INF 1301 Programação Modular

INF 1628 Programação em Ponto Grande

Turmas

INF1301 turma 3VA e INF1628 turma 3WC

2as 21:00 às 23:00 sala L506; 4as 21:00 às 23:00 sala LAQ7

Prof. Flavio Bevilacqua

e-mail: bevilac@inf.puc-rio.br

INF1301 turma 3VB e INF1628 turma 3WB

2as e 4as 17:00 às 19:00 sala LAQ5A

Prof. Flavio Bevilacqua

e-mail: bevilac@inf.puc-rio.br

INF1628 turma 3WA

3as e 5as 9:00 às 11:00 sala L422

Prof. Arndt von Staa

e-mail: arndt@inf.puc-rio.br

Página das duas disciplinas: `HTTP://www.inf.puc-rio.br/~INF1301`

Atendimento:

Arndt: e-mail: `arndt@inf.puc-rio.br`

Flavio: e-mail: bevilac`@inf.puc-rio.br`

- Procure usar e-mail. Muitas das dúvidas podem ser tiradas através de trocas de e-mail. De maneira geral as repostas serão dadas no mesmo dia, entretanto, em alguns casos (por exemplo: durante os fins de semana), podem levar mais de um dia. Mesmo assim, o e-mail acaba sendo mais rápido do que uma entrevista presencial.

Observações:

1. As duas disciplinas são idênticas, INF1301 Programação Modular (PMOD) faz parte do programa de Bacharelado em Informática e INF1628 Programação em Ponto Grande (PPG) faz parte do programa de Engenharia de Computação.
2. Foram abertas vagas tanto para alunos de Bacharelado como de Engenharia de Computação em todas as turmas. Para evitar confusão, **PPG** designa as turmas de *INF1301 Programação Modular* e *INF1628 Programação em Ponto Grande* com aulas nas terças e quintas das 9 às 11 h. Já **PMod** designa as demais turmas de *INF1301 Programação Modular* e *INF1628 Programação em Ponto Grande* com aulas a partir das 17h.

Apresentação da disciplina

1. Ementa

Módulos, Interfaces, Acoplamento. Construção e uso de Bibliotecas; compilação independente e em separado. Tipos Abstratos de Dados; independência entre especificação e implementação. Princípios de Orientação a Objetos; programação orientada a eventos, amarração dinâmica. Tratamento de exceções. Princípios de testes de programas.

2. Objetivo

Espera-se que, ao concluir a disciplina, o aluno saiba projetar, implementar, controlar a qualidade de **módulos** e integrá-los, produzindo, rotineiramente, e com elevada produtividade, programas modulares de qualidade assegurada. Espera-se que o aluno desenvolva módulos corretos por construção, fugindo da usual seqüência de múltiplas tentativas e erros. Espera-se, ainda, que o aluno entenda como trabalhar em equipe. Finalmente, espera-se que o aluno entenda como organizar os programas e coordenar o seu desenvolvimento em um contexto em que ocorram alterações, tanto as devidas a evoluções do programa, como as decorrentes de especificações incorretas ou ambíguas.

3. Descrição

Programação modular é a base para se desenvolver programas de porte médio a muito grande. Através de um particionamento judicioso do programa em módulos possibilita-se o trabalho em equipe, facilita-se a gerência do desenvolvimento, facilita-se o controle e a garantia de qualidade, possibilita-se o reúso de módulos já desenvolvidos, viabiliza-se a criação de bibliotecas de componentes, e possibilita-se a redução do tempo necessário para dispor do programa.

Outra preocupação constante ao desenvolver programas é a garantia da qualidade. Para produzir programas de boa qualidade e que possam ter uma vida útil longa, é necessária uma atitude visando produzir módulos isentos de faltas. É fato conhecido que módulos desenvolvidos sem o devido cuidado dificilmente atingirão níveis de qualidade satisfatórios. Por outro lado, desenvolver módulos de boa qualidade usualmente acaba sendo bem mais barato do que desenvolvê-los sem preocupação com qualidade.

A programação modular apresenta diversos problemas. A seguir apresentamos uma lista que, embora longa, não pretende ser exaustiva:

- Quais são as diversas formas com que se apresentam módulos?
- Como particionar um programa em módulos?
- Como particionar um programa em módulos de modo que uma parcela significativa dos módulos possa ser efetiva e eficazmente reutilizada, em vários programas, sob a forma de componentes?
- Como desenvolver módulos reutilizáveis visando o estabelecimento de um conjunto de ativos (*assets*) que possam ser reutilizados em um grande número de projetos, constituindo, assim, a uma vantagem competitiva para a organização?
- Como explicitar e especificar a interface de cada um dos módulos?
- Como minimizar e simplificar interfaces de módulos?
- Como especificar cada um dos módulos?
- Como assegurar a qualidade de cada um dos módulos?

- Como assegurar a qualidade dos construtos[1] compostos a partir da integração de diversos módulos?
- Como viabilizar o desenvolvimento independente de cada módulo, possivelmente realizado por pessoas diferentes, talvez até desconhecidas umas das outras?
- Como coordenar o trabalho de desenvolvimento em equipe?
- Como organizar módulos de modo que sejam facilmente alterados, mesmo por pessoas que não tenham participado de seu desenvolvimento?
- Como manter a coerência entre os módulos à medida que forem ocorrendo alterações?
- Como manter a coerência de construtos, levando em conta a versão dos módulos que os compõem?
- Como manter a coerência dos módulos durante toda a sua vida útil, quando esta for longa (ex.: mais de 5 anos).

Nesta disciplina serão focalizadas técnicas de projeto, implementação e de teste de módulos e técnicas de integração de módulos formando programas. Serão abordadas, ainda, padrões e critérios de qualidade, visando a construção de módulos bem documentados e corretos por construção. Finalmente, será utilizado um processo de desenvolvimento, visando disciplinar as atividades realizadas ao desenvolver programas.

Foge ao escopo desta disciplina estudar os problemas relacionados com a análise de sistemas e a arquitetura de programas. Embora sejam estudados os princípios básicos de orientação a objetos, foge ao escopo da disciplina estudar técnicas de análise e projeto de sistemas implementados utilizando orientação a objetos. Estes assuntos, embora fortemente relacionados com programação modular, extrapolam o limite desejável para uma disciplina com um semestre de duração. São tipicamente estudados em disciplinas tais como *Especificação de Sistemas* e *Projeto de Sistemas de Software*. Para facilitar a integração com disciplinas específicas, procuramos dar indicações de como a programação modular interage com os problemas de arquitetura de programas e de projeto orientado a objetos.

3.1. Tópicos

A disciplina é particionada em quatro grandes seções, a saber:

- Processo de desenvolvimento.
- Tecnologia de apoio à modularidade.
- Técnicas visando o desenvolvimento de programas modulares.
- Teste e composição de programas.

Na seção *Processo de desenvolvimento* introduzimos a noção de processo, de organização do trabalho e de qualidade de software. A seção tem por objetivo familiarizar o aluno com técnicas de prevenção de faltas através do uso de padrões. Estas técnicas são bastante eficazes, sendo fundamentais ao produzir artefatos que não podem ser adequadamente testados. Nesta seção queremos conscientizar o aluno que qualidade é muito mais um problema de *atitude* a ser observada ao longo de todo o desenvolvimento, ao invés de ser o resultado da aplicação impensada de um conjunto de técnicas. Os tópicos desta seção são distribuídos sobre o conjunto de aulas, ou seja, não é formado por um bloco de aulas identificável isoladamente. É formado por conceitos distribuído sobre diferentes aulas.

Na seção *Tecnologia de apoio à modularidade* introduzimos os conceitos básicos envolvendo programas modulares. São vistos tanto o paradigma da estrutura de funções como, de forma bem resumida, o paradigma orientado a objetos. Esperamos que, ao final dessa seção, o aluno domine os conceitos básicos e operacionais do desenvolvimento de programas modulares, nos quais módulos implementam noções (tipos abstratos de dados, classes, funções) bem definidas. O aluno deverá ser capaz de compreender as especificações do conjunto de módulos que compõem um programa, produzir estes módulos,

1 Construto é uma implementação parcial de um programa, permitindo avaliar e, possivelmente, utilizá-lo de forma produtiva.

compilá-los e compor o programa com sucesso, mesmo em um contexto em que as especificações evoluam no tempo. Tópicos:

- *Princípios de modularidade* discute os conceitos relativos a módulos e interfaces entre módulos. É dada ênfase especial às interfaces, pois estas são os pontos de toque de qualquer projeto modular bem sucedido. É brevemente abordado o papel e a necessidade da arquitetura de programas.
- *Teste automatizado de módulos* discute os conceitos de teste automatizado e apresenta um arcabouço de apoio ao teste automatizado de módulos redigidos em C. O teste automatizado permite a repetição dos testes e auxilia no desenvolvimento incremental de módulos e programas.
- *Implementação da programação modular* mostra como implementar interfaces bem definidas nas diferentes linguagens de programação. É mostrado como transferir para o processador de linguagem o máximo possível do controle da integração dos módulos. Apesar do ponto de vista utilizado ser específico para a linguagem, no caso C, os conceitos aplicam-se a uma variedade de linguagens, entre elas C++, C# e Java.
- *Ferramentas de apoio ao desenvolvimento* apresenta as ferramentas básicas utilizadas em ambientes de engenharia de software, em particular em ambientes visando o desenvolvimento de programas modulares. As ferramentas são apresentadas junto com a descrição de um processo físico em que são utilizadas. O objetivo é familiarizar o aluno com a idéia de um processo definido e apoiado em ferramentas. Julgamos ser necessário programadores conhecerem estas ferramentas, mesmo sabendo que costumam estar embutidas nos ambientes de programação disponíveis no mercado. Entre as ferramentas encontram-se algumas visando a automação dos testes e o apoio ao desenvolvimento incremental.
- *Estrutura de funções* examina organizações de programas baseadas em funções. Além de estruturas de funções propriamente são discutidos o particionamento de algoritmos extensos em estruturas de funções, o tratamento de exceções no contexto de estruturas de funções, e as técnicas de projeto de algoritmos baseados em esquemas de algoritmos contendo pontos a serem instanciados. Algoritmos com estas características permitem desenvolver módulos corretos que independem das estruturas de dados efetivamente manipuladas. Embora o foco seja o de estruturas de funções, procuramos sempre utilizar, de forma subjacente, o paradigma orientado a objetos ao projetar os módulos compostos por funções.

Na seção *Técnicas visando programas modulares* apresentamos técnicas de especificação, projeto e codificação de programas modulares tendo por objetivo produzir módulos de qualidade satisfatória por construção. Nesta seção seguimos a forma de desenvolvimento tradicional do mais geral para o mais detalhado.

- *Especificação de módulos* discute o que vêm a ser boas especificações de módulos e dos seus componentes. São apresentadas regras e recomendações a serem observadas pelas especificações e pela documentação dos módulos. Espera-se que ao final deste capítulo, o aluno não só esteja convencido da necessidade de se produzir e documentar especificações, como também saiba como produzir uma boa especificação ou transformar uma especificação de qualidade duvidosa em uma boa especificação. São utilizadas técnicas de especificação baseadas em linguagem natural.
- *Qualidade estrutural* apresenta regras e recomendações relativas a estruturas em geral, e estruturas de projeto de módulos em particular. Tais estruturas aparecem sempre que se desenvolve um artefato composto. Os conceitos introduzidos neste capítulo podem ser adaptados a documentos, auxílios (*help*) e tutoriais. Os elementos usados ao desenvolver programas são tipicamente estruturas, tais como estruturas de módulos, estruturas de classes, estruturas de funções, estruturas de dados. Estruturas aparecem como conseqüência de passos de projeto que transformam uma especificação em uma solução composta. As regras e recomendações procuram induzir uma atitude de contínua avaliação dos passos de projeto e das especificações de interface dos elementos criados durante os passos de projeto. Esta atitude previne uma grande gama de faltas e deficiências cuja remoção tende a ser cara e demorada quando realizada tardiamente.
- *Introdução à argumentação da corretude* apresenta algumas das técnicas utilizadas ao provar a corretude de algoritmos e estruturas de dados. O objetivo é habilitar o aluno a redigir pré e pós-condições (assertivas) usando uma abordagem formal. As técnicas são apresentadas sob a ótica

de um processo construtivo a ser praticado passo a passo ao projetar a composição interna de funções, classes, estruturas de dados complexas e módulos. Utilizando as técnicas apresentadas, o aluno deverá ser capaz de argumentar a corretude de algoritmos, formular casos teste e localizar, com precisão, as faltas existentes no programa. Habilitando o aluno a utilizar um raciocínio lógico, espera-se que sejam prevenidos erros de programação e que seja facilitada a localização e remoção completa das faltas remanescentes nos diversos módulos que constituem um programa.

Na seção *Testes e composição de programas* discutimos assuntos relacionados com teste de módulos e de integração de módulos formando programas completos. São examinadas as técnicas de teste mais elementares, sendo dada ênfase em testes de módulos que implementam estruturas de dados. Apresentamos técnicas de instrumentação com o intuito de criar programas robustos e de transferir para o computador uma parte substantiva do esforço de teste e de avaliação dos resultados dos testes. Já no decorrer do curso são utilizadas extensivamente técnicas de teste automatizado. O teste de módulos está intimamente relacionado com a integração de módulos, formando sucessivos construtos cada qual mais abrangente que o anterior até que se disponha do programa completo. A composição de módulos deve levar em conta a versão de cada componente de modo que se assegure consistência entre todos eles. Tópicos:

- *Instrumentação de módulos* apresenta técnicas para a incorporação de instrumentos em módulos. Alguns destes instrumentos têm por objetivo automatizar a realização de testes. Outros têm por objetivo analisar a cobertura dos testes. Finalmente, outros têm o propósito de tornar robustos os programas. Esta última classe de instrumentos baseia-se na inclusão de assertivas executáveis nos módulos, integrando, assim, conceitos de argumentação da corretude de módulos com técnicas de teste de módulos.
- *Teste de módulos* discute as técnicas mais elementares de teste de módulos. Descrevemos critérios de seleção de casos de teste que resultem em conjuntos pequenos e eficazes de casos de teste, e que serão utilizados para testar módulos e, mais especificamente, estruturas de dados.
- *Integração de programas* apresenta técnicas para o desenvolvimento incremental de programas, criando sucessivos construtos, cada qual implementando uma parte maior da funcionalidade do programa.

3.2. Livro texto

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

3.3. Referências bibliográficas complementares

Gries, D.; *The Science of Programming*; New York: Springer; 1981

Souza, J.N.; *Lógica para Ciência da Computação*; Rio de Janeiro: Campus; 2002

Algum livro que trate da linguagem de programação C, exemplos:

Schildt, H.; *C Completo e Total*; 3a. edição; Makron; 1996

Kelley, A.; Pohl, I.: *Programming in C*; Fourth Edition; Addison Wesley Longman; 1998

4. Critério de aprovação

As disciplinas *INF 1301 Programação Modular* e *INF 1628 Programação em Ponto Grande* adotam o critério da categoria III da resolução 1/2005, copiada a seguir:

Categoria III - A avaliação do aproveitamento feita pelo professor será expressa por meio de dois graus de qualificação, apresentados numericamente, em escala de zero (0) a dez (10), do seguinte modo:

a) o primeiro grau de qualificação, representando o aproveitamento de aluno na disciplina, será obtido através de testes, relatórios, trabalho ou uma única prova realizada no meio do período letivo ou de trabalho equivalente, tendo em vista um programa parcialmente lecionado;

b) o segundo grau de qualificação, resultante de testes, relatórios, trabalho, prova ou projeto, cobrindo um programa parcialmente lecionado ou toda a matéria lecionada no período letivo. Neste grau podem ser incluídos testes e relatórios relativos a programa parcialmente lecionado;

c) o Grau Final será calculado conforme um dos dois casos a seguir:

1. Se o segundo grau for maior ou igual a três (3,0), o **Grau Final** será a média aritmética das duas avaliações (mesmo peso).
2. Se o segundo grau for menor que três (3,0), o **Grau Final** será calculado tendo o primeiro grau peso um (1) e o segundo grau peso três (3).

Para o cálculo desses graus serão realizados:

2 provas (todas **com consulta**)

1. **P1: sábado 28/04**, local e horário a ser determinado. A prova incluirá a matéria até a aula 15 Conceitos básicos de OO, inclusive.
2. **P2: sábado, 16/06**, local e horário a ser determinado

> OBS. Todas as turmas fazem as provas nos mesmos dias e horários. As datas, horários e salas das provas poderão variar com relação ao planejado para evitar coincidência com as datas e horários de provas de outras disciplinas. Mantenha-se informado consultando com freqüência a página *Avisos gerais* da disciplina.

Cabe salientar que não comparecer a uma prova resulta na nota 0.

4 trabalhos, prazos

1. **T1**: **quinta 29/03**: revisão, teste e correção de um programa composto por módulos fornecidos.
2. **T2**: **quinta 26/04**: implementação de módulos de apoio ou que manipulam uma estrutura de dados
3. **T3**: **quinta 22/05**: modificação e generalização dos módulos do 2o. trabalho e desenvolvimento de novos módulos visando construir uma aplicação usando esse conjunto
4. **T4**: **quinta 14/06**: modificação dos módulos do 3o. trabalho, incluindo instrumentação e realização do teste dessa instrumentação

4.1. Cálculo do grau final

A seguir é apresentada a forma de cálculo dos dois graus e do grau final:

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

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

GrauFinal = **if** (G2 >= 3.) **then** (G1 + G2) / 2. **else** (G1 + 3. * G2) / 4. **fi** [2]

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

OBS. A data de cancelamento de disciplinas é: 31/05, o grau G1 estará disponível antes desta data.

4.2. Trabalhos

Os trabalhos serão feitos

- em grupos de 2 ou 3 alunos
- **serão aceitos somente programas redigidos em C.**
- 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 XT.

[2] Notação Algol 68.

- Os programas devem ser compiláveis utilizando os compiladores MS Visual C/C++ 6.0 ou MS Visual C/C++ dot Net[3].
- 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.**
- Para cada um dos módulos de produção deverá ser desenvolvido um módulo de interpretador de comandos de teste que permita automatizar o respectivo teste.

Os trabalhos devem ser entregues via e-mail.

- caso a mensagem contenha vírus, a nota será 0 (zero). São utilizados diversos controladores de vírus.
- será descontado 1 ponto por dia útil de atraso (domingos e feriados não são dias úteis, **sábado e dias enforcados são dias úteis**). O término de um "dia" é às 6 horas da manhã do dia a seguir. Por exemplo, se o trabalho for devido no dia 7 e for enviado no dia 8 antes das 6 horas, não perde ponto, após às 6 horas perde.
- ***leia com atenção o folheto de critérios de avaliação dos trabalhos em anexo.***

[3] O Departamento de Informática tem um licença de uso de software da Microsoft que permite a distribuição de cópias para alunos matriculados em disciplinas do DI. Veja no LabGrad como conseguir as cópias.

5. Plano de aulas

	Datas PPG	Assunto
		O plano de aulas é baseado na turma de terças e quintas (PPG). Para manter o sincronismo das aulas nas diversas turmas poderão ser necessárias algumas mudanças no calendário das outras turmas.
1	27/02	Apresentação da disciplina
2	01/03	Teste automatizado. Padrões de programação
3	06/03	Teste automatizado. Padrões de programação
4	08/03	Conceito de modularidade. Interfaces
5	13/03	Conceito de modularidade. Encapsulamento, Acoplamento
6	15/03	Conceito de modularidade. Encapsulamento, Acoplamento
7	20/03	Modelagem, modelo conceitual e físico, exemplo abstrato e concreto
8	22/03	Especificação de requisitos de módulos e de funções
9	27/03	Assertivas, pré-condições, pós-condições
10	29/03	Implementação dos conceitos de modularidade, tipagem
11	03/04	Implementação dos conceitos de modularidade, declaração definição
12	10/04	Ferramentas de apoio: *Make*, *lib*
13	12/04	Estrutura de funções
14	17/04	Estrutura de funções
15	19/04	Conceitos básicos de OO, classes, construtores, herança
16	24/04	Decomposição sucessiva
17	26/04	Assertivas, argumentação da corretude
	28/04	SÁBADO! 1a. prova horário e sala a confirmar
18	03/05	Assertivas, argumentação da corretude
19	08/05	Assertivas, argumentação da corretude
20	10/05	Assertivas estruturais: argumentação da corretude de estruturas de dados
21	15/05	Instrumentação
22	17/05	Instrumentação
23	22/05	Instrumentação
24	24/05	Métodos básicos de teste de módulos
25	29/05	Métodos de teste de módulos: cobertura de estrutura de código
26	31/05	Métodos de teste de módulos: cobertura de estruturas de dados
27	05/06	Métodos de teste de módulos: cobertura de estruturas de dados
28	12/06	Princípios de qualidade de software
	16/06	SÁBADO! 2ª. Prova horário e sala a confirmar

As datas das aulas são relativas às turmas de 3as e 5as (PPG). As datas das aulas das turmas de 2as e 4as, e das de 4as e 6as devem ser devidamente ajustadas (PMOD).

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. 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 tenha levado para desenvolver os programas.

De maneira geral os trabalhos são bastante trabalhosos e sua realização deve ser iniciada imediatamente ao receber o enunciado. Os enunciados deixarão margens para dúvidas. Sempre consulte o instrutor para tirá-las. O objetivo disto é simular o "mundo real" encontrado ao desenvolver programas em empresas.

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-idGrupo*** em que:
    - *IdDiscip* é o código da disciplina, ex. INF1628 ou INF1301
    - *nn* é o número do trabalho,
    - *idGrupo* é formado por *n* conjuntos de duas ou três letras, cada conjunto identificando um dos *n* membros do grupo.
      - Exemplo: INF1301-Trab1-JBOGESP
    - OBS.: Num espaço de 48 horas será enviada uma resposta a quem enviou o trabalho, confirmando se o trabalho foi recebido corretamente.
  - O texto da mensagem deve identificar cada um dos os autores fornecendo os respectivos: número de matrícula, nome e endereço e-mail.
  - 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-idGrupo*** conforme descrito e exemplificado acima.
  - 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 6 horas do dia a seguir. Por exemplo, se o trabalho era devido no dia 7 e for enviado no dia 8 antes das 6 horas, não perde ponto, após as 6 horas perde.

Composição do trabalho

- "Grupos" formados por um único ou por quatro ou mais alunos: **-2 pontos**.
  - Procure formar o seu grupo já na primeira aula. Leve em conta a afinidade e a disponibilidade de tempo de cada participante. O objetivo é aprender a organizar e realizar trabalho em grupo.
- 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 complementares.

  - Programa executável
  - Módulos implementação fonte
  - Módulos definição fonte
  - Arquivo RELATORid.TXT – um arquivo por membro do grupo, identificado por *id*. O arquivo conterá o registro de tempos do trabalho realizado pelo aluno.
  - 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 `.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.
- Se o programa não completa a execução (entra em *loop*, trava, cancela a execução, voa, etc.) **em todos** os testes usados: **-8 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**.
- 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 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 os módulos fonte não corresponderem ao programa executável entregue: **-8 pontos**
- Se outros arquivos não corresponderem ao programa implementado:.**-6 pontos**
- O programa não utiliza o arcabouço (*framework*) de teste fornecido: **-3 pontos**. O objetivo é aprender a utilizar componentes fornecidos por terceiros.
- 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 e de funções são inconsistentes com a implementação do elemento especificado: **-2 pontos**
- Especificações e 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 deixem os trabalhos para o último dia! Eles dão muito trabalho!