CIÊNCIA DA COMPUTAÇÃO
PROGRAMAÇÃO ORIENTADA A OBJETOS
PROF. JUDSON SANTIAGO

LABORATÓRIO 11

PROJETANDO CLASSES

EXERCÍCIOS DE REVISÃO

VOCÊ DEVE ACOMPANHAR PARA REVISAR OS CONCEITOS IMPORTANTES

1. O programador C++ tem bastante flexibilidade na hora de projetar suas classes. Para exercitar esta característica da linguagem, modifique a classe Vetor vista na última aula, substituindo todas as operações, que foram implementadas com sobrecargas de operadores e funções amigas, por métodos convencionais.

```
class Vetor
{
public:
    enum Coord { RET, POL };

private:
    double x, y;      // coordenadas cartesianas
    double mag;       // comprimento do vetor
    double ang;       // ângulo do vetor em graus
    Coord rep;        // representação padrão

    void SetMag();    // ajusta magnitude com base em (x,y)
    void SetAng();    // ajusta ângulo com base em (x,y)
    void SetX();      // ajusta posição x com base em magnitude e ângulo
    void SetY();      // ajusta posição y com base em magnitude e ângulo

public:
    Vetor();
    Vetor(double n1, double n2, Coord tipo = RET);

    double Magnitude() { return mag;  }
    double Angulo() { return ang; }
    void SetCoord(Coord modo) { rep = modo; }

    Vetor operator+(const Vetor& v) const;
    Vetor operator-(const Vetor& v) const;
    Vetor operator-() const;
    Vetor operator*(double n) const;
    friend Vetor operator*(double n, const Vetor& v);
    friend ostream& operator<<(ostream& os, const Vetor& v);
};
```

Construa um programa para testar a criação de objetos e os métodos da classe. Quais as vantagens e desvantagens de cada abordagem?

2. Modifique a classe Vetor, desenvolvida na última aula, para que ela armazene em seus atributos apenas a representação por coordenadas retangulares. Acrescente métodos para converter e retornar a representação do vetor em coordenadas polares.

```
class Vetor
{
private:
   double x;             // coordenada cartesiana horizontal
   double y;             // coordenada cartesiana vertical

public:
    Vetor();
    Vetor(double n1, double n2);

    double Magnitude();  // retorna magnitude (coordenada polar)
    double Angulo();     // retorna ângulo (coordenada polar)

    Vetor operator+(const Vetor& v) const;
    Vetor operator-(const Vetor& v) const;
    Vetor operator-() const;
    Vetor operator*(double n) const;
    friend Vetor operator*(double n, const Vetor& v);
    friend ostream& operator<<(ostream& os, const Vetor& v);
};
```

3. Agora modifique a classe Vetor para que ela armazene em seus atributos apenas a representação por coordenadas polares. Acrescente métodos para converter e retornar a representação do vetor em coordenadas retangulares.

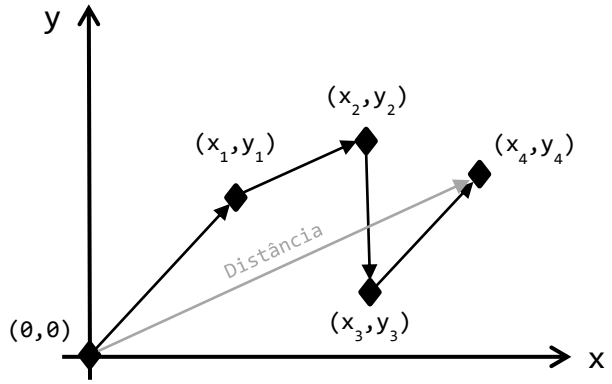
```
class Vetor
{
private:
   double mag;       // comprimento do vetor
   double ang;       // ângulo do vetor em graus

public:
    Vetor();
    Vetor(double n1, double n2);

    double X();      // coordenada retangular horizontal
    double Y();      // coordenada retangular vertical

    Vetor operator+(const Vetor& v) const;
    Vetor operator-(const Vetor& v) const;
    Vetor operator-() const;
    Vetor operator*(double n) const;
    friend Vetor operator*(double n, const Vetor& v);
    friend ostream& operator<<(ostream& os, const Vetor& v);
};
```

4. Com base na experiência adquirida pelas implementações feitas nas questões anteriores, qual seria a melhor forma de implementar a classe Vetor e quais as vantagens e desvantagens de cada abordagem?

EXERCÍCIOS DE FIXAÇÃO

VOCÊ DEVE FAZER OS EXERCÍCIOS PARA FIXAR O CONTEÚDO

1. Construa um programa para simular o "voo da mosca". A mosca voa em uma direção por um certo tempo, depois muda de direção, voa por mais um tempo e repete esse processo indefinidamente. A direção da mosca pode ser definida por um ângulo aleatório (0 a 359 graus) e o seu tempo de voo por uma magnitude aleatória (0 a 9) de um vetor. A imagem abaixo mostra o que seria o comportamento típico de uma mosca.



Escolha uma das versões da classe Vetor implementadas nos exercícios anteriores para simular o voo da mosca. O programa deve ler do usuário uma distância e exibir quantos passos de simulação foram necessários até a mosca atingir aquela distância a partir do seu ponto de origem. O quadro abaixo mostra um exemplo.

```
Entre com a distância desejada: 100
-----------------------------------------------
Foram 288 passos de simulação para chegar em:
(x,y) = (23.7105, 97.8239)
(m,a) = (100.656, 76.3754)
-----------------------------------------------
Distância média por passo: 0.347222
```

2. Modifique o programa anterior de forma a registrar cada passo da simulação em um arquivo texto, como mostrado no exemplo abaixo:

```
Distância alvo: 100
1: (x,y) = (2.83656,-0.976704)
2: (x,y) = (4.69092, -0.227491)
3: (x,y) = (5.68849, -0.157735)
...
279: (x,y) = (-87.2427, 51.2761)
-----------------------------------------------
Foram 279 passos de simulação para chegar em:
(x,y) = (-87.2427, 51.2761)
(m,a) = (101.195, 149.555)
-----------------------------------------------
Distância média por passo: 0.358423
```

EXERCÍCIOS DE APRENDIZAGEM

VOCÊ DEVE ESCREVER PROGRAMAS PARA REALMENTE APRENDER

1. Um número complexo é um número z que pode ser escrito na forma $z = x + yi$, sendo x e y números reais e i denotando a parte imaginária. Construa uma classe para representar números complexos com métodos para realizar as seguintes operações:

   Soma: $(a + bi) + (c + di) = (a + c) + (b + d)i$
   Subtração: $(a + bi) - (c + di) = (a - c) + (b - d)i$
   Multiplicação: $(a + bi)(c + di) = (ac - bd) + (bc + ad)i$
   Multiplicação por real: $n(c + di) = (nc) + (nd)i$
   Conjugado: $\sim(a + bi) = (a - bi)$

   Construa um programa para testar a classe e as operações sobre complexos.
   A saída deve ser como no exemplo abaixo.

```
Digite dois números complexos:
3+4i
10+12i

Complexo b: 10+12i
Seu conjugado: 10-12i
Complexo a: 3+4i
a + b : 13+16i
a – b : -7-8i
a * b : -18+76i
2 * b : 20+24i
```

   Dica: `os.setf(ios::showpos)` sempre mostra o sinal +/- para valores decimais.

2. Altere o programa da questão anterior para que todas as operações sejam feitas através de funções sobrecarregadas e função amigas. O código abaixo mostra um exemplo que deve funcionar com a sua classe Complexo.

```
int main()
{
    cout << "Digite dois números complexos: ";
    Complexo a, b;
    cin >> a >> b;

    cout << "Complexo b: " << b << endl;
    cout << "Seu conjugado: " << ~b << endl;
    cout << "Complexo a: " << a << endl;
    cout << "a + b : " << a + b << endl;
    cout << "a - b : " << a - b << endl;
    cout << "a * b : " << a * b << endl;
    cout << "2 * b : " << 2 * b << endl;

    return 0;
}
```