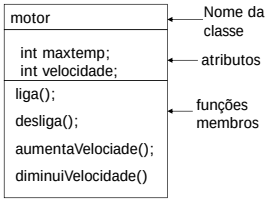
Estrutura de Dados

Profa.: Marcela
Revisão C++

Classe

- É o molde que será usado para criar objetos.
- A classe envolve, associa, funções e dados, controlando o acesso a estes. Definir uma classe implica em especificar os seus atributos (dados) e suas funções membro (código).
- Suponha que você está fazendo um sistema para controlar motores. Então você vai fazer um molde para os objetos motor do seu sistema. Esse molde é a classe.

Classe



Exercícios

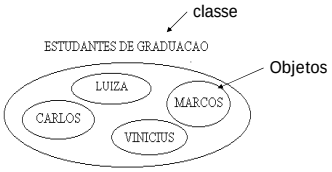
1)Lembre-se de algum programa em que você trabalhou, cite que tipos de classes seriam criadas se esse programa fosse escrito em C++, que atributos e que funções membro estariam associadas a esses objetos?

2) Como seria a definição de uma classe Conta que trata dos dados e operações em conta de um banco?

3) Como seria a definição de uma classe para estudante de graduação?

Objetos

- Objetos são instâncias de uma classe.
- Quando um objeto é criado ele precisa ser inicializado.



Atributos ou dados membros

```
#include
<iostream.h>
class circulo{
  public:
        float raio;
        float x;
        float y;
};
```

```
int main(){
        circulo ac;
        ac.raio=10.0;
        ac.x=1.0;
        ac.y=1.0;
        cout << "Raio:"<<ac.raio <<"\n";
                return (0);
}
```

Este código é a declaração da classe círculo. Entre chaves vem os dados membro e as funções membro (que não foram apresentadas ainda).
O acesso aos dados membro deve ser feito usando o nome do objeto e o nome do dado membro, separados por um ponto.

Atributos ou dados membros

- **Exercícios:**
- 1) Repita o mesmo exemplo só que agora mova o círculo alterando as componentes x e y. Ponha o círculo em (0.0,0.0). Através de atribuições do tipo ac.x=1.0; mova o círculo para (1.0,1.0).
- 2)Simplifique o programa anterior retirando o atributo raio. Você pode dar o nome de ponto ou ponto_geometico para esta classe.

Métodos ou Funções Membro

- C++ permite que se acrescente funções de manipulação da classe em sua declaração, juntando tudo numa só entidade que é uma classe.
- Essas funções membro podem ter sua declaração (cabeçalho) e implementação (código) dentro da classe ou só o cabeçalho (assinatura) na classe e a implementação, código, fora.
- Essas funções compõem a interface da classe. A terminologia usada para designá-las é bastante variada: funções membro, métodos, etc. Quando uma função membro é chamada, se diz que o objeto está recebendo uma mensagem (para executar uma ação).

Métodos ou Funções Membro

```
#include <iostream.h>
class contador {
 public:
    int num;
    void incrementa(void){
       num=num+1;
    }
    void comeca(void){
       num=0;
    }
};
```

```
int main() {
    contador a;
    a.comeca();
    cout << a.num << "\n";
    a.incrementa();
    cout << a.num << "\n";
}
```


Métodos ou Funções Membro

```
#include <iostream.h>
class circulo
{  public:
          float raio;
          float x;
          float y;
   void move(float dx,float dy) {
          x+=dx;
          y+=dy;
    }
```

1

```
void mostra(void) {
          cout << "Raio:"<<raio <<endl;
          cout << "X:"<<x << endl;
          cout << "Y:" <<y<< endl;
}
};
```

2

```
int main()
{  circulo ac;
   ac.x=0.0;   ac.y=0.0;  ac.raio=10.0;
   ac.mostra();
   ac.move(1.0,1.0);
   ac.mostra(); }
```

3

- Exercícios 1)Neste mesmo programa, crie uma função para a classe chamada "inicializa" que deve ter como argumentos um valor para x, um para y e outro para o raio.
- **FUNÇÕES MEMBRO QUE RETORNAM VALORES.**
  - Até agora só tínhamos visto funções membro com valor de retorno igual a void. Uma função membro, assim como uma função comum, pode retornar qualquer tipo, inclusive os definidos pelo usuário.

```
class meunumero{
      public:
          int num;
      int getnum(){

return(num);
      }
}
```


FUNÇÕES DECLARADAS EXTERNAS A CLASSE ,

- O programador não é obrigado a implementar as funções membro dentro da declaração da classe, basta defini-las e apresentar a implementação em separado segundo a sintaxe (compilável) descrita a seguir:

```
#include <iostream.h>
class teste
{ public:
  int x;
  void altera_x(int v);
};
void teste::altera_x(int v) {
 x=v;
}
int main(){
  teste a;
  a.altera_x(10);
  cout << a.x;
}
```


FUNÇÕES MEMBRO CHAMAMANDO FUNÇÕES MEMBRO

```
#include <iostream.h>
class motor
{ public:
  int veloc;
  int tempo;
  int obterDistancia();
  int obterConsumo(int kmlitro);

};

int motor::obterDistancia(){
          return veloc* tempo;
  }
```

```
int motor::obterConsumo(int
kmlitro){
    return obterDistancia()/kmlitro;
  }

int main(){
  motor m;;
  m.veloc = 100;
  m.tempo = 2;
  cout << m.obterConsumo(17.0);
}
```

Observe não é necessário usar o **.** na chamada de obterDistancia().

Construtores

- Construtores são funções membro especiais chamadas pelo sistema no momento da criação de um objeto. Elas não possuem valor de retorno, porque você não pode chamar um construtor para um objeto.
- Contrutores representam uma oportunidade de inicializar de forma organizada os objetos, imagine se você esquece de inicializar corretamente ou o faz duas vezes, etc.
- Um construtor tem sempre o mesmo nome da classe e não pode ser chamado pelo usuário desta.
- Na classe anterior poderíamos ter o seguinte construtor:

```
motor(int v, int t){
        veloc = v;
        tempo = t;
 }
```

A chamada a ele é feita na declaração do objeto:
motor m(100,2);

Por enquanto uso estático dos Objetos

13

Destrutores

- Análogos aos construtores, os destrutores também são funções membro chamadas pelo sistema, só que elas são chamadas quando o objeto sai de escopo ou em alocação dinâmica, tem seu ponteiro desalocado, ambas (construtor e destrutor) não possuem valor de retorno.
- Você não pode chamar o destrutor, o que você faz é fornecer ao compilador o código a ser executado quando o objeto é destruído, apagado. Ao contrário dos construtores, os destrutores não tem argumentos.

14

Destrutor

- A sintaxe do destrutor é simples, ele também tem o mesmo nome da classe só que precedido por ~ , ele não possui valor de retorno e seu argumento é void sempre:
- ~nomedaclasse(void) { /* Codigo do destrutor */ }
- Na classe motor adicione o seguinte destrutor:

```
~motor(){
  cout<<"\nO objeto esta sendo destruido";
 }
```

15

ENCAPSULAMENTO
Membros Privados e Públicos

Encapsulamento, “data hiding”. Neste tópico vamos falar das maneiras de restringir o acesso as declarações de uma classe, isto é feito em C++ através do uso das palavras reservadas public e private.

A

```
class ponto{
int x;
public:
    ponto();
    int muda(int y);
}
```

B

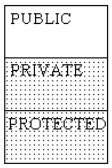
```
class ponto{
private:
    int x;
public:
    ponto();
    int muda(int y);
}
```

As declarações de classe A e B são equivalentes.

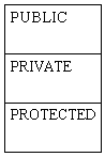
16

Visibilidade

Visibilidade das declarações de uma classe, fora dela e de sua hierarquia.
Só a parte public é visível neste caso:



Visibilidade das declarações de uma classe, dentro dela mesma:



17

Visibilidade

```
#include <iostream.h>
class ponto{
  private:
    float x;
    float y;
  public:
    void inicializa(float a,float b) {  x=a; y=b; }
    void mostra(void) {cout << "X:" << x << " , Y:" << y << endl;}
};
void main() {
ponto ap;
ap.inicializa(0.0,0.0);
ap.mostra();
}
```

Comentários:

Este programa não deixa você tirar o ponto de (0,0) a não ser que seja chamada inicializa novamente. Fica claro que agora, encapsulando x e y precisamos de mais métodos para que a classe não tenha sua funcionalidade limitada.

Novamente: escrever ap.x=10; em main é um erro! Pois x está qualificada como private.

18

Fazer outros exemplos

COMPILANDO UM PROGRAMA COM VÁRIOS ARQUIVOS

```
//Arquivo 1 ponto.h
class ponto{
private:
      float x;
public:
      void inicializa(float);
      void mostra(void);
};
```

```
//Arquivo 2 , ponto.cpp
#include <iostream.h>
#include "ponto.h"
void ponto::inicializa(float a) {
 x=a}
void ponto::mostra(void)
 {cout << "X:" << x << endl;}
```

```
////Arquivo 3 princ.cpp
 #include "ponto.h"
 void main(){
 ponto ap;
 ap.inicializa(0.0);
 ap.mostra();
 }
```

Normalmente os programas C++ são divididos em arquivos para melhor organização e encapsulamento, porém nada impede que o programador faça seu programa em um só arquivo.

19

COMPILANDO UM PROGRAMA COM VÁRIOS ARQUIVOS

- Os arquivos com extensão .h indicam header files. É costume deixar nesses arquivos somente a interface das classes e funções para que o usuário possa olhá-lo, como numa "library".
- O arquivo 2 é um arquivo de implementação. Ele tem o mesmo nome do arquivo ".h", embora isto não seja obrigatório. Este arquivo fornece o código para as operações da classe ponto, daí a declaração: #include "ponto.h". As aspas indicam que se trata de um arquivo criado pelo usuário.
- O arquivo 3 é o arquivo principal do programa. Observe que o arquivo 3 também declara #include "ponto.h", isto porque ele faz uso das dos tipos definidos em "ponto.h", porém "princ.cpp" não precisa declarar #include <iostream.h> porque este não usa diretamente as definições de iostream em nenhum momento, caso "princ.cpp" o fizesse, o include <iostream> seria necessário.

20

COMPILANDO UM PROGRAMA COM VÁRIOS ARQUIVOS

- O leitor encontrará em alguns dos exemplos seguintes as diretivas de compilação. Quando da elaboração de uma library para ser usada por outros programadores, não se esqueça de usá-las no arquivo header:

```
#ifndef PONTO_H
#define PONTO_H
...
#endif
```

Essas diretivas servem para evitar que um header file seja incluído mais de uma vez no mesmo projeto.

21

Tipo Abstrato de Dados (TAD)

- O TAD encapsula tipos de dados. A definição do tipo e todas as operações ficam localizadas numa seção do programa.
- **TAD FRAÇÃO**
- Tipo abstrato de dados fração. Baseado no conceito de número racional do campo da matemática. Resumo das operações matemáticas envolvidas:
- Soma de fração: (a/b)+(c/d)=((a.d+c.b) / b.d) simplificada.
- Multiplicação de fração: (a/b) * (c/d)= ((a*c) / (b*d)) simplificada.
- Igualdade: (a/b)== (c/d) se a*d == b*c.

22

TAD Fração

1

```
class fracao {
 private:
          long num;
          long den;
 public:
          fracao(long t,long m);
          fracao soma (fracao j);
          int igual(fracao t);
};
```

2

```
fracao::fracao(long t,long m)
{
 num=t;
 den=m;
}
fracao fracao::soma(fracao j){
fracao g((num*j.den)+(j.num*den),den*j.den);
return g;
}
int fracao::igual(fracao t){
 return ((num*t.den)==(den*t.num));
}
```

3

```
#include <stdio.h>
int main(){
fracao a(0,1),b(0,1);
a.soma(b);
if(a.igual(b))
   printf("\nIguais");
}
```

23

- Referência: Mizrahi, V. V., Treinamento em Linguagem C++, Módulo 2

24