

# Linguagem de Programação C++

## Formas de utilização de uma classe

Para exemplificar as formas de utilização de uma classe, nos utilizaremos da classe Ponto2D que definimos anteriormente.

Uma vez que a classe foi definida, ela pode ser utilizada como um tipo em declarações de objeto, array, ponteiro e referência, como mostrado a seguir:

```
Ponto2D ponto;
```

```
Ponto2D vetorDePontos[10];
```

```
Ponto2D *ponteiroParaPonto = &ponto;
```

```
Ponto2D &referenciaAPonto = ponto;
```

```
...
```

```
ponto.mostraCoordenadas();
```

```
vetorDePontos[0].mostraCoordenadas();
```

```
(*ponteiroParaPonto).mostraCoordenadas();
```

```
ponteiroParaPonto->mostraCoordenadas();
```

```
212 referenciaAPonto.mostraCoordenadas();
```

# Linguagem de Programação C++

## Escopo de variáveis

Variáveis declaradas em uma função-membro têm escopo de bloco e são conhecidas apenas por esta função.

Uma função membro pode definir uma variável com o mesmo nome de uma variável com escopo de classe, a variável de escopo de classe é ocultada pela variável de escopo de bloco no escopo do bloco. Esta variável oculta pode ser acessada colocando-se o nome da classe seguido pelo operador de resolução de escopo binário antes do nome da variável.

Um detalhe importante é que variáveis globais ocultas podem ser acessadas utilizando-se o operador de resolução de escopo unário.

Vamos analisar um exemplo.

# Linguagem de Programação C++

```
int var;  
class Exemplo  
{  
    private:  
        int var;  
    public:  
        void teste()  
        {  
            int var;  
            var = 3;  
            Exemplo::var = 2;  
            ::var = 1;  
        }  
};  
...
```

# Linguagem de Programação C++

## Argumentos-padrão

A linguagem C++ possibilita a atribuição de um valor padrão para parâmetros de funções membros.

Uma grande utilidade para este recurso é a possibilidade de definirmos um construtor padrão com parâmetros. Pois, não existirá mais a obrigatoriedade da definição de argumentos na instanciação de objetos, já que se nenhum valor for fornecido na chamada de construtor, o mesmo ainda inicializará os membros de dados mantendo o objeto instanciado em um estado consistente.

Exemplo:

```
...  
class Ponto2D  
{  
    ...  
    public:  
        Ponto2D (float = 0, float = 0);  
    ...  
};
```

## Linguagem de Programação C++

Podemos então instanciar objetos da classe Ponto2D da seguinte forma:

...

```
Ponto2D ponto1; //x e y receberão 0
```

```
Ponto2D ponto2(4); //x receberá 4 e y receberá 0
```

```
Ponto2D ponto3(7,3); //x receberá 7 e y receberá 3
```

...

## Linguagem de Programação C++

### Exercício:

Crie uma classe Jogo que permitirá escrever um programa completo para jogar o jogo-da-velha. A classe contém como membros de dados privados um vetor bidimensional 3 x 3 de inteiros. O construtor deve inicializar a grade vazia com todos os valores como zero. Deve permitir dois jogadores humanos. Para onde quer que o primeiro jogador se mova, coloque 1 no quadrado especificado. Coloque 2 para onde quer que o segundo jogador se mova. Todo movimento deve ocorrer em um quadrado vazio. Depois de cada movimento, determine se houve uma derrota ou um empate. Se você se sentir motivado, modifique seu programa de modo que o computador faça o movimento para um dos jogadores. Além disso, permita que o jogador humano especifique se quer ser o primeiro ou o segundo a jogar.

# Linguagem de Programação C++

## Retorno de referência a um membro de dados

Uma referência a um objeto é um alias para o nome de um objeto e, portanto, pode ser utilizado a esquerda de uma instrução de atribuição.

Neste contexto, a referência constitui um *lvalue* válido. Uma maneira, **indesejável**, de utilizar esta capacidade é fazer uma função membro pública de uma classe retornar uma referência a um membro de dados privado desta classe.

Por exemplo:

```
#include <iostream>
using std::cout;
using std::endl;
//continua
```

```
class Exemplo2
{
    private:
        int valor;
    public:
        Exemplo2()
        {
            valor=1;
        }
        int &funcao()
        {
            return valor;
        }
        int getValor()
        {
            return valor;
        }
};
```

## Linguagem de Programação C++

```
int main()
{
    Exemplo2 objeto;
    int &ref=objeto.funcao();
    cout << "Valor inicial: " << objeto.getValor() << endl;
    ref = 7;
    cout << "Valor atualizado atraves da referencia: " <<
objeto.getValor() << endl;
    objeto.funcao() = 3;
    cout << "Valor atualizado atraves da referencia " <<
"como lvalue: " << objeto.getValor() << endl;
    return 0;
}
```

# Linguagem de Programação C++

## 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;
}
```

# Linguagem de Programação C++

## 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.

# Linguagem de Programação C++

## 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.

# Linguagem de Programação C++

## Exercício:

Localize o(s) erro(s) em cada uma das seguintes sequências 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;
        ...
};
```

# Linguagem de Programação C++

## Exercício:

Localize o(s) erro(s) em cada uma das seguintes sequências 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.**

# Linguagem de Programação C++

## 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 Horário e Data.

```
//conteúdo do arquivo horario.h
```

```
#ifndef HORARIO_H
```

```
#define HORARIO_H
```

```
class Horário
```

```
{//Note uma inversão na ordem dos membros de classe públicos
```

```
public: //e privados, apresentando assim a interface primeiro
```

```
    Horário(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<2020)
        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;
}

```

# Linguagem de Programação C++

## 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;
}

```

# Linguagem de Programação C++

## 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;
}
```