRestaurante e RestauranteCaseiro permitem organizar o código de forma mais modular e legível.

Reutilização: É fácil reutilizar a lógica de manipulação de mesas em diferentes contextos ou projetos.

Encapsulamento: A lógica de manipulação de pedidos está encapsulada nas classes correspondentes, o que facilita a manutenção e evita efeitos colaterais inesperados.

Desvantagens:

Rigidez no número de mesas: A implementação fixa um número predeterminado e imutável de instâncias de MesaDeRestaurante. Isso pode ser problemático se o restaurante desejar alterar o número de mesas em tempo de execução.

Sugestões de melhoria:

Poderia ser adicionado tratamento de erros para situações como tentar adicionar a um pedido um item inválido ou uma quantidade negativa.

A implementação atual assume que os preços dos itens estão em um vetor chamado precos. Se os preços podem variar, seria bom adicionar uma forma de definir esses preços dinamicamente.

\*/