Atribuição padrão de membro a membro

O operador de atribuição (=) pode ser utilizado para atribuir o estado de um objeto a outro objeto da mesma classe. Por padrão, tal atribuição é realizada pela **atribuição de membro a membro**.

Exemplo:

```
#include “Ponto2D”
int main()
{
   Ponto2D p1(3, 5), p2(0, 0);
   p2 = p1;
   ...
   return 0;
}
```


UNIVASF

Atribuição padrão de membro a membro

Observação:

A atribuição de membro a membro pode causar problemas sérios quando utilizada em uma classe cujos membros de dados contêm ponteiros para a memória alocada dinamicamente.

Exercício:

Preencha as lacunas em cada uma das seguintes sentenças:

a) Os membros de classe são acessados via operador _ponto (.)_ em conjunto com o nome de um objeto (ou referência a um objeto) da classe ou via operador _seta (->)_ em conjunto com um ponteiro para um objeto da classe.

b) Os membros de classe especificados como _*private*_ são acessíveis apenas às funções membros da classe.

c) Os membros de classe especificados como _*public*_ são acessíveis em qualquer lugar em que um objeto da classe esteja no escopo.

d) A _atribuição-padrão de membro a membro_ pode ser utilizada para atribuir o estado de um objeto de uma classe a outro objeto da mesma classe.

Exercício:

Localize o(s) erro(s) em cada uma das seguintes seqüências e explique como corrigi-lo(s):

a) Suponha que o seguinte protótipo é declarado na classe Ponto2D:

```
void ~Ponto2D(int);
```

b) A seguinte definição é uma definição parcial da classe Ponto2D:

```
class Ponto2D
{
   private:
      int x=0;
      int y=0;
   ...
};
```

Exercício:

Localize o(s) erro(s) em cada uma das seguintes seqüências e explique como corrigi-lo(s):

c) Suponha que o seguinte protótipo é declarado na classe Ponto2D:

```
int Ponto2D(int, int);
```

a) Erro: Os destrutores não têm permissão de retornar valores (nem mesmo de especificar um tipo de retorno) nem de aceitar argumentos.
Correção: Remova o tipo de retorno void e o parâmetro int da declaração.

b) Erro: Os membros de dados não podem ser explicitamente inicializados na definição da classe.
Correção: Remova a inicialização explícita da definição da classe e inicialize os membros de dados em um construtor.

c) Erro: Os construtores não têm permissão de retornar nada.
Correção: Remova o tipo de retorno int da declaração.

UNIVASF

Agregação

Uma classe pode ter objetos de outra classe como membros. Essa capacidade é chamada agregação e às vezes é referida como um relacionamento tem um.

Vamos ver agora a definição das classes Horario e Data.

```
//conteúdo do arquivo horario.h
#ifndef HORARIO_H
#define HORARIO_H
  class Horario
  {//Note uma inversão na ordem dos membros de classe públicos
    public: //e privados, apresentando assim a interface primeiro
      Horario(int=0, int=0, int=0);
      void setHorario(int, int, int);
      void setHora (int);
      void setMinuto (int);
      void setSegundo (int);
      int getHora();
      int getMinuto();
      int getSegundo();
      void imprima();
```

```
 private:
        int hora;
        int minuto;
        int segundo;
   };
#endif

//conteúdo do arquivo horario.cpp
#include <iostream>
using std::cout;
#include <iomanip>
using std::setfill;
using std::setw;
#include "horario.h"
Horario::Horario(int h, int m, int s)
{
   setHorario(h, m, s);
}
void Horario::setHorario(int h, int m, int s)
{
   setHora(h);
   setMinuto(m);
   setSegundo(s);
}
```

```
void Horario::setHora (int h)
{
  hora = (h>=0 && h<24)? h: 0;
}
void Horario::setMinuto (int m)
{
  minuto = (m>=0 && m<60)? m: 0;
}
void Horario::setSegundo (int s)
{
  segundo = (s>=0 && s<60)? s: 0;
}
int Horario::getHora()
{
  return hora;
}
int Horario::getMinuto()
{
  return minuto;
}
int Horario::getSegundo()
{
  return segundo;
}
```

```
void Horario::imprima() {
   cout << setfill('0') << setw(2) << getHora() << ":" << setw(2) <<
getMinuto() << ":" << setw(2) << getSegundo();
}

//conteúdo do arquivo data.h
#ifndef DATA_H
#define DATA_H
   class Data {
      public:
         Data(int=1, int=1, int=1900);
         void apresenta();
         void setDia(int);
         void setMes(int);
         void setAno(int);
         int getDia();
         int getMes();
         int getAno();
      private:
         int dia;
         int mes;
         int ano;
         int verificaDia(int);
   };
#endif
```

```
//conteúdo do arquivo data.cpp
#include <iostream>
using std::cout;
using std::endl;
#include "data.h"
Data::Data(int d, int m, int a)
{
  setMes(m);
  setAno(a);
  setDia(d);
}
void Data::setDia(int d)
{
  dia=verificaDia(d);
}
void Data::setMes(int m)
{
  if (m>0 && m<=12)
    mes=m;
  else
  {
    mes=1;
    cout << endl << "Mes invalido (" << m << ") setado para 1." << endl;
  }
}
```

```
void Data::setAno(int a)
{
   if (a>=1900 && a<2015)
      ano=a;
   else
   {
      ano=1900;
      cout << endl << "Ano invalido (" << a << ") setado para 1900." <<
endl;
   }
}
int Data::getDia()
{
   return dia;
}
int Data::getMes()
{
   return mes;
}
int Data::getAno()
{
   return ano;
}
```

```
void Data::apresenta()
{
   cout << getDia() << '/' << getMes() << '/' << getAno();
}
int Data::verificaDia(int diaTeste)
{
   static const int diasPorMes[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
   if (diaTeste>0 && diaTeste<=diasPorMes[getMes()])
      return diaTeste;
   if (getMes()==2 && diaTeste==29 && (getAno()%400==0 ||
(getAno()%4==0 && getAno()%100!=0)))
      return diaTeste;
   cout << endl << "Dia invalido (" << diaTeste << ")setado para 1." <<
endl;
   return 1;
}
```

Agregação

Vamos analisar agora a classe Compromisso, que tem como membros de dados um objeto da classe Data, outro da classe Horario e um vetor de caracteres representando a descrição do compromisso.

```
//conteúdo do arquivo compromisso.h
#ifndef COMPROMISSO_H
#define COMPROMISSO_H
  #include "data.h"
  #include "horario.h"
  class Compromisso
  {
    public:
      Compromisso(Data, Horario, char [50]);
      void escreva();
    private:
      Data data;
      Horario horario;
      char descricao[50];
  };
#endif
```

```
//conteúdo do arquivo compromisso.cpp
#include <cstring>
#include "compromisso.h"
#include <iostream>
using std::out;
using std::endl;
Compromisso::Compromisso(Data data, Horario horario, char
descricao[50])
{
   Compromisso::data=data; // atribuição-padrão membro a membro
   Compromisso::horario=horario; // atribuição-padrão membro a membro
   strcpy(Compromisso::descricao,descricao);
}
void Compromisso::escreva()
{
   cout << endl << "Informacoes sobre o compromisso:" << descricao <<
endl << "Horario: ";
   horario.imprima();
   cout << endl << "Data: ";
   data.apresenta();
   cout << endl;
}
```

Agregação

A seguir um programa *driver* que se utiliza das classes anteriormente definidas.

```
//conteúdo do arquivo principalCompromisso.cpp
#include "horario.h"
#include "data.h"
#include "compromisso.h"
int main()
{
  Data d(22,3,2010);
  Horario h(12);
  Compromisso c(d, h, "teste");
  c.escreva();
  return 0;
}
```

Inicializador de membro

Existe outra forma de se inicializar os membros de dados, utilizando-se os inicializadores de membros. Sua sintaxe é a seguinte:

```
//conteúdo do arquivo compromisso.cpp
#include <cstring>
#include "compromisso.h"
#include <iostream>
using std::out;
using std::endl;
Compromisso::Compromisso(Data data, Horario horario, char
descricao[50]):data(data), horario(horario)
{
  strcpy(Compromisso::descricao,descricao);
}
...
```

Inicializador de membro

Todos os membros de dados podem ser inicializados utilizando a sintaxe de inicializador de membro.

Sendo assim o membro *descricao* pode ser inicializado com a sintaxe de inicializador de membro?

Não.

Pois, o inicalizador de membro se utiliza implicitamente do operador de atribuição o qual não pode ser utilizado para atribuirmos um vetor de caracteres a outro.

Sendo assim, a afirmativa correta é: Todos os membros de dados que possibilitam a aplicação do operador de atribuição podem ser inicializados utilizando a sintaxe de inicializador de membro.

Inicializador de membro

É interessante ressaltar que um objeto membro não precisa ser explicitamente inicializado por um inicializador de membro. Se um inicializador de membro não for utilizado, o construtor padrão do objeto membro será chamado implicitamente.

Os valores (se houverem) estabelecidos pelo construtor-padrão podem ser sobrescritos pelas funções set. Entretanto, para inicializações complexas, esta abordagem pode exigir trabalho e tempo adicional significativos.

Exercício:

Construir uma classe Agenda que será uma agregação de Compromisso.

Implemente um programa que se utilize da classe Agenda, disponibilizando um menu para o usuário com as seguintes funcionalidades:

- Inserir compromisso;
- Excluir compromisso (com base em seu horário e data de ocorrência);
- Consultar a disponibilidade de um determinado horário em um determinado dia;
- Consultar o horário e a data da ocorrência de um compromisso;
- Imprimir o conjunto de compromissos

Observação: Considerar que os compromissos agendados iniciam em horas exatas, por exemplo 07:00:00, e têm sempre duração de uma hora. Caso julgue necessário, efetue as devidas adaptações nas classes Horario, Data e Compromisso definidas anteriormente.

UNIVASF