Residência em Software

Revisão OO & Métodos

Professores:
Álvaro Coelho, Edgar Alexander, Esbel Valero e Hélder Almeida

INSTITUIÇÃO EXECUTORA
cepedi UESC
COORDENADORA
MCTI FUTURO Softex
APOIO
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Implementar os pontos no R^3

Partindo do exemplo dos slides 15 a 17
da aula 19

```
int main () {
    Poligono poli;
    //lendo os pontos
    poli.le_pontos();
    //listando os pontos
    poli.lista_pontos();
    // calculando o perímetro
    cout << endl << "Perimetro calculado: " <<
            poli.perimetro() << endl;
}
```

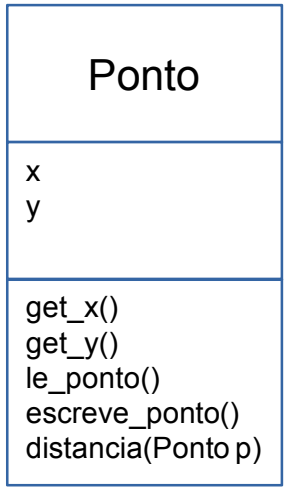
Poligono
vector<ponto> pontos
le_pontos() lista_pontos(); perimetro()

Classe Poligono

```
class Poligono {
    vector<Ponto> pontos;

    public:
    void le_pontos() {
        cout << "Criando um poligono!" << endl;
        char sn;
        do {
            Ponto p;
            p.le_Ponto();
            pontos.push_back(p);
            cout << "Deseja inserir mais pontos (s/n)?";
            cin >> sn;
        } while (sn!='n');
    }

    void lista_pontos() {
        cout << "As coordenadas digitadas foram" << endl;
        for (Ponto p: pontos)
            cout << "(" << p.escreve_ponto() << ") ";
    }

    float perimetro() {
        float per = 0;
        vector<Ponto>::iterator it2;
        Ponto p1;
        Ponto p2;
        for (auto it = pontos.begin();
                  it != pontos.end()-1; it++) {
            it2 = it;
            advance(it2,1);
            p1 = *it;
            p2 = *it2;
            per += p1.distancia(p2);
        }
        //pegando distancia entre o primeiro e ultimo
        it2 = pontos.begin();
        p1 = *it2;
        per += p1.distancia(p2);
        return per;
    }
};
```

Classe ponto

```
class Ponto {
    float x, y;
    public:

    float get_x() {
        return x;
    }

    float get_y() {
        return y;
    }

    void le_Ponto() {
        cout << "Digite as coordenadas do ponto: ";
        cin >> x >> y;
    }

    string escreve_ponto() {
        return to_string(x) + " , " + to_string(y);
    }

    float distancia(Ponto p1) {
        return sqrt( pow(x - p1.get_x(),2) +
                     pow (y - p1.get_y(),2)  );
    }
};
```

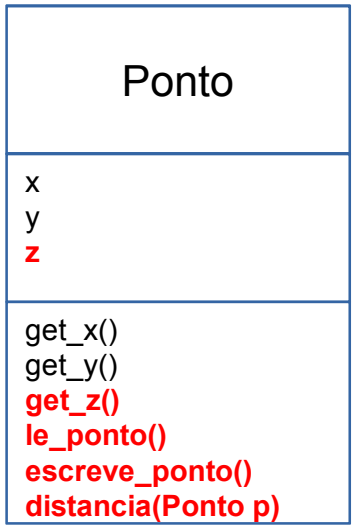


Residência em Software

- Com esta nova estrutura: modifique a classe ponto para refletir um ponto no R^3

$$d = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)}$$

Onde haverão mudanças?



O Desafio

- Implementar sobrecarga de operadores para polígonos
  - Operador = → Cria um polígono com os mesmos pontos de outro
    - p1 = p2; // p1 terá o mesmo conjunto de pontos de p2
  - Operador == → verifica se dois polígonos são iguais
    - Se eles contém exatamente os mesmos pontos, e na mesma ordem*
    - p1==p2 //será *true* se os pontos de p1 e p2 coincidirem (inclusive na mesma posição), ou *false* caso contrário

*Considere que os pontos serão inseridos de forma que SEMPRE formarão um polígono

Nossas classes

Ponto
x y
get_x() get_y() get_z() le_ponto() escreve_ponto() distancia(Ponto p) op==(Ponto p) op+(Ponto p)

Poligono
vector<ponto> pontos
le_pontos() lista_pontos(); perimetro()

Objetivo

Ponto
x y
get_x() get_y() get_z() le_ponto() escreve_ponto() distancia(Ponto p) op==(Ponto p) op+(Ponto p)

Poligono
vector<ponto> pontos
le_pontos() lista_pontos(); perimetro() **op=(Poligono p)** **op==(Poligono p)** **op+(Poligono P)**

Precisaremos de uma função main()

- Para testes
- Eventualmente para funcionalidade
- O que precisaremos implementar?
  - Criar um polígono
  - Criar um segundo polígono
  - Verificar se são iguais
  - Atribuir um deles a um terceiro polígono

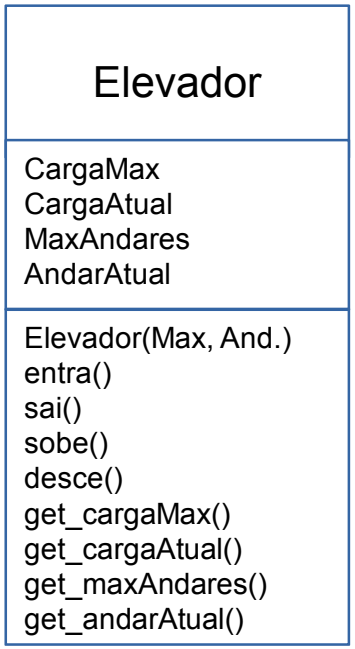
Precisaremos de uma função main()

- Para testes
- Eventualmente para funcionalidade
- O que precisaremos implementar?
  - Criar um polígono
  - Criar um segundo polígono
  - Verificar se são iguais
  - Atribuir um deles a um terceiro polígono

```
int main () {
    Poligono poli1, poli2, poli3;
    //lendo os pontos de poli1
    poli1.le_pontos();
    //lendo os pontos de poli2
    poli2.le_pontos();
    //testa se sao iguais
    if (poli1==poli2)
        cout << "Os poligonos sao iguais" << endl;
    else
        cout << "Os poligonos sao diferentes" << endl;
    //cria poli3 = poli1
    poli3 = poli1;
    cout << "Testando Se copiou" << endl;
    poli3.lista_pontos();
    cout << endl;
    poli1.lista_pontos();
}
```

Mais exercícios

- Uma classe Elevador
- Atributos: Carga máxima (no de pessoas, Carga atual (no de pessoas), MaxAndares, AndarAtual
- Métodos
  - Construtor (Carga Máxima, no. De andares)
  - Entra: aumenta uma pessoa (se for possível)
  - Sai: remove uma pessoa (se for possível)
  - Sobe: vai um andar acima (se for possível)
  - Desce: vai um andar abaixo, se for possível.
  - Gets dos atributos.



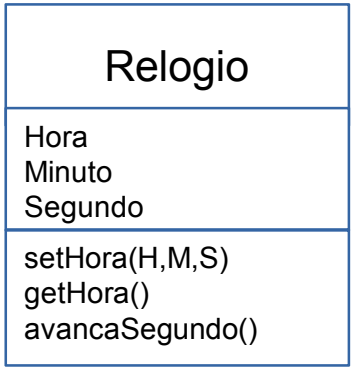
Segue a função main()

- Uma classe Elevador
- Atributos: Carga máxima (no de pessoas, Carga atual (no de pessoas), MaxAndares, AndarAtual
- Métodos
  - Construtor (Carga Máxima, no. De andares)
  - Entra: aumenta uma pessoa (se for possível)
  - Sai: remove uma pessoa (se for possível)
  - Sobe: vai um andar acima (se for possível)
  - Desce: vai um andar abaixo, se for possível.
  - Gets dos atributos.

```
int main () {
    Elevador e = Elevador(10,20);//max de 10 pessoas, 20 andares
    char op;
    do {
        cout << "Estamos com "
            << e.get_cargaAtual() << " passageiros" << endl;
        cout << "Andar: " << e.get_andarAtual() << endl;
        cout << "U: sobe" << endl;
        cout << "D: Desce" << endl;
        cout << "E: Entra pessoa" << endl;
        cout << "S: Sai pessoa" << endl;
        cout << "F> Fim do programa" << endl;
        cin >> op;
        switch (op)
        {
            case 'U':
                e.sobe();
            break;
            case 'D':
                e.desce();
            break;
            case 'E':
                e.entra();
            break;
            case 'S':
                e.sai();
            break;
            case 'F':
                cout << "Encerrando...";
            break;
           default:
                cout << "Opcao invalida!!!";
        }
    } while (op!='F');

}
```

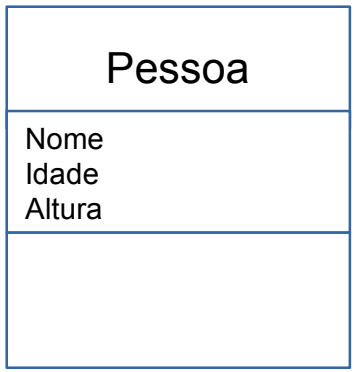
2º exercício

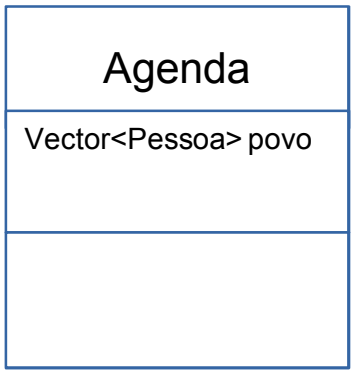
- Classe Relogio
- Atributos: Hora, Minuto, Segundo (devidamente limitados: 0-23, 0-59 e 0-59)
- Métodos
  - setHora(H,M,S): verifica se é uma hora válida e seta o relógio se for o caso
  - getHora(): retorna um string com a hora no formato HH:MM:SS
  - avancaSegundo(): avança 1 segundo (observe os limites)



3º exercício

- Dada a função main()
- Dada a descriçao das classes
- Implemente-as






```
#include <iostream>
#include <string>>

using namespace std;

int main () {
    string linha = "------------
        ------------------------------------";
    Agenda A;

    A.armazenaPessoa("Abel", 22, 1.78);
    A.armazenaPessoa("Tiago", 20, 1.80);
    A.imprimePovo();
    cout << linha << endl;

    int posicao = A.buscaPessoa("Tiago");
    if (posicao > 0)
        A.imprimePessoa(posicao);
    cout << linha << endl;

    A.removePessoa("Tiago");
    A.imprimePovo();
    cout << linha << endl;


    return 0;
}
```

4º exercício

- O exercício anterior tem falhas
  - Mudanças na classe Pessoa geram muitos efeitos colaterais!
- Melhore (desde a função main())
  - Minimizando o acoplamento (use encapsulamento)