Universidade Federal de Viçosa
Campus Rio Paranaíba
Sistemas de Informação
Prof. Guilherme Pena

SIN141
Computação Orientada a Objetos
Lista de Exercícios 1
06/06/2022

Classes

ATENÇÃO: Todas as respostas e programas serão enviados via **PVANet**.

Exercícios (Entrega **individual** um dia antes da prova: **20/06/2022)**

Descritivos (para Revisão apenas)

1) Preencha as seguintes lacunas:

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

b) Os membros de classe especificados como _____ são acessíveis às funções-membro da classe e friend da classe.

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

d) A _____ pode ser utilizada para atribuir um objeto de uma classe a outro objeto da mesma classe.

e) Ao seperar a interface de uma classe da sua implementação, o *link* entre os métodos é feito através do nome da classe seguido do operador _____ .

f) Um função não-membro deve ser declarada como _____ de uma classe para ter acesso aos membros private dessa classe.

g) Toda sobrecarga de operador merece uma atenção especial quanto ao seu retorno e a sobrecarga é feita através da palavra chave _____ .

h) A toda classe deve-se definir explicitamente os 3 construtores, o destrutor e a sobrecarga do operador de _____.

i) A _____ é uma forma de reutilização de software em que novas classes absorvem os dados e comportamentos de classes existentes aprimorando-as com novas capacidades.

j) Ao derivar de uma classe básica pela herança public, membros public se tornam _____ na classe derivada, membros protected se tornam _____ na classe derivada e membros _____ não são herdados diretamente.

2) Descreva de forma breve cada um dos construtores e o destrutor, dizendo os detalhes de sua implementação e em qual momento eles são acionados.

3) Descreva de forma breve quais modificações são feitas quando construtores e destrutores são implementados na hierarquia de herança, e como é a sua forma de acionamento neste caso.

Implementação (Divida cada projeto em **.h**, **.cpp**, **main** e **makefile**)

1) (Classe Fracao) Crie uma classe chamada **Fracao** para realizar aritmética com frações. Utilize váriáveis int para representar os dados private da classe numerador e denominador. **Lembre-se de que o denominador nunca pode ser nulo!**

Implemente os seguintes itens na sua classe:

- Construtores (padrão, normal e de cópia)
- Destrutor
- Get/Set
- Imprimir a fracao no formato *num/den*
- Imprimir a fracao no formato de ponto flutuante

- Faça um método ***simplifica*** para a classe Fracao. Para simplificar, você pode dividir o numerador e o denominador pelo seu Máximo Divisor Comum (MDC).

Exemplo: A fracao (2/4) possui MDC = 2, simplificando-a ela se tornará (1/2)

O valor do MDC pode ser calculado recursivamente por:

$$MDC(a, b) = \begin{cases} b, & \text{se } a \% b = 0 \\ MDC(b, a), & \text{se } a < b \\ MDC(b, a\%b), & \text{caso contrário} \end{cases}$$

Obs: *Embora possa ser uma função-membro da classe, a função MDC não deveria ser parte de classe, mas ser uma função separada e independente, pois não é uma função de número racional, mas uma função de dois inteiros*.

- Sobrecarga da atribuição (=)

- Sobrecarga de entrada e saída de fluxo (<< e >>)

- Sobrecarga das operações (+, -, *, /, +=, -=, *=, /=, <, <=, >, >=, == e !=)

Inclua a chamada do método simplifica em todas as operações realizadas pela classe e nas funções set e <<.

Por fim, escreva um programa main para testar a sua classe.

2) (Classe Conjunto) Crie uma classe **Conjunto** para que objetos dessa classe possam armazenar conjuntos de inteiros no intervalo 0 a 100. Represente o conjunto internamente por um array de booleanos, onde o elemento [i] desse array é **true** se o valor i faz parte do conjunto ou **false** senão. O construtor-padrão deve inicializar o conjunto como "conjunto vazio", isto é, sem elemento algum (todos false).

Como exemplo, segue a representação do conjunto {1, 2, 5, 100}

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	…	*98*	*99*	*100*
false	**true**	**true**	false	false	**true**	false	false	false	false	false		false	false	**true**

Implemente os operadores clássicos de conjunto como funções-membro desta classe:

- **união** (dados dois conjuntos retorna um terceiro que contém todos os elementos desses dois)
- **interseção** (dados dois conjuntos retorna um terceiro que contém apenas os elementos que são comuns aos dois)
- **pertence** (dados um conjunto e um inteiro retorna true se o inteiro faz parte do conjunto, e false senão)
- **insere** (dados um conjunto e um inteiro insere o inteiro no conjunto nada é alterado se ele já fazia parte)
- **retira** (dados um conjunto e um inteiro exclui o inteiro no conjunto nada é alterado se ele nem fazia parte)
- **vazio** (retorna true se o conjunto é vazio, e false senão)
- **contém** (dados dois conjuntos retorna true se todos os elementos do segundo estão no primeiro, false senão)
- **estácontido** (dados dois conjuntos retorna true se todos elementos do primeiro estão no segundo, false senão)
- **imprime** (dado um conjunto imprime seus elementos)
- **igual** (dados dois conjuntos retorna true se os dois contêm exatamente os mesmos elementos)

Implemente um construtor adicional que recebe um array de inteiros e o tamanho desse array, e inicializa o conjunto com os valores desse array. Veja um exemplo da utilização desse construtor:

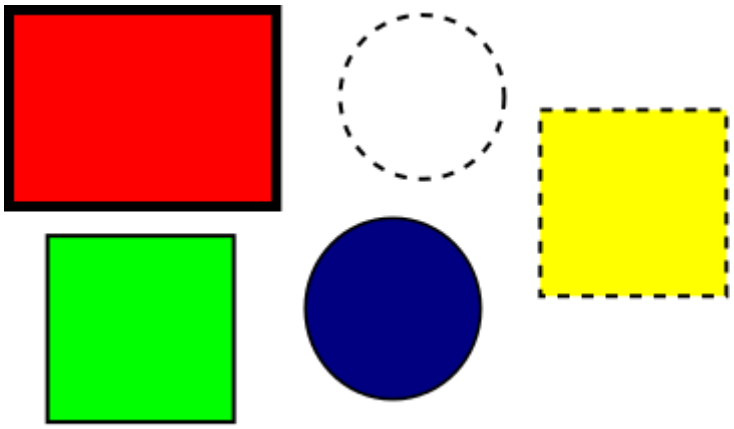
```
Conjunto a, b;
int v[3] = {1, 2, 5};

b.imprime(); //imprime o conjunto vazio b
a.insere(4); //insere o elemento 4 no conjunto a
a.imprime(); //imprime o conjunto unitário a

//cria conjunto c contendo os 3 valores de v, ou seja, {1, 2, 5}
Conjunto c(v, 3);

//atribui a b o conjunto a união c, que resulta em {1, 2, 4, 5}
b = a.uniao(c);
```

Faça sobrecarga dos operadores **==**, **<**, **>**, **+** e ***** para funcionarem respectivamente como as funções membro **igual**, **estácontido**, **contém**, **união** e **interseção** definidas anteriormente.

Por fim, escreva um programa main para testar a sua classe.

3) Em programas de manipulação de imagens, figuras geométricas, que são desenhadas numa tela, podem ter suas características armazenadas por diferentes tipos de objetos.

Por exemplo:



Tais figuras geométricas podem manter características como **posição na tela (x, y)**, **cor de fundo**, **tipo de contorno**, **espessura do contorno** além de outras características relacionadas com o tipo da figura (círculo ou retângulo).

Vamos supor uma aplicação (Main) que irá trabalhar com dois tipos de figuras, que deverão ser modeladas por classes de acordo com os diagramas UML abaixo que você deverá implementar:

Circulo
- x : double; - y : double; - cor : int; - espessuraContorno : int; - tipoContorno : int; - raio : double;
+ Circulo( ); + Circulo(const double, const double, const int, const int, const int, const double); + Circulo(const Circulo&); + ~Circulo( ); + setX(double); + getX( ) : double; + setY(double); + getY( ) : double; + setCor(int); + getCor( ) : int; + setEspessura(int); + getEspessura( ) : int; + setContorno(int); + getContorno( ) : int; + setRaio(double); + getRaio( ) : double; + operator=(const Circulo&) : Circulo&; + imprime( ); + area( ) : double; friend operator<<(...) : ostream&; friend operator>>(...) : istream&;

Retangulo
- x : double; - y : double; - cor : int; - espessuraContorno : int; - tipoContorno : int; - largura : double; - altura : double;
+ Retangulo( ); + Retangulo(const double, const double, const int, const int, const int, const double, const double); + Retangulo(const Circulo&); + ~Retangulo( ); + setX(double); + getX( ) : double; + setY(double); + getY( ) : double; + setCor(int); + getCor( ) : int; + setEspessura(int); + getEspessura( ) : int; + setContorno(int); + getContorno( ) : int; + setLargura(double); + getLargura( ) : double; + setAltura(double); + getAltura( ) : double; + operator=(const Retangulo&) : Retangulo&; + imprime( ); + area( ) : double; friend operator<<(...) : ostream&; friend operator>>(...) : istream&;

Os diagramas das classes Circulo e Retangulo possuem os **construtores**, **destrutores**, **get/set**, **operador de atribuição** e dois métodos adicionais "**imprime**" (imprime as informações do objeto) e "**area**" (calcula e retorna a area do objeto). Membros sem retorno são tratados ***void***. Além da sobrecarga dos operadores "<<" e ">>";

Obs: **Os atributos de cor e contorno da figura são representados da seguinte forma**
cor : inteiro entre 1 e 5, sendo que: 1=branca; 2=vermelha; 3=verde; 4=azul; 5=amarela;
espessuraControno : inteiro entre 1 e 5, sendo que 1 indica a linha mais fina e 5 a mais grossa
tipoControno : inteiro entre 1 e 2, sendo que 1=contínua; 2=tracejada;

Execute o seguinte programa Main para a sua implementação e verifique o resultado esperado:

Main.cpp

```
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include "Circulo.h"
#include "Retangulo.h"

using namespace std;

int main() {
   cout << fixed << setprecision(2);

   Circulo c1(100, 120, 3, 2, 1, 30);
   c1.imprime();
   cout << "Area = " << c1.area() << endl;

   Retangulo r1(40, 50, 1, 1, 2, 80, 30);
   r1.imprime();
   cout << "Area = " << r1.area() << endl;

   Circulo c2 = c1;
   c2.setX(25);
   c2.setY(17);
   c2.setRaio(25);
   c2.setEspessura(9); //um valor invalido
   c2.imprime();

   Retangulo r2;
   r2 = r1;
   r2.imprime();

   return 0;
}
```

Saída Esperada

```
--- [Circulo] ---
x = 100.00 y = 120.00
Cor da Figura = 3
EspessuraContorno = 2
TipoContorno = 1
Raio = 30.00
Area = 2827.43

--- [Retangulo] ---
x = 40.00 y = 50.00
Cor da Figura = 1
EspessuraContorno = 1
TipoContorno = 2
Largura = 80.00 Altura = 30.00
Area = 2400.00

--- [Circulo] ---
x = 25.00 y = 17.00
Cor da Figura = 3
EspessuraContorno = 1
TipoContorno = 1
Raio = 25.00

--- [Retangulo] ---
x = 40.00 y = 50.00
Cor da Figura = 1
EspessuraContorno = 1
TipoContorno = 2
Largura = 80.00 Altura = 30.00
```

Inclua neste main, as instruções para testar cada um dos métodos implementados e analise as saídas para ver se os objetos ficaram consistentes.

4) Propositalmente, pode-se notar que as classes **Circulo** e **Retangulo** mantém uma série de atributos e métodos em comum.

Sendo assim, defina uma *hierarquia de herança* contendo uma classe básica **FormaBasica** que deverá conter os **membros comuns** e que mantenha os atributos como "**private**" na sua implementação. Dessa forma, Circulo e Retangulo passarão a ser classes derivadas.

Use a mesma implementação de sua função main do exercício anterior, sem alterações, e verifique se os resultados se mantiveram iguais.