Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0030 – Linguagem de Programação I - T3

Docente: Umberto S. Costa

Problema: desenvolvimento de habilidades de programação na linguagem C++.

Subproblema 5: templates de classes e TADs.

Produto do subproblema: (*i*) resumo das principais características e recursos C++ identificados durante a exploração das questões deste subproblema (até duas páginas, podendo haver apêndices); (*ii*) respostas às questões abaixo; e (*iii*) código-fonte dos programas implementados.

Data de entrega via SIGAA: 27 de setembro de 2016.

Instruções: neste problema o aluno deve consultar as referências indicadas pelo docente para se familiarizar com os recursos necessários à criação de programas simples em C++, sem prejuízo à consulta de outras fontes como manuais e tutoriais. Usar as questões e programas mostrados a seguir como guia para as discussões em grupo e para orientar a exploração da linguagem C++. Para facilitar o aprendizado, recomenda-se que o aluno compare os recursos e conceitos de C++ com seu conhecimento prévio acerca de outras linguagens de programação. Leia e modifique os códigos mostrados e utilize os conceitos e recursos explorados para a criar os programas solicitados. Recursos exclusivos da linguagem C devem ser ignorados e substituídos por seus correspondentes em C++.

Questões[1]:

1. Considere a listagem a seguir, onde definimos um template para a classe `point`:

```
/** Listing 48-1. The point Class Template */
template<class T>
class point
{
public:
  point(T const& x, T const& y)
  : x_{x}, y_{y}
  {}

```

[1] Em parte inspiradas em *Exploring C++ 11*, Ray Lischner. Alguns programas foram retirados desta mesma fonte.

```
  point()
  : x_{}, y_{}
  {}

  T const& x()
  const
  {
    return x_;
  }

  T const& y()
  const
  {
    return y_;
  }
private:
  T x_;
  T y_;
};

int main() {}
```

lists/list4801.cpp

Nesta listagem as funções-membro (*métodos*) foram escritas utilizando diversas linhas para destacar o fato de que elas seguem a mesma estrutura de uma função. Perceba que, quando os métodos forem mais complexos e tomarem diversas linhas, os limites da definição da classe poderão ser tornar confusos. Como alternativa, podemos definir os métodos fora da definição da classe, conforme mostrado na listagem `list4801V2.cpp`, mostrada a seguir:

```
/** Listing 48-1, V2. The point Class Template */
template<class T>
class point
{
public:
  point(T const&, T const&);
  point();
  T const& x() const;
  T const& y() const;
private:
  T x_;
  T y_;
};

template <class T>
point<T>::point(T const& x, T const& y)
: x_{x}, y_{y}
{}

template <class T>
point<T>::point()
: x_{}, y_{}
{}
```

```
                                                                    24
template <class T>                                                  25
T const& point<T>::x()                                              26
const                                                               27
{                                                                   28
  return x_;                                                        29
}                                                                   30
                                                                    31
template <class T>                                                  32
T const& point<T>::y()                                              33
const                                                               34
{                                                                   35
  return y_;                                                        36
}                                                                   37
                                                                    38
int main() {}                                                       39
```

lists/list4801V2.cpp

Observe atentamente esta nova versão da classe e perceba que o nome de cada método é precedido com `point<T>::`, para indicar vinculação à classe `point`, que utiliza um nome de tipo `T`. Definições de métodos internas podem ser combinadas com definições externas a uma classe. Salve esta listagem com o nome `list4801V3.cpp` e acrescente dois novos métodos:

```
void move_to(T x, T y); /// mova o ponto para as coordenadas (x, y)
void move_by(T x, T y); /// adicione (x, y) à posição atual do ponto
```

Observe os comentários para definir os novos dois métodos, externamente à definição da classe.

2. Podemos representar os n elementos de um conjunto como um vetor com n posições. Pede-se:
   (a) Crie uma classe `conjunto` assumindo que:
      i. A classe deve suportar elementos numéricos. Utilize templates;
      ii. A cardinalidade de um conjunto deve ser conhecida por seu construtor;
      iii. Utilize níveis de acesso de forma a tornar sua classe segura;
      iv. Não existem repetições em um conjunto. Ignore elementos repetidos;
      v. Implemente métodos para **ler** conjuntos, **escrever** conjuntos, testar a **pertinência** de um elemento a um conjunto, calcular o conjunto resultante da **diferença**, da **união** e da **interseção** entre dois conjuntos. Crie os métodos auxiliares que julgar necessários;
      vi. A memória reservada deve ser liberada via um destrutor adequado.
   (b) Crie um programa principal que:
      i. Solicite e leia dois conjuntos A e B com elementos inteiros;
      ii. Compute e imprima $A - B$, $A \cup B$ e $A \cap B$.
3. Analise a listagem `list4802.cpp`, responsável por criar um novo Tipo de Dado Abstrato (TAD) para representar números racionais:

```
#include <iostream>                                                 1
                                                                    2
template<class T>                                                   3
class rational                                                      4
```

```
{
public:
  typedef T value_type;
  rational() : rational{0} {}
  rational(T num) : numerator_{num}, denominator_{1} {}
  rational(T num, T den);

  void assign(T num, T den);

  template<class U>
  U convert()
  const
  {
    return static_cast<U>(numerator()) / static_cast<U>(denominator());
  }

  T numerator() const { return numerator_; }
  T denominator() const { return denominator_; }
private:
    void reduce();
    T numerator_;   // numerator gets the sign of the rational value
    T denominator_; // denominator is always positive
};

// Construct a rational object, given a numerator and a denominator.
template<class T>
rational<T>::rational(T num, T den)
: numerator_{num}, denominator_{den}
{
  rational<T>::reduce();
}

// Assign a numerator and a denominator, then reduce to normal form.
template<class T>
void rational<T>::assign(T num, T den)
{
  numerator_ = num;
  denominator_ = den;
  rational<T>::reduce();
}

// Reduce the numerator and denominator by their GCD.
template<class T>
void rational<T>::reduce(){}

// Compare two rational numbers for equality.
template<class T>
bool operator==(rational<T> const& a, rational<T> const& b)
{
  return a.numerator() == b.numerator() and
         a.denominator() == b.denominator();
}
```

```
57
// Compare two rational numbers for inequality.                    58
template<class T>                                                  59
inline bool operator!=(rational<T> const& a, rational<T> const& b) 60
{                                                                  61
  return not (a == b);                                             62
}                                                                  63
                                                                   64
int main() {                                                       65
 rational<short> zero{};                                           66
 rational<int> pi1{355, 113}, x{1,2}, y{2, 4};                     67
 rational<long> pi2{80143857L, 25510582L};                         68
                                                                   69
 if (x == y)                                                       70
   std::cout << "1/2 == 2/4" << std::endl;                         71
 else                                                              72
   std::cout << "1/2 != 2/4" << std::endl;                         73
}                                                                  74
```

lists/list4802.cpp

Neste TAD, o tipo do numerador e do denominador é parametrizado via template, de forma que o usuário pode usar os tipos `short`, `int` ou `long`, de acordo com a precisão desejada. Pede-se:

(a) Observe a convenção utilizada na linha `7`, onde o `typedef` é usado para acessar o parâmetro do template de uma forma padronizada. Por exemplo, `vector<char>::value_type` é um `typedef` para `char`. Esta linha pode ser omitida caso não desejemos seguir tal padrão. Salve e compile esta listagem utilizando o nome `list4802V2.cpp`.

(b) Note, nas linhas `14` a `19`, a definição de um template que utiliza um segundo parâmetro para o método `convert()`. O que significa o a palavra-chave `static_cast`? Mostre um exemplo de uso do método `convert()`.

(c) Os resultados de execução das linhas `70` a `73` da listagem `list4802.cpp` são os esperados? O que deu errado? Corrija este problema definindo o método `reduce()` na listagem `list4802V2.cpp`. Dica: crie uma função privada para calcular o MDC do numerador e do denominador do racional a ser reduzido, ela poderá auxiliar nos cálculos a serem feitos.

(d) Qual o efeito do uso da palavra-chave `inline` (linha `60`)?

(e) Nas linhas `51` a `63` duas funções independentes da classe são definidas como operadores sobrecarregados de igualdade e diferença entre operandos do tipo `rational<T>`, enquanto na linhas `70` usa-se um desses operadores. Modifique seu programa `list4802V2.cpp` para incluir os operadores $<$, $\leq$, $>$, $\geq$ sobre o tipo `rational<T>`. Teste seus novos operadores.

4. Salve a última versão da listagem `list4802V2.cpp` como `list4803.cpp`.

(a) Inclua, na listagem `list4803.cpp`, operadores aritméticos unários e binários.

(b) Qual o resultado se utilizarmos a seguinte função principal em `list4803.cpp`?

```
int main() {
  rational<int> r{(rational<int>)3 * rational<int>{1, 3}};
  std::cout << r.numerator() << '/' << r.denominator() << std::endl;
}
```