```
//conteúdo do arquivo Matriz.h
class Matriz
{
  private:
    int *elementos;
    int numeroDeLinhas;
    int numeroDeColunas;
    void setNumeroDeLinhas(int);
    void setNumeroDeColunas(int);
    int getNumeroDeLinhas();
    int getNumeroDeColunas();
  public:
    Matriz(int, int);
    ~ Matriz();
    void setElemento(int, int, int);
    int getElemento(int, int);
    void inicialize();
    void imprima();
};
```

```
//conteúdo do arquivo Matriz.cpp
#include "Matriz.h"
#include <iomanip>
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
Matriz::Matriz(int l, int c)
{
   ...
}
Matriz::~Matriz()
{
   delete []elementos;
}
...
```

Linguagem de Programação C++

Exercício:

Com base no que vimos a respeito da representação do construtor de uma classe em um diagrama de classes em UML siga sua lógica e construa um diagrama de classe em UML que represente a classe Matriz.

Matriz
-elementos: int * -numeroDeLinhas: int -numeroDeColunas: int
+<<construtor>> Matriz(in int,in int) +<<destrutor>> ~Matriz() +setElemento(in int,in int,in int): void +getElemento(in int,in int): int +inicialize(): void +imprima(): void -setNumeroDeLinhas(in int): void -setNumeroDeColunas(in int): void -getNumeroDeLinhas(): int -getNumeroDeColunas(): int

Lembre-se que neste caso o diagrama de classes apresentado está direcionado para a linguagem C o que foi mencionado não ser aconselhável.

Empacotador de pré-processador

Creio que alguns de vocês já devem ter se perguntado sobre a possibilidade de inserir, por acidente, a definição de uma classe mais de uma vez em um programa, pois em um programa grande ocorrem muitas inclusões de arquivos cabeçalhos que por sua vez podem incluir outros arquivos cabeçalhos.

Para evitar a ocorrência deste erro devemos utilizar o empacotador de pré-processador **#ifndef**. O qual representa 'se não definido'.

Sua sintaxe é:

```
#ifndef <ROTULO>
#define <ROTULO>
   ...
#endif
```

Para uma compreensão adequada analisaremos sua utilização na classe Matriz.

```
//conteúdo do arquivo Matriz.h
#ifndef MATRIZ_H
#define MATRIZ_H
  class Matriz
  {
    private:
      int *elementos;
      int numeroDeLinhas;
      int numeroDeColunas;
      void setNumeroDeLinhas(int);
      void setNumeroDeColunas(int);
      int getNumeroDeLinhas();
      int getNumeroDeColunas();
    public:
      Matriz(int, int);
      ~ Matriz();
      void setElemento(int, int, int);
      int getElemento(int, int);
      void inicialize();
      void imprima();
  };
#endif
```

Exercício:

Visando melhor fixar a utilização da alocação dinâmica de memória, explorando as funções-membros construtor e destrutor, explorar o princípio do ocultamento de implementação e demonstrar as vantagens na manutenibilidade de sistemas gerados com a utilização da OO, adapte a solução do exercício do slide 191 sobre a classe Matriz. Faça com que este armazene os elementos da matriz através de um membro de dados int ** ao invés de int * ou vise e versa para quem gerou inicialmente sua solução com int **. Utilize o empacotador de pré-processador.

```
//conteúdo do arquivo Matriz.h
#ifndef MATRIZ_H
    #define MATRIZ_H
    class Matriz
    {
      private:
        int **elementos;
        int numeroDeLinhas;
        int numeroDeColunas;
        void setNumeroDeLinhas(int);
        void setNumeroDeColunas(int);
        int getNumeroDeLinhas();
        int getNumeroDeColunas();
      public:
        Matriz(int, int);
        ~ Matriz();
        void setElemento(int, int, int);
        int getElemento(int, int);
        void inicialize();
        void imprima();
    };
#endif
```

```
//conteúdo do arquivo Matriz.cpp anterior
Matriz::Matriz(int l, int c) {
   if (l>0 && c>0) {
      setNumeroDeLinhas(l);
      setNumeroDeColunas(c);
      elementos = new int [getNumeroDeLinhas()*getNumeroDeColunas()];
      for (int i=0; i<getNumeroDeLinhas()*getNumeroDeColunas(); i++)
         setElemento(i/getNumeroDeColunas(), i%getNumeroDeColunas(), 0);
   } else  {
      setNumeroDeLinhas(1);
      setNumeroDeColunas(1);
      elementos = new int;
      setElemento(0,0,0); }
}
//conteúdo do arquivo Matriz.cpp atual
Matriz::Matriz(int l, int c) {
   if (l>0 && c>0) {
      setNumeroDeLinhas(l);
      setNumeroDeColunas(c);
   } else {
      setNumeroDeLinhas(1);
      setNumeroDeColunas(1); }
   elementos = new int *[getNumeroDeLinhas()];
   for (int i=0; i<getNumeroDeLinhas(); i++)
      elementos[i] = new int [getNumeroDeColunas()];
   for (int i=0; i<getNumeroDeLinhas(); i++)
      for (int j=0; j<getNumeroDeColunas(); j++)
         setElemento(i, j, 0);
}
```

```
//conteúdo do arquivo Matriz.cpp anterior
Matriz::~Matriz()
{
  delete []elementos;
}

//conteúdo do arquivo Matriz.cpp atual
Matriz::~Matriz()
{
  for (int i=0; i<getNumeroDeLinhas(); i++)
    delete [](elementos[i]);
  delete []elementos;
}
```

```
//conteúdo do arquivo Matriz.cpp anterior
void Matriz::setElemento(int l, int c, int v)
{
  if (v>=0 && (0<=l && l<getNumeroDeLinhas()) && (0<=c &&
c<getNumeroDeColunas()))
    *(elementos+l*getNumeroDeColunas()+c)=v;
}

//conteúdo do arquivo Matriz.cpp atual
void Matriz::setElemento(int l, int c, int v)
{
  if (v>=0 && (0<=l && l<getNumeroDeLinhas()) && (0<=c &&
c<getNumeroDeColunas()))
    elementos[l][c]=v;
}
```

```
//conteúdo do arquivo Matriz.cpp anterior
int Matriz::getElemento(int l, int c)
{
  if ((0<=l && l<getNumeroDeLinhas()) && (0<=c &&
c<getNumeroDeColunas()))
    return (*(elementos+l*getNumeroDeColunas()+c));
  else
    return (-1);
}

//conteúdo do arquivo Matriz.cpp atual
int Matriz::getElemento(int l, int c)
{
  if ((0<=l && l<getNumeroDeLinhas()) && (0<=c &&
c<getNumeroDeColunas()))
    return (elementos[l][c]);
  else
    return (-1);
}
```

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

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;
      }
};
...
```

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. Permita 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.

```
//conteúdo do arquivo jogo.h
#ifndef JOGO_H
#define JOGO_H
  class Jogo
  {
    private:
      int grade[3][3];
    public:
      Jogo();
      int setElementoDaGrade(int, int, int);
      int getElementoDaGrade(int, int);
      void mova(int, int, int);
      void solicitaJogada(int &, int &, int &);
      int verificaSituacaoDoJogo();
      void exibaJogo();
      void jogar();
  };
#endif
```

```
//conteúdo do arquivo jogo.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include "jogo.h"
Jogo::Jogo()
{
  for (int i=0; i<3; i++)
    for (int j=0; j<3; j++)
      grade[i][j]=0;
}
int Jogo::setElementoDaGrade(int jogador, int linha, int coluna)
{
  if (jogador==1 || jogador==2)
    if (1<=linha && linha<=3)
      if (1<=coluna && coluna<=3)
        if (grade[linha-1][coluna-1] == 0)
        {
          grade[linha-1][coluna-1] = jogador;
          return 0;
        }
        else
          return 4;
      else
```

```
            return 3;
        else
            return 2;
    else
        return 1;
}
int Jogo::getElementoDaGrade(int linha, int coluna)
{
    return grade[linha][coluna];
}
void Jogo::mova(int jogador, int linha, int coluna)
{
    int retorno;
    while (retorno=setElementoDaGrade(jogador, linha, coluna))
    {
        switch (retorno)
        {
            case 1:
                cout << endl << "ERRO!" << endl <<"Numero do jogador
invalido." << endl;
                cout << "Forneca um numero de jogador valido: ";
                cin >> jogador;
                cout << endl;
                break;
```

```
        case 2:
          cout << endl << "ERRO!" << endl <<"Numero da linha
invalido." << endl;
          cout << "Forneca um numero de linha valido: ";
          cin >> linha;
          cout << endl;
          break;
        case 3:
          cout << endl << "ERRO!" << endl <<"Numero da coluna
invalido." << endl;
          cout << "Forneca um numero de coluna valido: ";
          cin >> coluna;
          cout << endl;
          break;
        case 4:
          cout << endl << "Posicao ja ocupada." << endl;
          cout << "Forneca uma nova posicao." << endl;
          solicitaJogada(jogador, linha, coluna);
          break;
    }
  }
}
void Jogo::solicitaJogada(int &jogador, int &linha, int &coluna)
{
  cout << endl << "JOGADOR " << jogador;
```

```
  cout << endl << "Entre com a linha para a qual deseja se mover: ";
  cin >> linha;
  cout << endl << "Entre com a coluna para a qual deseja se mover: ";
  cin >> coluna;
}
int Jogo::verificaSituacaoDoJogo()
{
  bool jogadorUmGanhouComLinha, jogadorDoisGanhouComLinha,
  jogadorUmGanhouComColuna,  jogadorDoisGanhouComColuna,
  empate = true;
  for (int i=0; i<3; i++)
  {
    jogadorUmGanhouComLinha = jogadorDoisGanhouComLinha = true;
    for (int j=0; j<3; j++)
    {
      if (!getElementoDaGrade(i, j))
      {
        jogadorUmGanhouComLinha = jogadorDoisGanhouComLinha =
empate = false;
        break;
      }
      else
        if (getElementoDaGrade(i, j)==1)
          jogadorDoisGanhouComLinha = false;
        else
```

```
            jogadorUmGanhouComLinha = false;
      }
      if (jogadorUmGanhouComLinha)
         return 1;
      if (jogadorDoisGanhouComLinha)
         return 2;
   }
   for (int i=0; i<3; i++)
   {
      jogadorUmGanhouComColuna = jogadorDoisGanhouComColuna =
true;
      for (int j=0; j<3; j++)
      {
         if (!getElementoDaGrade(j, i))
         {
            jogadorUmGanhouComColuna =
jogadorDoisGanhouComColuna = empate = false;
            break;
         }
         else
            if (getElementoDaGrade(j, i)==1)
               jogadorDoisGanhouComColuna = false;
            else
               jogadorUmGanhouComColuna = false;
   }
```

```
      if (jogadoUmGanhouComColuna)
         return 1;
      if (jogadoDoisGanhouComColuna)
         return 2;
   }
   if ( getElementoDaGrade(1,1)==1 &&
((getElementoDaGrade(0,0)==getElementoDaGrade(1,1) &&
      getElementoDaGrade(0,0)==getElementoDaGrade(2,2)) ||
      (getElementoDaGrade(2,0)==getElementoDaGrade(1,1) &&
getElementoDaGrade(2,0)==getElementoDaGrade(0,2))))
      return 1;
   if ( getElementoDaGrade(1,1)==2 &&
((getElementoDaGrade(0,0)==getElementoDaGrade(1,1) &&
getElementoDaGrade(0,0)==getElementoDaGrade(2,2)) ||
      (getElementoDaGrade(2,0)==getElementoDaGrade(1,1) &&
getElementoDaGrade(2,0)==getElementoDaGrade(0,2))))
      return 2;
   if (empate)
      return 3;
   return 0;
}
void Jogo::exibaJogo()
{
   cout << " **************" << endl;
   for (int i=0; i<3; i++)
   {
```

```
      for (int j=0; j<3; j++)
         cout << " * " << getElementoDaGrade(i,j);
      cout << " *" << endl;
      cout << " **********" << endl;
   }
}
void Jogo::jogar()
{
   int jogador=1, linha, coluna;
   cout << endl << "Bem vindo ao jogo da velha!" << endl;
   exibaJogo();
   do
   {
      cout << "Entre com as coordenadas da posicao para a qual deseja
se mover.";
      solicitaJogada(jogador, linha, coluna);
      mova(jogador, linha, coluna);
      exibaJogo();
      if (verificaSituacaoDoJogo()==jogador)
      {
         cout << endl << "Parabens jogador " << jogador << " voce
ganhou!" << endl;
         break;
      }
```

```
      if (jogador==1)
         jogador = 2;
      else
         jogador = 1;
   }while(verificaSituacaoDoJogo()!=3);
   if (verificaSituacaoDoJogo()==3)
      cout << endl << "Ocorreu empate!" << endl;
}
```

```
//conteúdo do arquivo main.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include "jogo.h"
int main()
{
   Jogo novoJogo;
   novoJogo.jogar();
   return 0;
}
```

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;
      }
};
```

```
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;
}
```