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

Docente: Umberto S. Costa

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

Subproblema 4: introdução à Programação Orientada a Objetos (POO).

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: 19 de setembro de 2017.

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 o programa a seguir:

```
/** @file list3103.cpp */                                    1
/** Listing 31-3. Using a Class and its Members */           2
#include <iostream>                                          3
#include <ostream>                                           4
                                                             5
struct point                                                 6
{                                                            7
  double x;                                                  8
  double y;                                                  9
```

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

```
};

int main()
{
  point origin{}, unity{};
  origin.x = 0;
  origin.y = 0;
  unity.x = 1;
  unity.y = 1;
  std::cout << "origin = (" << origin.x << ", " << origin.y << ")\n";
  std::cout << "unity  = (" << unity.x  << ", " << unity.y  << ")\n";
}
```

lists/list3103.cpp

No tipo `point` os campos da estrutura definem os *membros de dados* ou *atributos* do tipo. Observe atentamente esta listagem e responda:

(a) Variáveis `point` são inicializadas?

(b) Execute o código e note que cada variável `point` tem sua versão dos membros de dados.

2. Redefina do tipo `point` da questão anterior de acordo com a listagem `list3104.cpp`:

```
/** @file list3104.cpp */
/** Listing 31-4. Member Functions for Class point */
#include <cmath> // for sqrt and atan
struct point
{
  /// Distance to the origin.
  double distance()
  {
    return std::sqrt(x*x + y*y);
  }
  /// Angle relative to x-axis.
  double angle()
  {
    return std::atan2(y, x);
  }

  /// Add an offset to x and y.
  void offset(double off)
  {
    offset(off, off);
  }
  /// Add an offset to x and an offset to y
  void offset(double  xoff, double yoff)
  {
    x = x + xoff;
    y = y + yoff;
  }

  /// Scale x and y.
  void scale(double mult)
  {
```

```
    this->scale(mult, mult);                                    32
  }                                                             33
  /// Scale x and y.                                           34
  void scale(double xmult, double ymult)                        35
  {                                                             36
    this->x = this->x * xmult;                                  37
    this->y = this->y * ymult;                                  38
  }                                                             39
  double x;                                                     40
  double y;                                                     41
};                                                              42
```

lists/list3104.cpp

(a) Identifique as *funções membro* adicionadas. Como essas funções são acessadas?

(b) Observe as linhas `32`, `37` e `38` e responda:

i. Qual o significado e funcionamento esperado destas linhas?

ii. Os acessos providos nestas linhas (dados na forma `this->x`) podem ser substituídos por acessos da forma `(*this).x`? Explique.

iii. Podemos simplificar os acessos eliminando o `this->` destas linhas? Explique.

(c) Crie em programa que peça dois pontos e imprima suas distâncias em relação à origem. Salve seu código como `q2.cpp`.

3. Considere a seguinte listagem:

```
/** @file list3105.cpp */                                       1
/** Listing 31-5. Constructors for Class point */               2
                                                                3
struct point                                                    4
{                                                               5
  point()                                                       6
  : point{0.0, 0.0}                                             7
  {}                                                            8
  point(double x, double y)                                     9
  : x_{x}, y_{y}                                                10
  {}                                                            11
  point(point const& pt)                                        12
  : point{pt.x_, pt.y_}                                         13
  {}                                                            14
  double x_;                                                    15
  double y_;                                                    16
};                                                              17
```

lists/list3105.cpp

Pede-se:

(a) Qual a função de um construtor?

(b) Explique o funcionamento de cada construtor:

i. Explique como ocorre o casamento entre a *lista de inicialização* e os membros de dados.

ii. O que faz o construtor das linhas `9` a `11`? Podemos omiti-lo? Explique.

iii. Identifique o *construtor por cópia*. Precisamos explicitar este construtor?

iv. O que é um *construtor implícito* (ou *default*)?

(c) Observe o padrão utilizado para identificar membros de dados (linhas 15 e 16). Utilizaremos este padrão de agora em diante para facilitar a identificação dos membros de dados.

4. Qual o comportamento esperado do seguinte programa?

```
/** @file list3106.cpp */
/** Listing 31-6. Visual Constructors */
#include <iostream>

struct demo {
  demo()       : demo{0} { std::cout << "default constructor\n"; }
  demo(int x) : x_{x} { std::cout << "constructor(" << x << ")\n"; }
  demo(demo const& that)
  : x_{that.x_}
    {
      std::cout << "copy constructor(" << x_ << ")\n";
    }
  int x_;
};

demo addone(demo d)
{
  ++d.x_;
  return d;
}

int main() {
  demo d1{};
  demo d2{d1};
  demo d3{42};
  demo d4{addone(d3)};
}
```

lists/list3106.cpp

5. Qual o comportamento esperado da seguinte listagem?

```
struct point
{
  point() = default;
  point(point const&) = delete;
  int x, y;
};
```

lists/list3107.cpp

6. Considere o seguinte código:

```
/** @file list3301.cpp */
/** Listing 33-1. The point Class with Access Level Specifiers */
#include <iostream>
#include <cmath>

```

```
struct point                                                                    6
{                                                                               7
public:                                                                         8
  point() : point{0.0, 0.0} {}                                                  9
  point(double x, double y) : x_{x}, y_{y} {}                                   10
  point(point const&) = default;                                                11
  ~point() { std::cout << "~point" << std::endl; } // destrutor da classe point 12
                                                                                13
  double x() const { return x_; }                                               14
  double y() const { return y_; }                                               15
                                                                                16
  double angle()    const { return std::atan2(y(), x()); }                      17
  double distance() const { return std::sqrt(x()*x() + y()*y()); }              18
                                                                                19
  void move_cartesian(double x, double y)                                       20
  {                                                                             21
    x_ = x;                                                                     22
    y_ = y;                                                                     23
  }                                                                             24
  void move_polar(double r, double angle)                                       25
  {                                                                             26
    move_cartesian(r * std::cos(angle), r * std::sin(angle));                   27
  }                                                                             28
                                                                                29
  void scale_cartesian(double s)       { scale_cartesian(s, s); }               30
  void scale_cartesian(double xs, double ys)                                    31
  {                                                                             32
    move_cartesian(x() * xs, y() * ys);                                         33
  }                                                                             34
  void scale_polar(double r)           { move_polar(distance() * r, angle()); } 35
  void rotate(double a)                { move_polar(distance(), angle() + a); } 36
  void offset(double o)                { offset(o, o); }                        37
  void offset(double xo, double yo)    { move_cartesian(x() + xo, y() + yo); }  38
                                                                                39
private:                                                                        40
  double x_;                                                                    41
  double y_;                                                                    42
};                                                                              43
                                                                                44
int main(){                                                                     45
  point p1, p2;                                                                 46
}                                                                               47
```

lists/list3301.cpp

(a) Identifique os níveis de acesso introduzidos nesta listagem. O que eles significam?

(b) O que faz um destrutor (linha `12`)? O que este programa exibirá?

(c) Qual o efeito dos modificadores `const` apresentados nas linhas `14` a `18`?

(d) As funções membro públicas desta listagem permitem trabalhar em coordenadas cartesianas, especificando um ponto por suas posições x e y, ou em coordenadas polares, especificando um ponto por meio de um ângulo relativo ao eixo x e de uma distância em relação

à origem. Independente da representação, esta implementação não permite acesso direto aos membros de dados, assim você pode mudar a representação de um ponto (atualmente $x\_$ e $y\_$) armazenando seu ângulo e distância ($angle\_$ e $r\_$) sem afetar a interface desta classe. Para efetuar tal mudança, quais funções membro precisariam ser alteradas?

(e) Reescreva a classe `point` conforme descrito no item anterior. Assuma que o construtor `point(double x, double y)` recebe coordenadas cartesianas, que deverão ser transformadas na representação polar equivalente. Salve seu código como `list3302.cpp`.

(f) Na listagem `list3302.cpp` produzida no item anterior, substitua a palavra-chave `struct` por `class`. Note que ambas as palavras-chave podem ser utilizadas para definir classes, mudando apenas o nível de acesso padrão: `private` para `class` e `public` para `struct`.

7. No **Problema 3** definimos o tipo `Vector` (questão 5) utilizando um tipo `struct` e funções auxiliares. Naquele momento não tínhamos as noções sobre classes, de forma que os dados e as funções foram implementados sem integração explícita. Com base nos conceitos aprendidos neste problema, proponha uma nova solução à questão citada. Utilize a palavra-chave `class` e se beneficie dos níveis de acesso: as funções membro não deverão ter acesso direto aos membros de dados, mas apenas acessa-los por meio de funções públicas específicas. Um sinal de que você fez boas escolhas quanto à modelagem de sua solução, incluindo níveis de acesso, será a facilidade com a qual as funções membro poderão ser modificadas. Salve seu código em `q7.cpp`.

8. Com base nos conceitos relativos às classes, implemente o tipo `Matrix` e as funcionalidades relacionadas conforme pedido na questão 6 do **Problema 3**. Salve seu código como `q8.cpp`.