POO - Carlos Eduardo Batista / Lista de Exercícios - Terceira Prova

1) Considere o código a seguir de um programa em C++:

a) Explique por que há duas implementações diferentes para a sobrecarga do operador de multiplicação (*) na classe Pixel. Qual é o motivo e propósito de cada implementação?

b) Qual a saída da execução do código (função main())? Explique o motivo, explicando também como funciona o tratamento de exceções em C++.

c) Explique como a classe Pixel pode ser generalizada usando templates da STL (*Standard Template Library*) para permitir o uso de coordenadas (x, y) de qualquer tipo arbitrário, assim como o atributo color. Como você usaria a classe Pixel generalizada com templates para criar um vetor com coordenadas do tipo *double* e o atributo color *char**?

```
#include <iostream>
#include <stdexcept>
class Pixel {
private:
    int x, y;
    std::string color;
public:
     Pixel(int _x = 0, int _y = 0, std::string
_color = "white") : x(_x), y(_y), color(_color)
{
        if (x < 0 || y < 0) {
      throw std::invalid_argument("COORD-NEG");
        }
        if (color.empty()) {
                                              throw
std::invalid_argument("COLOR-EMPTY");
        }
    }
    Pixel operator+(const Pixel& other) const {
         return Pixel(x + other.x, y + other.y,
color);
    }
    Pixel operator-(const Pixel& other) const {
         return Pixel(x - other.x, y - other.y,
color);
    }
   Pixel operator*(double scalar) const {
          return Pixel(x * scalar, y * scalar,
color);
    }
    friend Pixel operator*(double scalar, const
Pixel& pixel) {
        return pixel * scalar;
    }
     friend Pixel scale(double scalar, const
Pixel& pixel) {
        if (scalar == 0) {
 throw std::invalid_argument("DIV-ZERO");
        }
         return Pixel(pixel.x * scalar, pixel.y
* scalar, pixel.color);
    }
                        friend      std::ostream&
operator<<(std::ostream&   os,   const   Pixel&
pixel) {
         os << "Pixel at (" << pixel.x << ", "
<< pixel.y << ") with color " << pixel.color;
        return os;
    }
    void changeColor(std::string newColor) {
        if (newColor.empty()) {
          throw          std::invalid_argument(
"NEW-COLOR-EMPTY");
        }
        color = newColor;
    }
};
```

```
int main() {
    try {
        Pixel p1(10, 20, "red");
        std::cout << p1 << std::endl;
        p1.changeColor("blue");
        std::cout << "After color change: " << p1 << std::endl;

        Pixel p2(-5, 15, "green"); // This will throw an exception
        std::cout << p2 << std::endl;

        Pixel p3(30, 40, ""); // This will throw an exception
        std::cout << p3 << std::endl;
    } catch (const std::invalid_argument& e) {
        std::cerr << "Erro de argumento: " << e.what() << std::endl;
    } catch (const std::exception& e) {
        std::cerr << "Erro desconhecido: " << e.what() << std::endl;
    }
    return 0;
}
```

2) Suponha que eu queira armazenar objetos da versão original da classe Pixel em uma lista STL (std::list). Em um dado momento eu quero exibir o conteúdo dessa lista reversamente ordenado. Responda:

a) Como você poderia usar iteradores para imprimir cada elemento da lista? Forneça um esboço de código demonstrando como iteradores são usados para percorrer a lista e imprimir cada elemento.

b) Que operador você precisaria sobrecarregar na classe Pixel para permitir sua ordenação dentro da lista? Explique como a sobrecarga desse operador afetaria o processo de ordenação ao usar algoritmos da STL, como std::sort.

3) Explique a importância da sobrecarga de operadores em C++ e como ela facilita a criação de tipos de dados personalizados. Discuta os diferentes contextos em que a sobrecarga de operadores pode ser aplicada e forneça exemplos (esqueletos de código C++) para ilustrar cada contexto.

4) Explique a relação entre a sobrecarga de operadores em C++ e o uso de funções friend. Por que, em alguns casos, é necessário declarar uma função como friend para sobrecarregar um operador? Dê exemplos para ilustrar sua explicação.

5) O que são templates C++? Como templates são utilizados na construção dos recipientes (contêiners) STL?

6) O que são iteradores e qual a diferença entre os iteradores dos recipientes STL Vector, List e Set? Ressalte quais características e funcionalidades de C++ estão associadas à implementação e uso dos iteradores.

7) Crie dois esqueletos de código em C++ para comparar o uso dos seguintes recipientes: vector e list. Comente como o uso de STL facilitaria a troca de um recipiente pelo outro em cada caso.

8) Descreva a saída da execução do código abaixo:

```
#include <iostream>
using namespace std;
class DivisionException {
public:
 int type;
 DivisionException(int i) { type = i; }
};
class DivideByZeroException : public DivisionException {
public:
 DivideByZeroException() : DivisionException(1000){};
};
class InvalidOperandsException : public DivisionException
{
public:
 InvalidOperandsException() : DivisionException(1001){};
};
int func1(int x) {
 try {
    cout << "FUNC1()" << endl;
    throw 10;
    throw DivisionException(1020);
  }
 catch(int e) {
  cout << e << endl;
 }
```

```
catch(DivisionException& e){
  cout << "[A] DIV-EXCEP-0" << endl;
 }
 throw InvalidOperandsException();
 return 0;
}

int main() {
 cout << "INICIO DO PROGRAMA" << endl;
 try {
  func1(1);
 }
 catch (DivideByZeroException e) {
   cout << "[B] DIV-EXCEP-1" << endl;
   try {
    func1(2);
   }
   catch(DivisionException e) {
     cout << "[C] DIV-EXCEP-2" << endl;
   }
 }
 catch (InvalidOperandsException& e) {
   cout << "[D] INV-EXCEP-0 " << endl;
 }
 cout << "FIM DO PROGRAMA" << endl;
}
```