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

Avaliação I
05/09/2017

Instruções Gerais

Leia atentamente as instruções a seguir antes de iniciar a avaliação:

1. Não esqueça de colocar o seu nome completo nesta folha.
2. Esta folha de questões, devidamente identificada, deve ser entregue ao professor ao final da avaliação.
3. A resolução desta avaliação é **individual**, porém está permitida a consulta a qualquer material de sua posse (implementações já realizadas por você, livros, notas de aula, dispositivos eletrônicos, etc.) e não compartilhado.
4. Você terá duas aulas (totalizando 1h40min) para responder a todas as questões. Por isso, gerencie bem o seu tempo.
5. Esta avaliação vale 6,00 (seis) pontos no total e o valor de cada questão é fornecido junto ao seu respectivo enunciado.
6. Leia atentamente o enunciado de cada questão antes de iniciar a sua resolução, as implementações devendo ser feitas na linguagem de programação C++.
7. Ao final da avaliação, envie um único arquivo compactado no formato ZIP com os arquivos de suas implementações para a tarefa correpondente no SIGAA.
8. É **obrigatório** o uso de Makefile e a correta organização em subdiretórios, seguindo o mesmo modelo utilizado nos Laboratórios.
9. Decréscimos de nota poderão ser aplicados caso sejam observadas as seguintes faltas:

Falta	Decréscimo
Programa apresenta erros de compilação, não executa ou apresenta saída incorreta	−70%
Legibilidade comprometida (falta de identação, etc.)	−10%
Implementação na linguagem de programação C ou resultante de mistura entre C e C++	−30%
Programa compila com mensagens de aviso (*warnings*)	−50%
Plagiarismo (no caso de cópia de colega, ambos serão penalizados)	−100%

Questão 1. (0,25) 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 2. (0,25) 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 3. (0,25) Para que servem membros estáticos de uma classe em C++? Justifique com um exemplo.

Questão 4. (0,25) 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 5. (0,50) 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 6. (0,50) 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 7. (4,0) 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 }
```

Para facilitar o seu trabalho e otimizar o tempo de implementação, resolvi disponibilizar estes arquivos na área da disciplina no SIGAA. Faça o download destes arquivos de lá! Espero que ajude!

Obrigado por sua ajuda, amigo! Nem sei como agradecer. –Teobaldo.