UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
IMD0030 – Linguagem de Programação I

Lista de Exercícios 01
26/03/2019

Questão 1. Implemente uma classe Carro. Um carro é definido por seu modelo e ano. Considere ainda que todo carro possui um tanque de combustível com capacidade máxima de X litros de gasolina e com consumo de Y km/litro. Sua implementação deve permitir: abastecer o carro com uma certa quantidade de gasolina; mover o carro em uma determinada distância (medida em km); informar a quantidade de combustível atual; informar a distância total percorrida; e a autonomia do carro, ou seja, quantos quilometros ainda será possível percorrer com a quantidade de combistível atual.

Acho que não é preciso dizer que o carro não se move sem combustível. Na verdade, o carro so deve se mover se houver mais do que 1L (um litro) de combustível no tanque.

Caso você tenha feito tudo como esperado, utilize o seguinte programa para testar a sua implementação.

```
1  #include "carro.h"
2  #include <iostream>
3
4  using namespace std;
5
6  int main(int argc, char const *argv[])
7  {
8    Carro meuCarro("Fusca", 1976, 40, 7);
9    meuCarro.mover(5);
10   meuCarro.abastecer(35);
11   cout << "Abastecido! Temos agora " << meuCarro.getQtdeCombustivel()
             << " litros." << endl;
12   cout << "Autonomia atual de " << meuCarro.getAutonomia() << "km."
13        << endl;
14   meuCarro.mover(127.6);
15   meuCarro.mover(3.2);
16   cout << "Ja percorremos " << meuCarro.getDistanciaPercorrida()
17        << "km" << endl;
18   cout << "Ainda temos " << meuCarro.getQtdeCombustivel()
19        << " litros." << endl;
20   cout << "Autonomia atual de " << meuCarro.getAutonomia() << "km."
21        << endl;
22   return 0;
23 }
```

A execução do programa acima deverá produzir como saída:

```
Foi criado um Fusca, ano 1976. Suporta 40 litros e consome 7 km/L.
Combustivel insuficiente para mover.
Abastecido! Temos agora 35 litros.
Autonomia atual de 238km.
Ja percorremos 130.8km
Ainda temos 16.3143 litros.
Autonomia atual de 107.2km.
```

Questão 2. Crie uma classe para representar uma pessoa, com os atributos privados de nome, idade e altura. Crie os métodos getters e setters necessários e sobrecarregue o operador de inserção («) para permitir imprimir os dados de uma pessoa.

Questão 3. Utilizando a sua implementação da classe Pessoa da questão anterior, crie uma classe Agenda que armazena até 100 contatos (pessoas) e seja capaz de operações como: inserir um novo contato na agenda, remover um contato existente, buscar um contato na agenda, listar todos os contatos ou aqueles que iniciam por uma determinada letra, além de listar os dados de um contato específico.

```
1  class Agenda{
2  public:
3    /* armazena um novo contato */
4    void insereContato(string nome_, int idade_, float altura_);
5
6    /* remove um contato pelo nome  e reorganiza a agenda */
7    void removeContato(string nome_);
8
9    /* retorna o indice para o contato ou -1 caso nao exista */
10   int buscaContato(string nome_);
11
12   /* lista todos os contatos ou apenas aqueles que
13      iniciam por uma determinada letra */
14   void listaContato(char letra = '');
15
16   /* imprime os dados do contato armazenado na posicao i */
17   void imprimeContato(int i);
18 private:
19   Pessoa m_contatos[MAX_CONTATOS];
20   int m_total_contatos;
21 };
```

Crie um programa para testar a implementação de sua agenda.

Questão 4. Escreva um programa para representar datas a partir dos atributos dia, mês e ano.

Sua classe Data deve implementar um construtor que inicializa os três atributos já em sua instanciação. Forneça também um construtor padrão que inicializa os atributos de acordo com a data atual fornecida pelo sistema operacional (isto deve envolver alguma pesquisa de sua parte. Google it!!).

Efetue a sobrecarga do operador(«) para retornar uma representação da data como uma string. Considere que a data deve ser formatada mostrando o dia, o mês e ano separados por barra (/).

Implemente os métodos *somarDias(int quantidade)*, *somarMeses(int quantidade)* e *somarAnos(int quantidade)* que permita avançar a data atual numa *quantidade* de dias, meses ou anos passada por parâmetro.

Implemente também o método *proximoDia()* que permita avançar a data atual para o dia seguinte.

Por fim, escreva uma classe de teste que demonstra as capacidades da classe Data.

Questão 5. Implemente uma classe que represente um livro, com os atributos título, autor, edição, editora, ano e isbn. Crie os métodos de acesso (getters e setters) necessários e sobrecarregue o operador de inserção («) para permitir imprimir os dados do livro.

Questão 6. Utilizando a sua implementação da classe livro da questão anterior, crie um programa que simule o comportamento de uma biblioteca, destinada apenas para empréstimos. Seu programa deverá permitir as seguintes operações:

- Buscar livros pelo nome;
- Buscar livros pelo isbn;
- Verificar Se um livro existe na biblioteca;
- Verificar a quantidade disponível para empréstimo.

Questão 7. Faça um programa que simule um jogo de bingo. Para isso, você deverá implementar as classes: Sorteadora, Jogador, Cartela e Bingo. Sorteadora deverá representar a máquina que sorteia automaticamente um números, não repetidos, entre 1 e 99. Cartela representa uma cartela com 15 números que são determinados aleatoriamente quando da criação da cartela. Jogador representa cada indivíduo que participa do jogo. Um jogador pode ter até 5 cartelas. Bingo deve simular as iterações entre os objetos sorteadora, jogadores e cartelas. Seu programa deverá permitir adicionar multiplos jogadores e sortear os numeros até que algum dos jogadores seja campeão. Ao final, deverá ser impresso os dados do jogador campeão, assim como os dados da cartela campeã.

Questão 8. Considere um programa em C++ que leia um tempo no formato HH:MM:SS e imprima o total em segundos, usando uma classe Tempo. Mais uma vez, Teobaldo já iniciou o código, mas precisa de sua ajuda para completar.

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  class Tempo
7  {
8      private:
9          int hh,mm,ss;
10     public:
11         // Le os dados do tempo a partir da entrada padrao
12         void lerTempo(void);
13         // Retorna o tempo em segundos
14         int converteEmSegundos(void);
15         // Imprime o tempo no formato HH:MM:SS e o seu total
16         // em segundos
17         void mostraTempo(void);
18 };
19
20 // Implementar os metodos...
21
22 int main()
```

```
23 {
24     Tempo T;
25
26     T.lerTempo();
27     T.mostraTempo();
28
29     return 0;
30 }
```

Questão 9. Mostre ao Teobaldo o que você aprendeu sobre sobrecarga de operadores e altere o código do programa da questão anterior para substituir os métodos lerTempo() e mostraTempo() pelos operadores de extração e inserção, respectivamente.

Questão 10. Ao tentar compilar os códigos abaixo, Janice ficou triste com a quantidade de erros apontados pelo compilador. Ajude Janice explicando cada erro apontado pelo compilador e apresentando a correção necessária. Patra cada erro que você encontrar, aponte-o usando a notação NOMEDOARQUIVO:LINHA e explique a razão do erro e o que deve ser feito para corrigi-lo.

Exemplo: **funcionario.cpp:2** - O tipo Funcionario não será reconhecido. A solução é inserir a linha *#include "funcionario.h"* no início do arquivo.

NÃO BASTA apresentar o código corrigido, pois Janice quer aprender e não apenas a solução!

```
1  /* Definicao da classe Funcionario: funcionario.h */
2  #ifndef _FUNCIONARIO_H_
3  #define _FUNCIONARIO_H_
4
5  #include <string>
6
7  class Funcionario {
8    private:
9        string m_empresa;
10     int m_matricula;
11     string m_nome;
12     double m_salario;
13     int getNome();
14     void setNome(string nome);
15     int getMatricula();
16     void setMatricula(int matricula);
17   public:
18     Funcionario (int matricula, string nome, double salario);
19     ~Funcionario();
20 };
21 #endif
```

```
1  /* Implementacao da classe Funcionario: funcionario.cpp */
2  Funcionario::Funcionario (int matricula, string nome, double salario):
3      m_matricula(matricula),m_nome(nome),m_salario(salario){}
4
5  Funcionario::~Funcionario(){}
6
```

```
 7  int
 8  Funcionario::getNome(){
 9      return m_nome;
10  }
11
12  void
13  Funcionario::setNome(string nome){
14      m_nome = nome;
15  }
16
17  int
18  Funcionario::getMatricula(){
19      return m_matricula;
20  }
21
22  void
23  Funcionario::setMatricula(int matricula){
24      m_matricula = matricula;
25  }
```

```
 1  /* Programa principal: main.cpp */
 2  #include <iostream>
 3  #include "funcionario.h"
 4
 5  string Funcionario::m_empresa = "JBS";
 6
 7  int main(int argc, char const *argv[])
 8  {
 9      /* Instancia um novo funcionario da empresa JBS */
10    Funcionario novo;
11    novo.m_matricula = 12345;
12    novo.setNome("Sandra Aparecida Sumiu");
13    novo.setSalario(2387.50);
14    std::cout << "Dados do novo funcionario: " << novo << std::endl;
15    return 0;
16  }
```

Questão 11. Alice, Bob e Carl resolveram iniciar uma aventura RPG, mas antes, para decidir quem comandaria o jogo, decidiram jogar 6 (seis) dados com diferentes dimensões, nomeadamente: Tetrahedron (três lados e valores de 1 a 4), Cube ou Hexahedron (6 lados e valores de 1 a 6), Octahedron (8 faces e valores de 1 a 8), Decahedron (10 faces e valores de 1 a 10), Dodecahedron (12 faces e valores de 1 a 12) e Icosahedron (20 faces e valores de 1 a 20). Assim, aquele que obtiver o MENOR valor agregado, será declarado o vencedor e comandará esta jornada de RPG.

Considere como implementação base a definição e implementação da classe Dado, apresentada a seguir. Realize as alterações necessárias necessárias à classe Dado para permitir que um programa que simule a situação de Alice, Bob e Carl, permitindo indicar o vencedor, utilize esta implementação do dado.

```
1  /* Definicao da classe Dado: dado.h */
2  #ifndef _DADO_H_
```

```
3  #define _DADO_H_
4
5  #include <random>
6
7  class Dado {
8    private:
9      int m_valor;
10     std::random_device m_rd;
11     std::default_random_engine m_gen;
12     std::uniform_int_distribution<> m_dis;
13   public:
14     Dado ();
15     int jogar();
16     int getValor();
17 };
18
19 #endif
```

```
1  /* Implementacao da classe Dado: dado.cpp */
2  #include <random>
3  #include "dado.h"
4
5  Dado::Dado():m_gen(m_rd()), m_dis(1, 6) {
6    m_valor = std::round(m_dis(m_gen));
7  }
8
9  int
10 Dado::jogar() {
11   m_valor = std::round(m_dis(m_gen));
12   return m_valor;
13 }
14
15 int
16 Dado::getValor() {
17   return m_valor;
18 }
```

Questão 12. A solução de problemas utilizando o paradigma de Programação Orientada a Objetos é essencialmente baseada na ideia de abstração. O que significa esse conceito e como é possível representar entidades do mundo real na forma de classes e objetos em um programa orientado a objetos?

Questão 13. Explique a função dos modificadores de visibilidade `public` e `private` na definição de membros de uma classe em C++. Por padrão, qual a visibilidade aplicada aos membros de uma classe em C++? Justifique sua resposta apresentando situações em que um ou outro tipo de visibilidade poderia/deveria ser utilizado.

Questão 14. Para que servem membros estáticos de uma classe em C++? Justifique com um exemplo.

Questão 15. Apresente, de forma precisa, a distinção entre os operadores . (ponto) e :: (duplo dois-pontos) e como eles são utilizados no contexto de classes, objetos e métodos em C++. São exemplos de uso dos operadores as instruções aluno.matricula e Turma::getAlunos().

Questão 16. Por que a sobrecarga dos operadores de inserção («) e extração (») de dados em streams é feita de forma diferente dos operadores convencionais? Qual a função da definição friend para um método ou classe no C++?

Questão 17. Explique a utilização de métodos construtores (padrão, parametrizado e cópia), métodos destrutores e de métodos *getters* e *setters* na definição de classes em C++.

Questão 18. Leia a descrição a seguir e complete o código do Teobaldo utilizando os conceitos discutidos em aula.

Teobaldo, aluno da ênfase Computação Mística no curso BTI (Bacharelado em Trabalhos Impossíveis) recebeu de seu professor Tony Estarky a tarefa de implementar em C++ as classes, atributos e métodos (incluindo construtores e destrutor) e programa principal necessários para simular um cadastro de turmas bem básico, atendendo às seguintes especificações gerais:

- Cada turma deve permitir informar a listagem dos alunos na turma, a quantidade de alunos e a média das notas dos alunos;
- Sobre cada aluno são guardadas algumas informações básicas, tais como matricula, como nome completo, total de faltas e nota (uma apenas por turma);
- Cada turma concentra um conjunto de alunos;
- Não deve ser permitida a inclusão duplicada de alunos na mesma turma;

Teobaldo sabia que seu professor iria exigir que ele utilizasse todos os conceitos vistos em aula na resolução deste problema. Infelizmente, Teobaldo teve alguns imprevistos e teve que viajar. Mas ele pediu a você para que complete a atividade da forma mais completa possível.

Ele pede desculpas, pois toda a implementação deve ser entregue ainda hoje, ao final desta avaliação. Para ajudar, ele deixou parte de seu código e alguns comentários e anotações.

Seguem as anotações e códigos de Teobaldo...

Após muito pensar, resolvi abstrair as classes *Aluno* e *AlunoTurma*. A primeira a ser usada para representar um Aluno do mundo real e a segunda para representar a participação de uma aluno em uma turma específica. Afinal, conclui que o Aluno é um só, mesmo participando de várias turmas. Segue a minha implementação incompleta destas classes.

```
1  #ifndef _ALUNO_H_
2  #define _ALUNO_H_
3
4  #include <string>
5
6  class Aluno {
7  private:
8    std::string matricula;
9    std::string nome;
10   int faltas;
11   double nota;
```

```
12  public:
13    Aluno();
14    Aluno(std::string _matricula, std::string _nome);
15  };
16
17  class AlunoTurma {
18  private:
19    Aluno* discente;
20    int faltas;
21    double nota;
22  public:
23    AlunoTurma();
24    AlunoTurma(Aluno* _discente, int _faltas, double _nota);
25    Aluno* getDiscente();
26  };
27
28  #endif
```

Eu sei que ainda falta muita coisa e me procupa a sobrecarga de operadores. Espero que você não se esqueça disso! Penso, pelo menos, em sobrecarregar os operadores de inserção e de igualdade.

Para representar uma Turma, defini a classe *Turma*, como mostrado a seguir.

```
1  #ifndef _TURMA_H_
2  #define _TURMA_H_
3
4  #include "aluno.h"
5  #include <vector>
6
7  class Turma
8  {
9  private:
10   std::vector<AlunoTurma> alunos;
11 public:
12   Turma();
13   ~Turma();
14   int addAluno(AlunoTurma _aluno);
15   Aluno* buscaAlunoPorNome (std::string _nome);
16   Aluno* buscaAlunoPorMatricula (std::string _matricula);
17   int removeAlunoPorNome (std::string _nome);
18   int removeAlunoPorMatricula (std::string _matricula);
19   void listaAlunos();
20 };
21
22 #endif
```

Até iniciei a implementação da classe *Turma*, mas não tive tempo de completar. Sei que a listagem irá manipular diferentes objetos e gostaria de ter iniciado o código, mas confio em sua capacidade. Segue o conteúdo do arquivo *turma.cpp*.

```
1  #include <iostream>
2  #include "turma.h"
3
```

```
4  Turma::Turma() {}
5
6  Turma::~Turma() {}
7
8  int
9  Turma::addAluno(AlunoTurma _aluno) {
10   /* Adiciona um aluno na lista de alunos */
11   return -1;
12 }
13
14 Aluno*
15 Turma::buscaAlunoPorNome (std::string _nome) {
16   /* Busca por um aluno na lista de alunos com o nome indicado.
17      Retorna um apontador para o objeto aluno se encontrado.
18      Retorna nulo, caso n o encontre um aluno com o nome indicado. */
19   return NULL;
20 }
21
22 Aluno*
23 Turma::buscaAlunoPorMatricula (std::string _matricula) {
24   /* Busca por um aluno na lista de alunos com a matricula indicada.
25        Retorna um apontador para o objeto aluno se encontrado.
26        Retorna nulo, caso n o encontre um aluno com a matricula
             indicada. */
27   return NULL;
28 }
29
30 int
31 Turma::removeAlunoPorNome (std::string _nome) {
32   /* Remove o aluno com o nome indicado.
33      Retorna 0 caso o aluno seja encontrado e removido com sucesso.
34      Retorna -1 em caso contr rio.
35      Dica: Para remover um elemento do vetor, utilize o metodo erase()
             .
36   */
37   return -1;
38 }
39
40 int
41 Turma::removeAlunoPorMatricula (std::string _matricula) {
42     /* Remove o aluno com a matricula indicada.
43      Retorna 0 caso o aluno seja encontrado e removido com sucesso.
44      Retorna -1 em caso contr rio.
45      Dica: Para remover um elemento do vetor, utilize o metodo erase()
             .
46   */
47   return -1;
48 }
49
50 void
51 Turma::listaAlunos() {
```

```
52    /* Lista os dados de todos os alunos da turma, incluindo o total de
         faltas e nota. Utilize a sobrecarga do operador de inser  o
         para a impress o dos dados do aluno. Deve listar ainda a
         quantidade de alunos e a m dia das notas dos alunos nesta turma.
53    */
54  }
```

Com a ajuda do professor, eu já tinha preparado um arquivo para testar a implementação das classes. Assim, NÃO ALTERE ESTE ARQUIVO! Se você completar corretamente as implementações que faltam, todo o código neste arquivo deve funcionar sem problemas. Se não funcionar, volte à definição e implementação das classes, pois o problema deverá estar por lá. Segue o código do arquivo *main.cpp*.

```
1  #include <iostream>
2  #include "turma.h"
3
4  int main(int argc, char const *argv[])
5  {
6    Turma t;
7    t.addAluno(AlunoTurma(new Aluno("00001.2017","Paulo"),4,7.30));
8    t.addAluno(AlunoTurma(new Aluno("00002.2017","Maria Luiza"),0,5.70))
         ;
9    t.addAluno(AlunoTurma(new Aluno("00005.2017","Adrina Ribeiro")
         ,6,9.75));
10   t.listaAlunos();
11   Aluno* encontrado = t.buscaAlunoPorNome("Maria Luiza");
12   if (encontrado) {
13     std::cout << encontrado->getNome() << " encontrado." << std::endl;
14   } else {
15     std::cout << " Aluno nao encontrado." << std::endl;
16   }
17   t.removeAlunoPorNome("Maria Luiza");
18   t.listaAlunos();
19   encontrado = t.buscaAlunoPorNome("Maria Luiza");
20   if (encontrado) {
21     std::cout << encontrado->getNome() << " encontrado." << std::endl;
22   } else {
23     std::cout << "Aluno nao encontrado." << std::endl;
24   }
25   return 0;
26 }
```