INF1301 – Programação Modular
Professor Flavio Bevilacqua
Período 2017.2

1. Sala e Horário

Segundas 17h-19h sala L522 Quartas 17h-19h sala L522
e-mail: bevilac@inf.puc-rio.br

2. Livro texto

Staa, A.v.; *Programação Modular*; Rio de Janeiro: Campus; 2000.

3. Critério de Aprovação

2 provas (todas **com consulta**)

P1: quarta 11/10 (17h às 19h) - Reposição 16/10 (17h às 19h)

P2: segunda, 11/12 (17h às 19h) - Reposição 13/12 (17h às 19h)

Obs: Se faltar a prova e a reposição, a nota da mesma fica zero.

4 trabalhos

T1: quarta 06/09

T2: quarta 04/10

T3: **quarta 01/11.**

T4: **segunda 04/12**

Cálculo do grau final

G1 = (P1 * 2 + T1 + T2 * 2) / 5.

G2 = (P2 * 2 + T3 + T4 * 2) / 5.

Se G2 >=3 então GrauFinal = (G1 + G2)/2

Senão GrauFinal = (G1 + 3*G2)/4

Para ser aprovado o aluno deverá alcançar um Grau Final igual ou maior do que 5.

IMPORTANTE: Nesta turma há cobrança de presença. Os alunos que ultrapassarem o limite de faltas permitido pela universidade (25% de faltas) estarão automaticamente reprovados por falta.

4. Sobre os Trabalhos

Algumas orientações:

- **serão aceitos somente programas redigidos em C. Não serão aceitos programas redigidos em C++, C# ou qualquer outra variação de C que não seja o "puro".**
- os trabalhos serão corrigidos em acordo com o documento *Critérios de Correção de Trabalhos* em anexo
- os programas podem ser desenvolvidos em qualquer plataforma, no entanto os programas entregues para a correção devem executar em máquinas utilizando o sistema operacional Windows 7.
- Os programas serão aplicações console, ou seja, devem operar em janelas CMD (DOS) do Windows.
- **Recomendamos especial cuidado com o teste final dos programas, assegurando que estes operem em máquina sem a infra-estrutura esperada pelo ambiente de desenvolvimento.**

Os trabalhos devem ser entregues via e-mail.

- caso a mensagem contenha vírus, a nota será 0 (zero). São utilizados diversos controladores de vírus.

Plano de aulas

	Datas	Assunto
1	14/08	Apresentação da disciplina
2	16/08	Introdução/Princípios de modularidade
3	21/08	Princípios de modularidade
4	23/08	Princípios de modularidade
5	28/08	Teste automatizado
6	30/08	Teste automatizado / Exercícios
7	04/09	Processo de Desenvolvimento em Engenharia de Software
8	06/09	Especificação de requisitos - Entrega Trab1
9	11/09	Modelagem
10	13/09	Exercícios
11	18/09	Assertivas
12	20/09	Implementação da programação modular
13	25/09	Implementação da programação modular
14	27/09	Exercícios
15	02/10	Estrutura de funções
16	04/10	Estrutura de funções - Entrega Trab2
17	09/10	Exercícios e revisão para a prova
18	11/10	Prova 1
19	16/10	Decomposição sucessiva - Reposição P1
20	18/10	Argumentação da corretude
21	23/10	Argumentação da corretude
22	25/10	Exercícios
23	30/10	Instrumentação
24	01/11	Instrumentação - Entrega Trab3
25	06/11	Instrumentação
26	08/11	Exercícios
27	13/11	Teste de módulos
28	22/11	Teste de módulos
29	27/11	Teste de módulos
30	29/11	Teste de módulos
31	04/12	Qualidade de software - Exercícios Finais - Entrega Trab4
32	06/12	Exercícios e revisão para a prova
33	11/12	Prova 2
34	13/12	Reposição de Prova 2
35	18/12	Entrega de Resultados Finais
36	20/12	

Critérios de correção dos trabalhos

Objetivos dos trabalhos

O objetivo da disciplina é aprender a ler, compreender, projetar e redigir programas modulares de boa qualidade, obedecendo a um conjunto de padrões. Este aprendizado é adquirido e demonstrado através da realização de uma série de trabalhos ***interdependentes***.

O objetivo dos trabalhos não é escrever algum programa, mas, sim, é desenvolver programas *modulares de boa qualidade* e que *comprovadamente* satisfaçam massas de teste previamente estabelecidas. Também não é objetivo verificar se o aluno conhece todas as sutilezas da linguagem de programação, ou dos algoritmos empregados, no entanto é claro que se espera um domínio razoável da linguagem e dos algoritmos. Ou seja, independentemente da dificuldade, do esforço e do tempo gasto pelo aluno, o que vai ser examinado ao corrigir os trabalhos é a *qualidade* destes, independentemente do número de horas que o aluno possivelmente tenha levado para desenvolver os programas.

O que será corrigido é o que foi entregue. Ou seja, caso sejam entregues componentes obsoletos ou errados, ou sejam esquecidos componentes, o aluno perderá os pontos correspondentes. O objetivo deste critério é induzir os alunos a tomarem cuidado ao compor a versão final a ser entregue.

Entrega do trabalho

- Os trabalhos deverão ser entregues via e-mail. Caso a mensagem não satisfaça qualquer um dos itens a seguir: **-2 pontos**
  - O assunto da mensagem (*subject*) deve ser: ***IdDiscip*-Trab*nn*** em que:
    - *IdDiscip* é o código da disciplina, (INF1301)
    - *nn* é o número do trabalho,
  - Todos os arquivos que compõem o trabalho devem estar "zipados" em um único arquivo anexado à mensagem enviada.
  - O nome do arquivo .ZIP deve ser ***IdDiscip*-Trab*nn*.ZIP** conforme descrito e exemplificado acima.
  - Para evitar a disseminação de vírus e outros tipos de malware, alguns provedores não permitem a inclusão de arquivos .exe nos arquivos .zip. neste caso adicione o sufixo .txt. Exemplo, se o arquivo era **Trab01.exe**, rebatize-o para **Trab01.exe.txt**
  - A codificação do arquivo anexado ao e-mail deve ser MIME.
- Para cada dia útil de atraso (domingos e feriados não são dias úteis, porém sábados e dias enforcados são dias úteis) é descontado: **1 ponto**. Obs. O término de um dia é às 7 horas do dia a seguir. Por exemplo, se o trabalho era devido no dia 7 e for enviado no dia 8 antes das 7 horas, não perde ponto, após as 7 horas perde.

Composição do trabalho

- Conteúdo da mensagem com vírus resulta em nota **zero**
  - Cuidado com as máquinas dos laboratórios, pois, por mais controle que se exerça, colegas ingênuos ou pouco éticos freqüentemente infectam estas máquinas.
- São exigidos tipicamente os arquivos a seguir. O enunciado do trabalho pode exigir arquivos e documentos complementares.
  - Programa executável
  - Módulos implementação fonte
  - Módulos definição fonte
  - Arquivo RELATORid.TXT – um arquivo por membro do grupo, identificado por *id* registro de trabalho realizado
  - Arquivo LEIAME.TXT explicando com se utiliza o programa.
  - Arquivos contendo diretivas (*scripts*) de teste
  - Arquivos adicionais constantes do enunciado do trabalho
- Se o conteúdo do arquivo ZIP estiver incompleto **-2 pontos**

Execução básica

- Se o leia.me não corresponde ao comportamento do programa: **-2 pontos**
- Se o programa não corresponde ao enunciado: **-8 pontos**.
- Se o programa não corresponde a uma implementação modular: **-8 pontos**.
- Se o programa for plágio de algum outro a nota do trabalho será **Zero**.
- Se o programa .exe não existe ou não dá partida: **-8 pontos**
  OBS: Recomendamos especial cuidado com o teste final, assegurando que o programa opere em máquina sem a infra-estrutura necessária para o ambiente de desenvolvimento. SUGESTÃO: Enviar ao professor um executável qualquer criado no ambiente de um dos componentes do grupo para ver se o mesmo solicitará DLLs extras. O grupo tem que ter o cuidado de sempre gerar a versão para entrega nesta máquina que linkou o executável sem DLLs extras. É de responsabilidade do próprio grupo descobrir como retirar estas dependências da aplicação. Caso o grupo não consiga retirar estas dependências, deverá gerar um passo a passo e enviar as DLLs necessárias e este manual deverá ser suficiente para que a aplicação com a dependência rode na máquina do professor.
- Se o programa não completa a execução (entra em *loop*, trava, cancela a execução, voa, etc.) **em todos** os testes usados: **-6 pontos**
- Se o programa não completa a execução em **um ou mais** dos casos de teste usados: **-4 pontos**. OBS. o teste será repetido para que se tenha certeza de não se tratar de uma falha fortuita do instrutor e na ficha de avaliação será descrito, em linhas gerais, como proceder para gerar a falha.
- O programa não executa o *script* de teste fornecido pelos instrutores: **-3 pontos**. OBS. Pode ocorrer que o *script* de teste fornecido esteja incorreto! Neste caso o grupo deve produzir um documento (.txt ou .doc) explicando o erro e incluir o arquivo *script* devidamente corrigido. O objetivo é aprender a conviver com especificações incompletas, incorretas ou inconsistentes, como é comum no "mundo real".
- Não foram incluídos *scripts* de teste produzidos pelo grupo: **-2 pontos**.
- Os *scripts* de teste são superficiais ou mal organizados: **-1 ponto**.
- O programa termina com a mensagem *Null pointer assignment*: **-1 ponto**.
- O programa não utiliza o arcabouço (*framework*) de teste fornecido: **-3 pontos**. O objetivo é aprender a utilizar componentes fornecidos por terceiros.
- Outros problemas encontrados: **-1 ponto por problema**

Código fonte e documentação

Estude os padrões contidos nos apêndices do livro!

- Se algum dos arquivos não identificar os autores: **-2 pontos**
- Se os módulos fonte não correspondem ao programa executável: **-8 pontos**
- Se outros arquivos não corresponderem ao programa implementado:.**-4 pontos**
- O programa fonte não existe ou não foi escrito em C: a nota do trabalho será **zero**
- Comentários inexistentes ou fora dos padrões: **-1 ponto**
- Nomes dos elementos fora dos padrões **-2 pontos**
- Constantes fora dos padrões, ex.: programa utiliza literais e "números mágicos" ao invés de constantes simbólicas **-2 pontos**
- Módulo de definição ou módulo de implementação fora dos padrões, ex.: o módulo de implementação contém a declaração ou definição de uma ou mais variáveis globais externas; o módulo de definição não pode ser utilizado tanto para compilar o módulo servidor cuja interface define, nem para compilar os clientes desse módulo: **-2 pontos**
- Estilo do código fora dos padrões **-2 pontos**. OBS. caso você tenha um padrão de estilo *documentado (impresso)* e revisto por terceiros (ex. padrão adotado em alguma empresa) você poderá utilizá-lo, desde que entregue uma cópia do documento ao instrutor.
- Especificações de funções não existem **-2 pontos**
- Especificações de funções estão incompletas **-1 ponto**
- Especificações de funções são inconsistentes com a implementação do elemento especificado:
  -2 pontos

- Especificações de funções mal escritas (português incorreto, ambíguo, confuso): **-1 ponto**
- Programa mal organizado **-1 ponto**
- Código duplicado, mesmo que ligeiramente alterado: **-2 pontos**
- Interfaces entre módulos, classes e funções inutilmente complexas: **-1 ponto**
- Código longo sem conter pseudo-operações: **-1 ponto**
- Código usa funções que retornam condições de funcionamento e que não são verificadas pelo programa (ex.: não verifica se malloc ou fopen retornaram NULL): **-1 ponto**
- Usualmente o enunciado contém critérios complementares! Cada um deles identificará o número de pontos perdidos caso não seja satisfeito.

Observação

Evidentemente o total de pontos que um aluno pode perder segundo os critérios acima é maior do que 10. Entretanto, a nota de cada trabalho é independente das notas e dos pontos perdidos em outros trabalhos. Porém, como os trabalhos são interdependentes, um primeiro trabalho mal feito pode prejudicar todos os outros. No mínimo acarretará um esforço muito maior nos trabalhos subseqüentes. Este esforço é necessário para corrigir os problemas identificados nos trabalhos anteriores.

Não será considerada como dúvida do grupo a retirada de erros de compilação dos programas. O grupo tem total responsabilidade em pesquisar por si só a solução para estes problemas em todos os meios possíveis, uma vez que é pré-requisito da disciplina ter conhecimento de linguagem C. Esta autonomia para ultrapassar obstáculos deste tipo faz parte do objetivo da disciplina e é uma característica fundamental para quem trabalha com desenvolvimento e implementação.

É permitido qualquer tipo de consulta nas provas. Porém, não será tolerada conversa durante a prova e se pelo menos uma questão tiver ao menos um trecho exatamente igual em duas provas, as duas receberão nota zero, caracterizando cola. Portanto, cuidado com sua prova!

Tenham todos um bom trabalho !