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

Lista de Exercícios 01
11/10/2018

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. Teobaldo está estudando o mecanismo de herança em C++, mas está com dificuldades em codificar a seguinte hierarquia de classes. Ele sabe que a função main() está correta, mas que há problemas com as definições de herança.

Aponte os problemas de uso de herança no código do Teobaldo, descrevendo claramente o problema e proponha as correções necessárias para que o programa funcione corretamente.

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Pessoa {
6  public:
7    string m_nome;
8    int m_idade;
9  public:
10   Pessoa(string nome_, int idade_): m_nome(nome_){};
11   ~Pessoa(){};
12   string getNome(){ return m_nome; };
13   int getIdade(){ return m_idade; };
14 };
15
16 class Empregado : private Pessoa {
17 private:
18   string m_matricula;
19   double m_salario;
20 public:
21   Empregado(string nome_, int idade_, string matricula_, double
        salario_):
22     Pessoa(nome_, idade_), m_matricula(matricula_), m_salario(salario_
         ){};
23   ~Empregado(){};
24   string getMatricula(){ return m_matricula; };
25   double getSalario(){ return m_salario; };
26 };
27
28 class Aluno : public Pessoa {
29 private:
```

```
30    string m_matricula;
31  public:
32    Aluno(string nome_, int idade_, string matricula_):
33      Pessoa(nome_, idade_), m_matricula(matricula_){};
34    ~Aluno(){};
35    string getMatricula(){ return m_matricula; };
36  };
37
38  class Vendedor : private Empregado {
39  private:
40    double m_meta_mensal;
41    double m_desconto_nivel1;
42  public:
43    Vendedor(string nome_, int idade_, string matricula_, double
          salario_, double meta_, double desconto_):
44      Empregado(nome_, idade_, matricula_, salario_),
45      m_meta_mensal(meta_), m_desconto_nivel1(desconto_){};
46    ~Vendedor(){};
47    double getMetaMensal(){ return m_meta_mensal; };
48    double getDescontoN1(){ return m_desconto_nivel1; };
49  };
50
51  class Gerente : private Empregado {
52  private:
53    string m_setor;
54    double m_desconto_nivel2;
55  public:
56    Gerente(string nome_, int idade_, string matricula_, double salario_
          , string setor_, double desconto_):
57      Empregado(nome_, idade_, matricula_, salario_),
58      m_setor(setor_), m_desconto_nivel2(desconto_){};
59    ~Gerente(){};
60    string getSetor(){ return m_setor; };
61    double getDescontoN2(){ return m_desconto_nivel2; };
62    void imprimeDados(){ cout << "Nome: " << m_nome  << "\t"
63              << "Idade" << m_idade << "\t"
64              << "Matricula: " << getMatricula() << endl
65              << "Salario: " << getSalario() << "\t"
66              << "Setor: " << m_setor << endl
67              << "Desconto N2: " << m_desconto_nivel2
68              << endl; };
69  };
70
71  int main(int argc, char const *argv[])
72  {
73    Aluno a("Maria de Lourdes", 22, "98765432-1");
74    Gerente g("Emiliano Emilio", 45, "666000666-1", 5780.00, "
          Departamento Financeiro", 7.5);
75    g.imprimeDados();
76    return 0;
77  }
```

Indique e justifique a ordem de construção e destruição de objetos das classes Aluno e Gerente.

Questão 11. Discuta as implicações de definirmos o seguinte template na classe ou programa principal em C++.

```
1 template <typename T>
2 ostream& operator<<(ostream& ostr, const T &x)
3 {
4     x.print(ostr);
5     return ostr;
6 }
```

Questão 12. Considere um contexto de uma oficina mecânica de grande porte. Na oficina, trabalham vários Funcionarios e apenas um Gerente. Além dos dados comuns a uma Pessoa, funcionários e gerentes possuem uma matrícula e um salário associados a eles. Porém, funcionário é uma definição genérica, de modo que não há nenhum simples funcionário, mas sim as figuras do Assistente Técnico e do Assistente Administrativo. Um Assistente Técnico, que também é um funcionário, possui um supervisor (que também é um funcionário) associado a ele. Além disso, os Assistentes Técnicos possuem um bônus salarial. Um Assistente Administrativo, por sua vez, possue um turno (dia ou noite) definido, além de um adicional de participação nos lucros da empresa.

Com base na descrição acima, implemente um conjunto de classes em C++ que permita representar o contexto da oficina mecânica, aplicando seus conhecimentos em POO, principalmente herança.

Questão 13. Crie uma classe Ingresso que possui um valor em reais e um percentual de impostos. Além dos métodos contrutor, destrutor e de acesso, implemente os métodos getTotal() e getImposto() que deverão retornar o valor total do ingresso e o valor do imposto (em reais) aplicado ao ingresso. Crie ainda uma classe IngressoVIP, que deriva da classe Ingresso e possui um valor adicional. O percentual de impostos aplicado ao IngressoVIP continua o mesmo do Ingresso simples. Objetos da classe IngressoVIP devem ser também capazes de informar o valor total do ingresso e o valor do imposto (em reais) aplicado ao ingresso.

Questão 14. Uma fábrica fabrica peças Azuis, Vermelhas e Amarelas. Cada tipo de peça (dado pela cor) tem o seu tempo de fabricação calculado de forma diferente. As peças azuis têm seu tempo de fabricação determinada por sua altura, sendo consumido 15s a cada centímetro de altura da peça. As peças vermelhas têm seu tempo de fabricação determinado pela largura, sendo consumido 7s a cada centímetro de largura da peça. No caso das peças amarelas, o tempo de fabricação é dado pelo produto de sua altura e sua largura em segundos.

Aplicando seus conhecimentos sobre herança em POO, implemente um conjunto de classes e um programa de testes que permita manter uma lista única de peças (independente da cor), onde seja possível determinar o tempo total de produção, considerando todos os elementos da lista.