Instituto Superior de Engenharia de Coimbra
Departamento de Engenharia Informática e de Sistemas
Engenharia Informática

Programação Orientada a Objetos 2023/2024

Exercícios

Ficha Nº 5 | Redefinição de operadores

1. **Frações** – Exemplo feito na teórica – **refazer** como **TPC/Estudo** e tirar dúvidas no gabinete

Os números racionais podem ser representados de forma exata (sem perda de precisão) através do quociente de dois valores inteiros, ou seja, uma fração. Pretende-se uma classe, *Fracao*, que permita representar desta forma os números racionais. Pretende-se também que seja possível usar os operadores aritméticos e de comparação habituais e que estes executem as operações aritméticas normais, adaptadas à natureza das frações. A classe tem as seguintes características:

- Tanto o numerador como o denominador serão números inteiros. O denominador será representado por um valor positivo e não nulo. Assim, o sinal da fração será implicitamente o sinal do numerador.
- Deve ser possível construir objetos da classe *Fracao* apenas das seguintes maneiras:
  - Sem especificar nenhum inicializador; neste caso o valor inicial representará a fração 0/1.
  - Especificando um valor inteiro que será o numerador, considerando-se o denominador com o valor 1.
  - Especificando dois valores inteiros: o numerador e o denominador.
- Devem existir funções para obter e para modificar o numerador e o denominador. As funções para obter o numerador e o denominador devem poder ser chamadas mesmo sobre objetos que são constantes.

a) Construa a classe com as características pretendidas. Defina uma função *main* para testar a classe. Declare as frações: *a* com o valor ½, *b* com o valor 3, e *c*, constante, com o valor ¾. Teste a classe obtendo, modificando e mostrando os valores. Verifique que não consegue chamar as funções para modificar o numerador e denominador sobre o objeto constante *c*, e que consegue chamar as funções para obter o numerador e o denominador.

b) Pretende-se que seja possível obter a multiplicação de duas frações, atribuindo o resultado a outra fração através da expressão: *a = b * c;*

- Existem duas formas de fazer com que esta expressão seja possível: um operador membro e um operador global. Analise as vantagens e desvantagens de cada.
- Faça de ambas as formas. Verifique que não consegue ter ambas em simultâneo. Explique porquê.
- Coloque cada uma das versões em comentários à vez e teste a funcionalidade da multiplicação com a expressão *a = b * c;*.
- Explique a razão do número de parâmetros ser diferente nas formas membro e não membro.

c) Teste agora a expressão *a * b * c*. Identifique e explique que alterações são necessárias para suportar esta nova versão. Se não for necessário nada, explique porquê. Peça ajuda ao professor se achar a solução inesperada. Nesta alínea considere ambas as versões do operador membro e não membro (sempre uma de cada vez mantendo a outra em comentários).

d) Experimente a operação *a = b * 4;* confirme que apesar de não ter nenhum operador * que receba um inteiro como segundo operando, a expressão é possível e o resultado é correto (obtenha e mostre os valores do resultado em *a* no ecrã). Confirme que é possível tanto na versão membro como não membro. Identifique e explique o que se passa. Se tiver dificuldade, peça ajuda ao professor do seu laboratório neste ponto pois é importante.

e) Acrescente a palavra chave *explicit* no início do protótipo do construtor da classe *Fração* que recebe um inteiro (ou que pode ser chamado apenas com um inteiro). Volte a tentar a expressão *a = b * 4* (ou apenas *b * 4*). Confirme já não é possível. Explique a situação. Depois de ter explicado e confirmado com o professor que a sua explicação é correta, remova a palavra *explicit*.

f) Experimente agora a expressão *a = 4 * b* (ou só *4 * b*). Confirme que é possível e correta quando usa a versão não membro do operador, mas com a versão membro não compila. Explique porquê à luz das conclusões das alíneas anteriores. Confirme com o professor e anote as conclusões das alíneas até agora no seu caderno.

g) Acrescente agora o código ao seu programa que suporte a seguinte expressão: *cout << a;*

- Se fizer este operador como membro, será membro de que classe?
- Tente fazer o operador como membro. Se não conseguir, explique porque é que não consegue (peça a ajuda ao professor).
- Se não fez o operador como membro de uma classe, faça-o como global. Teste a sua funcionalidade.

h) Faça agora com que seja possível fazer *cout << a << b;* Qual é a alteração necessária? Explique qual. Se não for preciso nada, explique porquê.

- Importante: teste também a expressão *cout << a << c;*

i) No decorrer das duas alíneas anteriores deve ter feito funções que recebem e passam objetos *ostream* por referência. Experimente passá-los por cópia e verifique que não é possível. Explique a forma que os programadores da classe *ostream* usaram para impedir a passagem e retorno de objetos *ostream* por cópia. Peça ajuda e exemplos ao professor se for necessário.

j) Acrescente um operador que suporte a expressão *a *= b*. Uma vez que já passou pelas alíneas anteriores, em princípio já entende a diferença entre operadores membro e não membro. Assim faça logo o operador desta classe como membro. Teste o seu operador imprimindo o seu conteúdo no ecrã (através da expressão *cout << a*) e confirmando que os valores são os esperados.

k) Qual o tipo de retorno que o seu operador *= tem? Verifique se consegue fazer *a *= b *= c*. Analise com cuidado a expressão tendo em atenção que a associatividade do operador *= é da direita para a esquerda, ou seja, é como se fosse *a *= (b *= c);*

l) Como resultado da alínea anterior, deverá ter um operador *= que retorna um objeto *Fraccao* por cópia. Experimente agora o seguinte código:

```
Fracao a(1,2),b(2,3),c(3,4);
(a *= b) *= c;
cout << a
// é suposto aparecer 6/24
```

Aparece o resultado esperado? Se aparecer 2/6 é porque o seu operador não está totalmente correto em relação à forma pretendida. Veja o protótipo da função correspondente ao operador e confirme com o professor se não conseguir obter o resultado correto. Este aspeto é importante.

m) Acrescente à sua classe o suporte para as expressões a++; e ++a; Depois de ter este aspeto a funcionar, experimente agora c++; e ++c; e confirme que o compilador não deixa compilar essas duas expressões.

n) Considere o código abaixo. Pretende-se que funcione. Deve modificar algo na sua classe de forma a que o código abaixo funcione. Não pode alterar nada no código apresentado.

```
void func(int n) {
   cout << n;  // aparece 2
}
int main() {
   const Fraccao f(7,3);
   func(f);  // é passado automaticamente o valor 7/3
             // arredondado para baixo
   return 0;
}
```

o) Pretende-se que a expressão *if (a == b)* seja aceite e funcione como esperado. Faça com que isso aconteça.

p) Antes que se esqueça, escreva no seu caderno todas as conclusões acerca de operadores obtidas ao longo deste exercício.

q) Para consolidar os seus conhecimentos acerca de operadores até agora obtidos e para concluir o exercício, faça com que o código apresentado abaixo compile e tenha o resultado intuitivamente esperado. Esta alínea pode ficar para trabalho de casa.

```
int main() {
  Fraccao x(2,1),y(1,3),z;
  cout << " z= " << z << endl;
  z=x*y;
  cout << x << " * " << y << " = " << z << endl;
  z = x/y;
  cout << x << " / " << y << " = " << z << endl;

  Fraccao a(2,-4),b(2);
  cout << " a= " << a << " b= " << b << endl;
  a *= b;
  cout << "a *= b " << endl;
  cout << "a= " << a << " b= " << b << endl;

  cin >> a;
}
```

No final deste exercício deverá ter

- Experimentado a redefinição de operadores nas suas variadas formas: binários e unários, membro e não-membro.
- Entendido as diferenças de aplicação entre operadores binários membro e operadores binários não membro, e as vantagens e desvantagens de cada uma das duas opções.
- Experimentado a redefinição de operadores no contexto do seu uso em expressões compostas.
- Experimentado a redefinição de operadores em situações em que são usados objetos de duas classes diferentes.
- Percebido a influência de passar e retornar objetos por cópia e por referência no contexto de redefinição de operadores.
- Entendido o conceito de construção implícita e a forma como pode poupar código na redefinição e operadores (e funções em geral).
- Experimentado operadores de conversão.

2. **Vetores** de geometria analítica

Considere os vetores da geometria analítica (não se trata dos vetores de C++ para guardar coisas, nem sequer matrizes de uma dimensão). Um vetor é um segmento de reta com uma orientação: tem sempre com origem nas coordenadas 0,0 (sendo *sempre*, não é preciso guardar essa informação) e termina num ponto nas coordenadas *x,y*. Assim, um vetor é definido simplesmente pelas coordenadas *x,y*.

a) Escreva uma classe em C++ chamada Vetor cujos objetos representem vetores descritos atrás. A classe deve cumprir o seguinte:

- Apenas deve ser possível construir objetos desta classe mediante:
  - A indicação de ambas as suas coordenadas de término (nota: "indicação" ≠ "perguntar ao utilizador"). Qualquer valor *double* é válido, tanto para *x* como para *y*.
  - A indicação de apenas um valor. Neste caso, ambas as coordenadas do ponto terminal ficam com esse valor.
- Deve ser possível obter e modificar cada uma das coordenadas, mas sem desrespeitar o conceito de encapsulamento. As funções que permitem obter os dados devem poder ser chamadas sobre objetos constantes da classe, e as que modificam as coordenadas não.
- Obter um objeto *string* com a descrição textual do seu conteúdo (formato: "(x,y)").

Não inclua o *header file* <vector> para evitar confusões entre o vector da STL e o vetor deste exercício.

Teste a classe através de uma função *main* que tenha dois vetores *a* e *b* com coordenadas (1,2) e (3,4). Confirme que não é possível ter vetores sem especificar as suas coordenadas.

b) Defina operadores (aritméticos, comparação para a igualdade/desigualdade, outros …) que permitam a utilização da classe *Vetor* expressa na seguinte função *main*:

```
int main() {
   Vetor v1(2.0, 1.0), v2(1.0, 3.0), v3(2.2), z;
   z = v1 + v2 + v3;
   cout << v1 << "+" << v2 << "+" << v3 << "=" << z << endl; // obs: "(x,y)"
   z = v1 + 10.0;
   cout << v1 << " + " << " 10 = " << z << endl;
   z = 20.0 + v1;
   cout << "20 + " << v1 << " = " << z << endl;
   z = v1 - v2;
   cout << v1 << " - " << v2 << " = " << z << endl;
   Vetor a(1.0, 1.0), b(2.0, 4.0);
   cout << " a= " << a << " b= " << b << endl;
   a += b += v1;
   a += b;
   a += 10.0;
   cout << " a= " << a << endl;
   cout << "(a == b)? " << (a == b) << endl;
   cout << "(a != b)? " << (a != b) << endl;
}
```

c) Indique **duas** maneiras que permitem tornar possível a instrução: ***z=v1+10.0;***

d) Defina operadores necessários para cumprir a função *main()* apresentada abaixo:

```
int main() {
   Vetor a(1,1);
   double modulo = double(a);  // significado = modulo do vetor
   double k = a;
   Vetor b = 2.5; // verifique primeiro se já está a ser possível isto
   b = a + 4.0;   // verifique se fizer um operador para este caso dará erro
   Vetor c(1.0, 1.0);
   cout << "\n Operadores unários \n";
   cout <<"\nc=" << c;
   cout << "\n++   -> c=" << ++c;
   cout <<"\nc="<< c;
   Vetor d(1.0, 1.0);
   cout <<"\nd="<< d ;
   cout << "\nd++:" << d++;
   cout << "\nd=" << d << endl;
   cin >> a >> b;
   cout << a << "\n" << b;
   if ((bool) a)
      cout << "o vetor a tem as coordenadas 0,0";
   return 0;
}
```

No final deste exercício deverá ter

- Os mesmos objetivos que no exercício anterior, mas deve fazer ambos, este aqui, o outro na aula teórica + em casa
- Experimentado um operador de conversão

3. **Automóvel** com dados que não se podem atribuir e um contador automático
Considere o conceito de *Automovel*. Um automóvel tem vários atributos. Defina alguns (2 ou 3) entre os quais se inclui a matrícula (*string*) (pode reaproveitar o código do exercício já feito na ficha 3).

a) Construa a classe especificada, incluindo um construtor que faça sentido em relação ao significado de "Automóvel" e que obrigue à especificação de uma matrícula.

b) Pretende-se que se traga para o programa a noção de "fazer um automóvel igual a outro". Isto significa que se pretende ter a cor, os extras, etc., iguais ao "outro", mas há uma coisa que nunca muda, que é a matrícula. Faça com que seja possível efetuar a atribuição entre dois automóveis com a expressão habitual a = b, mas em que a matrícula nunca é modificada no automóvel do lado esquerdo da atribuição.

c) Suponha que existe um contador de números de carros construídos. Pretende-se que esse contador considere também os automóveis que são construídos no contexto de um parâmetro de função passado por cópia. Acrescente/modifique o necessário à sua classe para que esta característica seja cumprida.

No final deste exercício deverá ter

- Consolidado a matéria de redefinição de operadores.
- Melhorado o entendimento acerca do operador de atribuição
- Melhorado o entendimento acerca do construtor por cópia

4. Tabela com operadores

Considere o exercício 1 da ficha 3: a classe Tabela. Obtenha esse código para um novo projeto e adicione o suporte às operações exemplificadas na função *main()* seguinte

```
int main() {
    Tabela a{10,15,20,25,30,25};
    cout << "elemento nas posições 2 e 3: " << a[2] << " " << a[3];
    // aparece os valores 20 e 25
    a[2] = 5;
    cout << "\na posição 2 tem agora: " << a[2];  // aparece 5
    cout << "\nNum. De elementos com valor 25 = " << a(25);
            // aparece 2
    a[500] = 33;
    cout << "esta mensagem não aparece";
}
```

No final deste exercício deverá ter

- Consolidado a matéria de redefinição de operadores.
- Experimentado redefinir o operador [] e ()
- Recordado o uso de referências e de exceções