Natã Goulart da Silva

Uma metodologia para o desenvolvimento de sistemas de correção automática de listas de exercícios de programação baseada em testes unitários

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Joaquim Quinteiro Uchôa

Co-Orientador
Prof. Thiago Ferreira de Noronha

Lavras
Minas Gerais - Brasil
2010

Natã Goulart da Silva

Uma metodologia para o desenvolvimento de sistemas de correção automática de listas de exercícios de programação baseada em testes unitários

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em *24 de abril de 2010*

Prof. Samuel Pereira Dias

Prof. Marluce Rodrigues Pereira

Prof. Joaquim Quinteiro Uchôa
(Orientador)

Prof. Thiago Ferreira de Noronha
(Co-Orientador)

Lavras
Minas Gerais - Brasil
2010

Agradecimentos

Agradeço a minha mãe, meus irmãos, toda a minha família, amigos, aos professores do curso ARL pelo aprendizado, aos professores Joaquim Uchôa e Thiago Noronha pelas valiosas contribuições e principalmente a minha noiva, Roseane pelo apoio que recebi em todas as horas.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho propõe a utilização de testes unitários que permitem a correção de listas de exercícios de linguagem de programação de forma automática. Utilizando os testes unitários, os alunos codificaram as práticas essenciais para o aprendizado e domínio das técnicas de programação, verificando se a sua codificação resolve corretamente o problema proposto. É objetivo também deste trabalho, disponibilizar um conjunto de listas de exercícios, codificadas com os testes unitários sob uma licença de *software* livre que possam ser usadas por professores, estudantes e outros interessados.

Palavras-Chave: *Software* Livre; Testes Unitários; Juízes Online; Ensino Linguagem Programação.

Capítulo 1

Introdução

Neste capítulo são apresentadas a descrição e as justificativas para o desenvolvimento de uma metodologia de correção automática de listas de exercícios de linguagem de programação. Através de um conjunto de testes unitários é possível construir listas que são avaliadas pelos próprios alunos. Estes testes permitem que os alunos tenham o código avaliado sem a intervenção dos tutores e professores. Além da correção automática, outras boas práticas de programação como utilização de testes e o uso de estilos na construção do código são abordadas.

1.1 Contextualização

Com a adoção de várias universidades federais ao programa de Reestruturação e Expansão das Universidades Federais - REUNI, espera-se um grande aumento no número de alunos por turma nas disciplinas de ensino de programação de computadores. A estimativa é que o número médio de alunos em disciplinas como Algoritmos e Estrutura de Dados I e II do DCC da Universidade Federal de Minas Gerais - UFMG, seja de aproximadamente 100 alunos por turma. Estas disciplinas são oferecidas em diversos cursos de graduação em instituições de todo o país. Na UFMG, vários cursos oferecem estas disciplinas em sua grade, dentre os quais podemos citar: Ciência da Computação, Engenharia de Controle e Automação, Engenharia da Produção, Engenharia Elétrica, Estatística, Matemática Computacional e Sistemas de Informação.

O ensino de linguagens de programação de computadores, em todos os seus níveis, exige dos alunos a resolução de um grande número de exercícios, a fim

de que eles desenvolvam o raciocínio lógico sequencial necessário para solucionar problemas computacionais. A correção destes exercícios demanda muito tempo dos professores e dos monitores das disciplinas. Com o crescimento do número de alunos nas disciplinas de ensino de programação, a correção manual de todos exercícios se tornará impraticável.

A abordagem mais empregada atualmente para correção automática de exercícios de programação é baseada nos chamados Juízes *Online*, do inglês *Online Judge* (CHEANG; KURNIA; OON, 2003). Nesta abordagem, o aluno escreve um programa de computador que soluciona um problema computacional e submete seu programa via Internet para um servidor de testes. Este servidor trata o programa do aluno como uma "caixa preta", aplicando-o em vários casos de teste. Se o resultado obtido for igual ao conjunto de resultado esperado, o juiz *online* atesta que o programa do aluno está correto.

Algumas desvantagens desta abordagem são: (i) uma vez que o programa do aluno não produz o resultado esperado, o juiz *online* não tem como indicar qual é o ponto do programa que está errado, nem sugerir uma possível solução; (ii) nenhuma pré-condição pode ser avaliada dentro do programa do aluno. Mesmo que o enunciado do exercício exija que o aluno utilize um determinado algoritmo ou estrutura de dados específica, isto não pode ser avaliado pelo juiz *online*; (iii) a manutenção do servidor de testes utiliza recursos financeiros e de pessoal técnico da universidade; (iv) se por alguma razão o servidor estiver congestionado ou inoperante, os alunos não poderão avaliar os seus exercícios. Surge então a necessidade de se estudar outros mecanismos alternativos ou complementares para correção automática de exercícios de programação.

1.2 Justificativas

As justificativas para a criação de um sistema de correção automática de exercícios são: (i) fornecer ao aluno não somente o parecer de que seu exercício está correto, mas ajudá-lo na resolução; (ii) apresentá-lo desde o início da sua formação às boas práticas de programação; (iii) redução de custos com equipamentos já que os testes unitários são compilados e executados junto com o programa escrito pelo aluno em sua máquina de trabalho, dispensando a necessidade de um servidor de testes.

Este projeto tem a colaboração do professor Thiago Noronha do departamento de Ciência da Computação da UFMG. A metodologia foi aplicada na disciplina de Algoritmos e Estrutura de Dados II - AEDS II, na qual o professor foi responsá-

vel. No primeiro semestre de 2010, através da aplicação da metodologia proposta, pode-se verificar a eficiência pedagógica da abordagem.

Para que esta proposta seja aplicada no ensino de disciplinas como AEDS II, inicialmente será desenvolvido um conjunto de problemas, código de testes e documentações com o licenciamento *GPL*. Estes problemas serão divididos por disciplinas, tópicos e níveis de dificuldade.

Acredita-se que o banco de questões e códigos disponíveis irá crescer ao longo do tempo através da colaboração de alunos e outros professores. O professor Thiago Noronha será inicialmente, o mantenedor das listas e avaliará as sugestões de novos problemas. Os alunos desde o início de sua graduação terão contato com a metodologia de desenvolvimento de *software* baseada em testes unitários, que é largamente utilizada no desenvolvimento de aplicativos por empresas e desta forma sairão mais preparados para o mercado de trabalho. Com a adoção desta metodologia, os professores e monitores poderão utilizar o tempo que foi economizado com a correção de exercícios, na resolução de dúvidas e na preparação das aula teóricas.

As listas de exercícios e os respectivos testes unitários desenvolvidos neste projeto poderão ser utilizados por várias disciplinas de programação em várias universidades em todo Brasil. Como a metodologia proposta não necessita de um servidor de teste (Juiz *Online*), mesmo as universidades com menos recursos poderão utilizar-se desta abordagem em suas disciplinas de programação.

1.3 Metodologia e Objetivos

Neste trabalho são utilizadas listas de exercícios com enunciados construídos de acordo com a ementa da disciplina. Serão disponibilizados para cada lista arquivos com: (i) as codificações das interfaces dos métodos utilizados para resolver as questões da lista; (ii) as bibliotecas do Google Test (TEST, 2010); (iii) os vários testes codificados para cada uma das questões da lista.

Inicialmente, não é ensinado aos alunos as técnicas de construção dos testes unitários para verificação das questões. Os alunos utilizam os testes unitários através de uma chamada de função presente no arquivo principal. Assim, a atenção dos alunos é voltada para a construção dos códigos que resolvem as questões e o uso dos testes unitários, através da observação dos logs com o resultado dos testes. Em um momento posterior, detalhes da construção dos testes podem ser abordados de acordo com a ementa da disciplina e o andamento do cronograma.

Na aula prática em laboratório, após a disponibilização da lista para os alunos, é informado que existe uma função que executa os testes sobre as questões da lista e que essa função pode ser usada. As listas contêm questões com níveis diferentes de dificuldade.

Os objetivos específicos do trabalho são: (i) construir um conjunto de listas de exercícios de linguagem de programação, que sejam automaticamente corrigidas por testes unitários; (ii) a criação de um projeto de código livre disponibilizando as listas na Internet.

1.4 Estrutura do Texto

O restante deste documento está estruturado da seguinte forma: O Capítulo 2 apresenta uma revisão bibliográfica sobre os sistemas Juízes *Online* utilizados para avaliar algoritmos codificados em determinada linguagem de programação. Ainda neste capítulo, há informações sobre dois *frameworks* utilizados na construção de testes unitários, padrões na construção de códigos e licenças de *software* livre. No Capítulo 3 são descritos detalhes sobre o *framework Google Test*. No Capítulo 4 há informações sobre as listas de exercícios utilizadas na disciplina AEDS II, a utilização dos testes unitários e os resultados obtidos com a aplicação das listas. O Capítulo 5 apresenta as observações finais sobre o trabalho e finalmente o Apêndice A traz o principal arquivo codificado de testes.

Capítulo 2

Trabalhos Relacionados

Este capítulo apresenta alguns sistemas e tecnologias que podem ser aplicadas no ensino de linguagens de programação e em outros cursos *online*. Serão apresentadas também características técnicas destes sistemas.

2.1 Juízes Online

Os sistemas chamados juízes *online* são capazes de receber um programa codificado em linguagens de programação como *C, C++*, entre outras, verificando se este programa apresenta para determinadas entradas pré-definidas, as saídas esperadas. Estes sistemas foram inicialmente concebidos para uso em concursos e competições de programação (IBM2010, 2010). Nos últimos anos porém, várias instituições têm utilizado e adaptado estes sistemas para o ensino de linguagens de programação em cursos técnicos e superiores ao longo do mundo (KURNIA; LIM; OON, 2001; KURNIA; MALAFIEJSK; NOINSK, 2008). Estes sistemas estão sendo integrados a sistemas de ensino e acompanhamento *online* chamados *Learning Management System - LMS* (WOO; HONG; KIM, 2008). Um dos sistemas LMS mais utilizado para o acompanhamento acadêmico e também em cursos a distância é o *Moodle* (MOODLE2010, 2010), que faz uso das amplas possibilidades da Internet para o ensino, seja presencial ou a distância.

Nos sistemas juízes *online* atuais, testes são realizados para cada código submetido baseados em vários conjuntos de dados de entrada e os resultados são disponibilizados para os alunos em poucos instantes, com avaliações pertinentes a cada teste realizado.

Atualmente, existem propostas de adicionar aos juízes *online* ferramentas com o objetivo de identificar a cópia dos trabalhos que são submetidos. A detecção desta cópia, conhecida na literatura como plágio, é uma função fundamental nos sistemas de avaliação utilizados no meio acadêmico. Dentre as opções de detecção de plágios disponíveis para uso, podemos citar um *plugin* desenvolvido para o *Moodle* baseado no sistema chamado *Measure Of Software Similarity - MOSS* (SCHLEIMER; WILKERSON; AIKEN, 2003). Este *plugin* tem o projeto de desenvolvimento disponível em (MOSS2010, 2010).

Existem sistemas para juízes *online* que permitem a submissão de códigos para várias linguagens de programação como *Java, C, C++, Pascal*, entre outras. Como os juízes *online* foram inicialmente concebidos para uso em competições, possuem para cada submissão um conjunto restrito de recursos computacionais sendo uma forma de avaliar a eficiência do código submetido. Estas restrições são também uma proteção para o ambiente onde os algoritmos são testados, de forma a evitar que a submissão de algum algoritmo atrapalhe o funcionamento correto do servidor. Nestes sistemas, um programa deve ser executado com um valor limitado de tempo de CPU e total de memória.

Dentre as aplicações destes sistemas encontradas na literatura, podemos citar o uso do sistema de correção automática que é utilizado na escola de computação da Universidade de Singapura, descrito em (CHEANG; KURNIA; OON, 2003). Nesta universidade, até o final da década de 90, os trabalhos acadêmicos na área de ensino de linguagem de programação eram corrigidos manualmente. Pode-se verificar que esta abordagem tem várias desvantagens dentre as quais podemos citar:

1. dificuldade na avaliação e correção de um grande número de programas;
2. muito tempo gasto na correção dos exercícios que poderia ser utilizado para outras atividades pedagógicas;
3. redução do número de exercícios aplicados e aumento da complexidade, devido a dificuldade de correção.

A partir do ano 2000, esta e outras instituições (KURNIA; LIM; OON, 2001) e (KURNIA; MALAFIEJSK; NOINSK, 2008) começaram a utilizar ferramentas de correção automática de problemas de linguagem de computação.

Um sistema de correção automática exige que as saídas geradas pelos programas sejam exibidas seguindo um padrão e formato próprio. Caso a saída não esteja corretamente formatada o sistema pode considerar que o programa não resolve o

problema adequadamente. Se por exemplo, o programa espera como saída dados tabulados em duas colunas, o resultado pode ser considerado errado se os dados apresentados forem separados por algum outro delimitador diferente de espaço ou gerado com apenas uma coluna. Por isto, para cada questão proposta pelo sistema, estão geralmente disponíveis para o aluno entradas e saídas que podem ser utilizadas como massa de teste, a fim de garantir que a formatação de saída seja obedecida.

Outro fator a ser analisado é o custo computacional das avaliações dos programas enviados pelos alunos. Alguns alunos podem usar a estratégia de submeter o seu programa sem que este esteja devidamente testado, utilizando o sistema de correção automática no lugar dos seus próprios testes. Para evitar que tais atitudes gerem uma grande carga ao sistema de avaliação, o número de submissões para cada aluno é limitado.

2.2 Testes Unitários

Testes unitários também chamados de testes de unidade, teste de módulos ou testes de componentes, avaliam a menor unidade lógica ou bloco de um programa. Em programação estruturada, por exemplo, esta unidade lógica pode ser definida como um procedimento ou função (DUSTIN; RASHKA; PAUL, 1999).

Os testes unitários fazem parte de um conjunto de testes chamados de testes de caixa branca (MISRA, 2003). Diferentemente dos testes de caixa preta, onde o sistema recebe uma entrada e após o processamento, gera uma saída, os testes de caixa branca permitem o conhecimento do funcionamento interno do sistema, com as suas funções e particularidades. Os testes unitários são utilizados para verificar se o *software* faz corretamente e adequadamente todos os detalhes especificados no projeto ou, no caso do uso acadêmico, no enunciado da questão. Como são testes que irão avaliar os detalhes da implementação, são codificados normalmente pelo próprio desenvolvedor que tem o conhecimento da lógica utilizada. Pode-se desenvolver os testes unitários antes da codificação do próprio módulo, durante o desenvolvimento ou poderá ocorrer uma iteração entre a codificação e os testes, a fim de verificar possíveis falhas. Em sistemas grandes e complexos formados pelas camadas de lógica, dados e interface com o usuário, os testes unitários devem verificar todos os possíveis caminhos do código, avaliar telas, menus, mensagens de usuário devidamente formatadas, validar campos e verificar se as exceções são devidamente codificadas.

Cada teste unitário em um *software* deve garantir que o algoritmo e a lógica estejam corretos e que os módulos ou funções do *software* atendam a todos os requisitos que lhe foram atribuídos. Uma boa estratégia é aliar testes de caixa branca e testes de caixa preta para um boa cobertura dos erros avaliados (BEYDEDA; GRUH; STACHORSKI, 2001).

No desenvolvimento de sistemas, quando possível, outro desenvolvedor deve executar os testes unitários que foram especificados pelo responsável do módulo. Normalmente, estes testes são executados e os resultados armazenados para a documentação do projeto, seja manualmente, seja com o auxílio de alguma ferramenta de suporte aos testes. Um modelo de controle dos testes unitários é a criação de uma lista de avaliação que pode ter como exemplo, os seguintes itens:

1. O código atende as especificações do projeto?
2. Para cada instrução condicional, a execução é realizada corretamente?
3. Os testes foram executados em todas as condições que a unidade de código deve avaliar?
4. Todas as exceções foram devidamente tratadas?
5. Toda a extensão do código foi verificada pelo menos uma vez?
6. Todas as condições de limites de avaliação foram realizadas?

Ainda, segundo (DUSTIN; RASHKA; PAUL, 1999), testes de caixa branca como os testes unitários geralmente utilizam cerca de 60% do esforço e custo dos testes em um sistema, por outro lado, pode garantir um aumento de cerca de 40% na qualidade do *software* desenvolvido. Devido ao alto custo gerado na elaboração dos testes unitários, é importante a utilização de algumas ferramentas e metodologias para o seu desenvolvimento e aplicação. Com o objetivo de melhorar a produtividade dos testes unitários foram criados vários *frameworks* que auxiliam no desenvolvimento e no controle dos testes.

2.2.1 Framework para Testes Unitários

Dentre os *frameworks* utilizados para a codificação dos testes unitários, são aqui apresentadas algumas informações sobre o *JUnit* (JUNIT2010, 2010) que é um *framework* para a linguagem *Java* e sobre o *Google Test Framework* (TEST, 2010) utilizado para a linguagem *C++*, sendo o último utilizado para codificar os primeiros testes unitários na disciplina de AEDS II.

2.2.1.1 JUnit *Framework*

O *JUnit* é um *framework open source* criado por *Eric Gamma e Kent Beck* para a codificação de testes unitários na linguagem *Java*. No momento da escrita deste documento, a versão disponível do *framework* é a 4.8.2 (JUNIT2010, 2010). O *JUnit* é composto pelos seguintes pacotes: *framework* que é o pacote básico ou núcleo do *framework*; o *runner* utilizado para a execução dos testes; *textui e swingui* para as interfaces de usuário e o *extensions* com funcionalidades adicionais.

Através do *JUnit* é possível criar classes que podem ser organizadas hierarquicamente através dos pacotes, permitindo a realização de testes em partes separadas ou um teste completo do código. *Frameworks* como o *JUnit* têm também como objetivo a criação de classes de testes que possam ser reutilizadas.

Para a utilização do *JUnit* é necessário baixar o arquivo junit.jar do site do *JUnit* e adicionar ao projeto que será testado. Em (RIEHLE, 2008), pode-se encontrar uma documentação do *framework* com uma representação em *Unified Modeling Language - UML*. A Figura 2.1 apresenta o diagrama de Classes do núcleo do *framework* que está documentado em (RIEHLE, 2008). Outras referências apresentam propostas da adoção do *framework JUnit* no desenvolvimento de aplicativos e também no ensino do desenvolvimento de aplicativos orientados a testes (EDWARDS, 2003; WICK; STEVENSON; WAGNER, 2005)

2.2.1.2 *Google Test Framework*

O *Google Test*, também chamado de *gtest*, é um *framework* utilizado para a criação de testes unitários para a linguagem C++ . É um *framework* portável, podendo ser executado em plataformas como *MS-Windows, Linux, Mac OS X*, entre outras. Utilizado em alguns projetos de *software* da empresa Google (TEST, 2010), se encontra atualmente na versão 1.4.0 e é distribuído sob os termos da licença *New BSD License* (LICENSE, 2010).

O *Google Test* foi desenvolvido pela empresa Google com o objetivo de agrupar em um único produto, um conjunto de funcionalidades que não se encontravam presentes em outras opções de *frameworks* disponíveis para a linguagem C++ (FAQ, 2010).

Este trabalho avalia a aplicação deste *framework* na construção de testes unitários para a correção automática de listas de exercícios. A escolha do *framework Google Test* se deve principalmente ao fato de que a disciplina que será utilizada

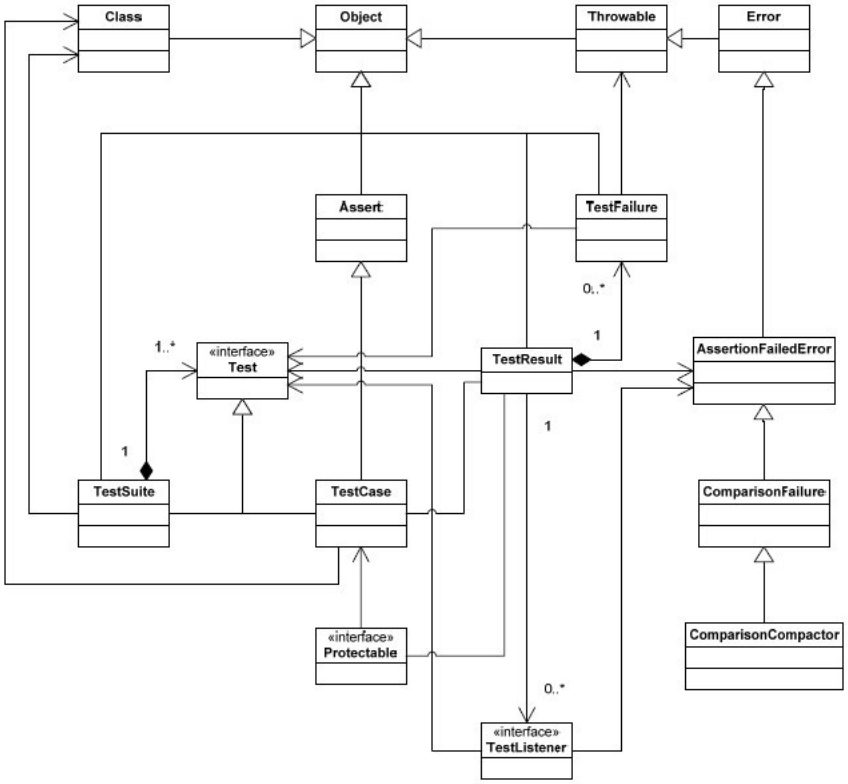

Figura 2.1: Diagrama de Classe do pacote *framework* do *JUnit*. Fonte: (RIEHLE, 2008)

como estudo de caso utiliza a linguagem C++. Maiores informações sobre o uso e os detalhes do *framework* serão apresentados no capítulo 3.

2.3 Estilos utilizados na programação

Para assegurar que a manutenção do *software* possa ser realizada com eficiência e garantir que toda a equipe de desenvolvedores usarão os mesmos critérios na escrita do código, algumas empresas utilizam um guia de estilo de codificação. Todos os membros da equipe poderão ler qualquer parte do código com mais facilidade se for utilizado um mesmo estilo. Dentre os estilos empregados na programação dos códigos na linguagem C++, este trabalho irá usar a codificação utilizada

pela empresa Google (GOOGLE2010, 2010). Este guia de estilo, disponível em (STYLE, 2010), apresenta um conjunto de boas práticas que abrange vários aspectos da linguagem C++. Entre estes aspectos estão a identação do código, especificação dos protótipos e definições de classes e objetos. Além de assegurar a adoção de boas práticas de programação, o guia de estilo da Google possui um aplicativo desenvolvido em *Python* que tem como objetivo avaliar se o código desenvolvido está de acordo com o estilo. Mais detalhes sobre este aplicativo chamado *C++lint* e sua utilização podem ser obtidos em (CPPLINT, 2009).

Dependendo da abordagem utilizada na condução do curso, o aplicativo *C++ lint* pode ser incorporado ao sistema de construção e avaliação dos códigos, a fim de alertar aos alunos sobre as falhas de estilo.

2.4 Licenças de *software*

Nos primeiros momentos do mercado comercial de informática, os lucros das empresas eram obtidos principalmente com a venda de computadores, periféricos e manutenção sobre estes itens. Neste cenário, as empresas produtoras de hardware forneciam gratuitamente alguns programas de computador para uso nessas máquinas. Porém, estes programas tinham funções limitadas comparadas com as funcionalidades hoje disponíveis. Além de fornecer alguns aplicativos, as empresas incentivavam o desenvolvimento e compartilhamento de programas entre os usuários (UCHOA, 2008), pois os aplicativos existentes não atendiam as necessidades de todos. Neste momento, as empresas não utilizavam a venda de programas de computador como fonte de renda porque o mercado consumidor era reduzido e não lucrativo. Os programas eram apenas acessórios que deviam ser compartilhados para possibilitar o aumento da venda de computadores.

Com o tempo porém, este paradigma se modificou. As empresas começaram a direcionar suas atividades para o desenvolvimento e venda de aplicativos. A antiga prática de compartilhamento livre de programas entre os usuários passou a ser prejudicial aos negócios. Para evitar que os aplicativos produzidos fossem compartilhados , as empresas passaram a utilizar leis de direitos autorais para protegerem os seus produtos. Dentro desse novo cenário, um usuário insatisfeito chamado *Richard Stallman* (STALLMAN, 2010), pesquisador do *MIT*, resolveu convidar outros programadores para criar um sistema operacional livre chamado *GNU*, que pudesse ser usado e aperfeiçoado por todos, de forma semelhante ao que acontecia nos anos anteriores. *Stallman* foi o fundador da *Free Software Fundation - FSF*, uma entidade sem fins lucrativos que tem como principal objetivo gerenciar o de-

senvolvimento de *software* livre. A sigla *GNU*, um acrônimo que significa *GNU* não é *UNIX*, fazendo a referência que este sistema não era um sistema operacional proprietário *UNIX*.

Seguindo a filosofia de *Stallman, Linus Torvalds* (TORVALDS, 2007), um estudante da universidade de *Helsink* convidou um grupo de programadores para contribuir com um *kernel* de um sistema operacional que ele havia desenvolvido a partir de uma necessidade acadêmica. Ele queria fazer conexões remotas aos servidores da universidade que trabalhava a partir de sua residência. Este *kernel* estava se transformando em um sistema com funcionalidades que o aproximavam de um sistema operacional completo. Com a evolução deste *kernel* e com a incorporação de um conjunto de aplicativos que haviam sido desenvolvidos pelo projeto *GNU*, nasceu uma das mais bem sucedidas iniciativas de *software* livre, o sistema operacional *LINUX*.

Todas as licenças utilizadas pelo projeto *GNU* publicadas pela *FSF* têm respaldo legal nos *EUA* e em todos os países que aceitam o acordo internacional de direitos autorais e patentes.

A partir da fundação da *FSF*, foram criadas licenças de *software* que permitiam que aplicativos licenciados sob estes termos pudessem ser utilizados por todos gratuitamente, protegendo os interesses das comunidades que desenvolvessem os programas. Dentre as licenças derivadas desta iniciativa, pode-se destacar a licença de *software* livre *GPL* e a licença de documentação livre do projeto *GNU*.

2.4.1 Licença *software* livre GPL

Uma das licenças mais amplamente conhecidas no contexto de *software* livre é a *GNU General Public Licence - GNU GPL*. O seu principal objetivo é garantir que todo esforço para construir e aperfeiçoar um *software* seja sempre retornado para a comunidade. Em linhas gerais, um *software GNU GPL* pode ser utilizado por qualquer instituição, pessoa física ou jurídica, com ou sem fins comerciais. O *software* distribuído sob esta licença deve estar sempre acompanhado do código fonte ou deve ser informado onde obter este código, caracterizando-o como código aberto. O código pode ser alterado, mas se houver uma redistribuição, a alteração deve ser incorporada e distribuída também sob as regras da *GPL*. O importante é que nenhum *software GPL* pode ser incorporado a um *software* proprietário e distribuído sem os códigos fontes.

É importante lembrar também que *software* livre não significa *software grátis*, ou seja, apesar de livre, pode-se cobrar pela distribuição das mídias, suporte, treinamento e manutenção. Esta distinção é importante para esclarecer que é possível manter um negócio rentável baseado no desenvolvimento de *software* livre, uma vez que grandes empresas como a *IBM* distribuem *software* livre cobrando apenas pelo suporte e treinamento. A licença *GPL* tem algumas diretrizes dentre as quais destacam-se (GPL3, 2010):

- todo usuário deve ter acesso ao código fonte e este deve estar disponível de maneira que o usuário possa obtê-lo gratuitamente;
- o usuário tem o direito de copiar e distribuir tanto o código fonte como o executável, desde que estes estejam acompanhados de avisos de direitos autorais e cópias das licenças, ou deve estar claro onde obter estas informações;
- o usuário pode fazer alterações do código, registrando a data e autoria das modificações. Além disso, o produto gerado com esta contribuição deve ser distribuído também sob a licença *GPL*.

A licença *GPL versão 1* foi criada em fevereiro de 1989 (GPL1, 2008). A partir de então, com o objetivo de solucionar possíveis divergências e devido às contribuições da comunidade, outras versões foram lançadas. No momento do desenvolvimento deste documento, a versão da licença *GPL* é a 3 (GPL3, 2010), lançada em julho de 2007. O que se tem observado ao longo da evolução da licença *GPL* é a busca da solução de alguns pontos divergentes. Porém, a cada versão lançada com a correção de alguns pontos, são realizadas novas críticas por membros importantes da comunidade de *software* livre como *Linus Torvalds*. Existem outros tipos de licenças como a licença Apache, a licença estilo *MIT* e a licença estilo *BSD* que permitem que os aplicativos protegidos por elas possam ser utilizados em aplicativos proprietários. Estas licenças não são compatíveis com o estilo da licença *GPL*.

Para licenciar um *software* através da licença *GPL* ou da *LGPL* é necessário a inserção de dois elementos em cada arquivo fonte: (GPLHT, 2009)

1. uma informação de *Copyright*, por exemplo, *Copyright* Natã Goulart 2009;
2. nota de permissão de cópia dizendo que o *software* é distribuído segundo os termos da licença *GPL* ou *LGPL*;

Independente da linguagem em que o *software* esteja sendo distribuído, deve-se sempre usar a palavra *Copyright* em inglês.

Todo programa licenciado sob a licença *GPL* ou *LGPL* deve conter, junto com o código distribuído, uma cópia da licença *GPL*. Um nome recomendado para o arquivo da licença é *COPYING*. Se o *software* é licenciado sob a licença *LGPL*, deve-se inserir um arquivo com a licença *LGPL*, normalmente chamado *COPYING.LESSER*.

Para um programa constituído de apenas um arquivo, após a nota de *Copyright*, deve-se descrever a norma de permissão de cópia. A seguir, um exemplo desta nota:

> *This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.*
> *This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.*
> *You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>*

Se o programa for constituído de mais de um arquivo, deve-se alterar as linhas iniciais.

> *This file is part of Foobar.*
> *Foobar is free software: you can redistribute ...*

Se a licença utilizada é a *LGPL*, deve-se inserir o termo *LESSER* antes das citações *GPL*.
Não há necessidade de realizar o registro de Copyright do *software*. Seguindo o procedimento anteriormente descrito e distribuindo o *software*, este já estará protegido pelas leis de patentes.

2.4.2 Licença Pública Geral Menor do GNU

O objetivo da Licença Pública Geral Menor, conhecida como *LGPL*, é permitir que programas proprietários utilizem os códigos licenciados como *LGPL*. O principal

uso deste tipo de licença é na construção de bibliotecas de *software*. Um exemplo é a biblioteca *GNU C Library (glibc)*, utilizada por vários aplicativos do sistema Linux. Como esta biblioteca é licenciada pela licença menor, aplicativos proprietários como o *Acrobat Reader* podem utilizar suas funções sem quebrar os termos de uso do licenciamento.

A forma de licenciamento deve ser adotada de acordo com o objetivo de uso do *software*. Se o objetivo é desenvolver uma biblioteca com os conceitos de *software* livre, que possa ser usada por um *software* proprietário, deve-se utilizar a licença *LGPL*. Caso não seja necessária a ligação com programas de computador proprietários, recomenda-se usar a licença *GNU GPL* tradicional.

Um documento presente em (LGPL, 2010) apresenta alguns indicativos de quando licenciar um biblioteca com a licença *LGPL* ou a *GPL*. A idéia básica é sempre fortalecer o movimento de *software* livre. Se uma biblioteca criada tem funcionalidades que já estão presentes em outras bibliotecas proprietárias, exemplo a *GNU C Library (glibc)*, não faz sentido licenciá-la com a licença *GPL*, pois não haverá interesse dos programas de computador proprietários em usá-la. Por outro lado, se há um diferencial na biblioteca produzida, deve-se utilizar a licença *GPL*, pois apenas aplicativos *GPL* poderão utilizá-la.

2.4.3 Licença GNU FDL

A *FSF* criou uma licença exclusiva para documentos publicados na Internet, mídia impressa ou eletrônica, com o objetivo de garantir aos autores os mesmos direitos que os desenvolvedores de *software* obtiveram com o uso de licenças como a *GPL*. Esta licença, chamada de *GNU Free Documentation License - GNU FDL* (FDL, 2008), protege estes autores contra o uso de seus textos por outras pessoas e entidades de forma semelhante com que a *GPL* protege os aplicativos. Esta licença tem sido utilizada para a produção de documentação de *software* livre.

A versão da *FDL* disponível no momento da escrita deste documento é a 1.3 de novembro de 2008 (FDL, 2008). Em termos gerais, a *FDL* permite que um documento seja distribuído sob qualquer forma ou método desde que os termos da licença estejam juntos com esta reprodução. Não poderá ser, de forma alguma, criada restrições que limitem a distribuição e reprodução futura do conteúdo licenciado. Um documento pode ser alterado, mas deve continuar tendo todas as restrições da licença *FDL* tradicional. O documento que descreve a licença deixa claro que é possível imprimir e vender cópias do documento licenciado. A versão 1.3 da *FDL* tem algumas opções de uso importantes comparadas com as versões

anteriores. Dentre estas opções existe uma que garante ao autor o direito de criar seções que nunca possam ser alteradas ou retiradas do documento. Esta opção é chamada de *invariant sections*. A descrição completa da *FDL* em sua última versão está disponível no site da *FSF* (FDL, 2008).

Para licenciar um documento sob a licença *FDL*, o autor deve incluir uma cópia do conteúdo da licença junto ao documento e inserir o seguinte conteúdo logo após a página inicial com as devidas alterações:

> *Copyright (C) YEAR YOUR NAME.*
> *Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.*
> *A copy of the license is included in the section entitled "GNU Free Documentation License".*

Se o documento utiliza alguma seção que não deve ser alterada, a linha ***with ... texts*** deve ser substituída por:

> *with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.*

Além disso, se o documento contiver alguns códigos de programa ou *scripts* relevantes, estes devem ser licenciados também, em paralelo, pela licença *GPL*.

Capítulo 3

Framework Utilizado para Criação das Listas

A metodolodia utilizada para a criação de listas de exercícios com o uso de testes unitários parte de um conjunto de exercícios descritos em enunciados que são normalmente utilizados pelos cursos de graduação, criando para cada problema destas listas, um conjunto de testes que irão verificar se a codificação criada pelos alunos apresenta o resultado esperado. Para a construção dos testes unitários será utilizado o *framework Google Test*.

3.1 Google Test

O *Google Test framework* permite a criação de testes em sistemas como *Windows, Linux e Mac OS* e foi construído baseado nos seguintes pontos(GUIDE, 2010):

- os testes devem ser independentes e passíveis de repetição. Realizar testes que podem falhar em decorrência de erros de outros testes atrapalham a produtividade. O *Google Test* isola cada teste em um objeto, permitindo que apenas partes com erros sejam testadas isoladamente;

- os códigos dos testes devem ser bem organizados para que permitam a reutilização do código através do compartilhamento de dados e rotinas, o que facilita também a manutenção do código;

- testes devem ser portáveis e reusáveis. O *Google Test* pode ser usado em compiladores para diferentes sistemas operacionais e plataformas. Estas características motivam a comunidade *open source* a contribuir com o projeto;

- quando um teste falha, ele deve promover informações sobre o erro com um nível adequado de detalhes. No *Google Test*, a falha em um teste não interrompe a execução do programa. A interrupção é realizada apenas no teste que falhou e o próximo teste é executado. Com esta abordagem, em um único ciclo de testes, é possível avaliar vários erros.

Com o objetivo de aumentar a velocidade dos testes, o *Google Test* pode usar mais de uma vez um conjunto de recursos compartilhados, sem comprometer a independência entre os testes.

O *Google Test* é baseado na arquitetura *xUnit* (HAMILL, 2004) e permite aos testadores que conhecem outros *frameworks* baseados nesta arquitetura, como o *JUnit*, a possibilidade de uma rápida curva de aprendizado.

O *Google Test* deve ser compilado em uma biblioteca para ser utilizado em um projeto. Após a criação da biblioteca, ela deve ser inserida no projeto. São disponibilizados pelo *Google Test* arquivos que auxiliam a construção dos projetos de teste. Dentre estes arquivos de construção estão disponibilizados arquivos para os compiladores *Microsoft Visual Studio*, para o *make* utilizado pelo *Linux* e o *codegear* utilizado pelo *Borland C++*.

Pode-se iniciar o uso do *Google Test* com a utilização de *assertions* que são um conjunto de código que verificam uma determinada condição. Os possíveis resultados de um *assertion* são sucesso, falha não fatal e falha fatal do inglês *success, nofatal failure ou fatal failure*. Quando uma *fatal failure* ocorre, o teste é interrompido.

Um caso de teste pode conter um ou mais testes e, geralmente, um programa tem vários casos de teste. Deve-se agrupar os testes em casos de teste de forma a refletir a estrutura lógica do código que será avaliado. Quando vários testes em um caso de teste utilizam objetos e rotinas compartilhadas, eles são reunidos em uma classe chamada *test fixture*.

3.1.1 *Assertions*

Assertions no *Google Test* são macros que comparam o resultado gerado por uma função com um resultado esperado. Para cada classe ou função podem ser avali-

ados vários valores de saída baseados em dados de entrada informados. Quando um teste falha, é gerada uma mensagem informando o arquivo e a linha que gerou o erro, juntamente com uma mensagem de erro. Pode ser codificada também uma mensagem personalizada para cada tipo de erro gerado, informando, por exemplo, os dados de entrada e a saída esperada.

Existem dois conjuntos básicos de *assertions* que apresentam diferentes comportamentos quando um teste falha. Se for desejado que ao ocorrer uma falha, todo o teste seja interrompido, utiliza-se as *assertions* iniciadas com $ASSERT\_*$. Para evitar que o teste seja interrompido deve-se usar as *assertions* iniciadas com $EXPECT\_*$. Como uma *assertion* $ASSERT\_*$ interrompe a execução dos testes, é importante verificar rotinas como a remoção de bibliotecas codificadas em testes que podem deixar de ser executados.

A instrução a seguir apresenta um uso de uma *assertion* que compara o tamanho de dois vetores e imprime uma mensagem personalizada, caso os tamanhos sejam diferentes:

> $ASSERT\_EQ$(x.size(), y.size()) « "Os vetores x e y tem tamanhos diferentes.";

A Tabela 3.1 apresenta algumas *assertions* utilizadas pelo *Google Test* com os comportamentos *Fatal e Nonfatal*.

Tabela 3.1: *Assertions* utilizadas pelo Google Test

Fatal *Assertion*	**Nonfatal** *Assertion*	**Resultado**
Retnorna true ou false		
$ASSERT\_TRUE(condition)$;	$EXPECT\_TRUE(condition)$;	condition is true
$ASSERT\_FALSE(condition)$;	$EXPECT\_FALSE(condition)$;	condition is false
Compara dois valores		
$ASSERT\_EQ(expected, actual)$;	$EXPECT\_EQ(expected, actual)$;	expected == actual
$ASSERT\_NE(val1, val2)$;	$EXPECT\_NE(val1, val2)$;	val1 != val2
$ASSERT\_LT(val1, val2)$;	$EXPECT\_LT(val1, val2)$;	val1 < val2
$ASSERT\_LE(val1, val2)$;	$EXPECT\_LE(val1, val2)$;	val1 <= val2
$ASSERT\_GT(val1, val2)$;	$EXPECT\_GT(val1, val2)$;	val1 > val2
$ASSERT\_GE(val1, val2)$;	$EXPECT\_GE(val1, val2)$;	val1 >= val2

Nas *assertions* $ASSERT\_EQ*$, $EXPECT\_EQ*$ e similares, o valor que se deseja testar deve ser colocado na posição do parâmetro *actual* e o valor esperado em *expected* porque o *Google Test* está codificado para emitir mensagens segundo esta convenção.

A *assertion ASSERT_EQ* compara se os endereços de memória dos parâmetros são os mesmos, ou seja, faz uma comparação dos ponteiros. Se o objetivo é comparar o conteúdo de duas *strings*, devem ser utilizadas outras *assertions*. Se o objetivo é comparar dois objetos, *ASSERT_EQ* pode ser usada.

Para realizar a comparação entre duas *strings* podem ser utilizadas as *assertions* descritas na Tabela 3.2. O uso da palavra *CASE* significa que não importa se as letras estarão em maiúsculo ou minúsculo. A comparação entre um ponteiro nulo em uma *string* vazia é considerada diferente.

Tabela 3.2: *Assertions* utilizadas pelo Google Test para strings

Fatal *Assertion*	**Nonfatal** *Assertion*
$ASSERT\_STREQ(expected\_str, actual\_str);$	$EXPECT\_STREQ(expected\_str, actual\_str);$
$ASSERT\_STRNE(str1, str2);$	$EXPECT\_STRNE(str1, str2);$
$ASSERT\_STRCASEEQ(expected\_str, actual\_str);$	$EXPECT\_STRCASEEQ(expected\_str, actual\_str);$
$ASSERT\_STRCASENE(str1, str2);$	$EXPECT\_STRCASENE(str1, str2);$

Na primeira linha da Tabela 3.2, o teste verifica se as duas *strings* têm o mesmo conteúdo. Na segunda linha é verificado se os conteúdos são diferentes. As terceira e quarta linhas fazem as mesmas verificações não importando se os caracteres estão em maiúsculo ou minúsculo.

Para criar um teste deve-se usar a macro *TEST()* que define as funções de teste. Dentro destas funções são utilizadas as *assertions* que farão as verificações dos testes. O primeiro argumento da função *TEST()* é o nome do caso de teste e o segundo argumento o nome do teste no caso de teste. Como um caso de teste pode conter vários testes, um nome completo de um teste é composto pelo nome do caso de teste mais o nome do teste, podendo ocorrer que testes de diferentes casos de teste tenham o mesmo nome.

Para uma função chamada *factorial* que calcula o fatorial de um número inteiro retornando um inteiro como resultado, um caso de teste pode ser construído conforme a Figura 3.1

Testes que têm um mesmo relacionamento lógico devem ser agrupados em um mesmo caso de teste. O *Google Test* agrupa os resultados dos testes por casos de teste. No exemplo da Figura 3.1 o caso de teste *FactorialTest* tem dois testes, o *HandlesZeroInput* e o *HandlesPositiveInput*.

```
// Verifica o fatorial de 0.
TEST(FactorialTest, HandlesZeroInput) {
  EXPECT_EQ(1, Factorial(0));
}
// Verifica o fatorial para um conjunto de números positivos
TEST(FactorialTest, HandlesPositiveInput) {
  EXPECT_EQ(1, Factorial(1));
  EXPECT_EQ(2, Factorial(2));
  EXPECT_EQ(6, Factorial(3));
  EXPECT_EQ(40320, Factorial(8));
}
```

Figura 3.1: Exemplo caso teste. Fonte: (PRIMER, 2010)

3.1.2 *Test Fixtures*

Test fixtures podem ser utilizados quando dois ou mais testes utilizam dados similares a fim de se utilizar as mesmas configurações para este conjunto de testes. Para usar um *test fixture*, devemos usar a palavra reservada $TEST\_F()$ no lugar de $TEST()$. Com o uso da instrução *SetUp()* que é automaticamente executada antes de cada teste e da instrução *TearDown()* executada após a cada teste, o mesmo conjunto de dados pode ser iniciado e removido da memória, garantido a independência entre as execuções, por outro lado, tendo como desvantagem um aumento do custo computacional. Se cada teste não modifica os dados utilizados, pode-se utilizar as instruções *SetUpTestCase() e TearDownTestCase()* conforme exemplo da Figura 3.2 que apenas inicia os dados no inicio do caso de teste e remove os dados da memória no final do caso do teste, tornando a execução do teste mais eficiente.

Após a definição de todos os testes, eles podem ser executados através da chamada $RUN\_ALL\_TESTS()$ que irá retornar 0 se todos os testes foram executados com sucesso e 1 de outro modo. A chamada irá salvar o estado de todos os *flags* do *Google Test*, criar um objeto *texfixture* para a execução do primeiro caso de teste, iniciar o objeto através da chamada *SetUp()*, executar os testes no objeto, executar a instrução *Teardown()*, apagar a *fixture* e restaurar o estado de todos os *flags* do *Google Test*. Isto será executado para cada teste contido no caso de teste. A função $RUN\_ALL\_TESTS()$ deve ser executada apenas uma vez e seu valor de retorno não deve ser ignorado pois podem ocorrer erros em compiladores como o *gcc*.

```
class FooTest : public ::testing::Test {
 protected:
  // Declaração do set-up para uso no caso de test
  // é chamado antes da execução dos casos de test
  static void SetUpTestCase() {
    //declaração de dados compartilhados
    shared_resource_ = new ...;
  }

  // tear-down para o caso de test
  // chamado após a execução do último teste
  static void TearDownTestCase() {
    delete shared_resource_;
    shared_resource_ = NULL;
  }

  // Pode-se definir as rotinas set-up e tear-down para
  // cada caso de teste.
  virtual void SetUp() { ... }
  virtual void TearDown() { ... }

  // declaração de um recurso compartilhado entre todos os testes
  static T* shared_resource_;
};

T* FooTest::shared_resource_ = NULL;

TEST_F(FooTest, Test1) {
  //local que pode-se utilizar os recursos compartilhados
}
TEST_F(FooTest, Test2) {
  //local que pode-se utilizar os recursos compartilhados
}
```

Figura 3.2: Uso de test fixture. Fonte: (PRIMER, 2010)

Os programas do *Google Test* podem ser executados diretamente pela linha de comando para se verificar os resultados dos testes. Para se verificar os parâmetros que podem ser passados para o programa de teste, pode-se usar o parâmetro *-h* na linha de comando, após o executável de um teste. É possível verificar quais os testes estão contidos em executável de teste, através do parâmetro

$--gtest\_list\_tests$. Se no Linux o executável de testes se chamar *teste1*, listar os testes nesse arquivo consiste em executar o comando: $./teste1 --gtest\_list\_tests$. De posse dos nomes dos testes, estes podem ser executados individualmente ou com o uso de caracteres curinga como o caractere $*$. Para o teste *teste1* citado anteriormente, executar o programa sem parâmetros ou utilizando o parâmetro $--gtest\_filter = *$ irá fazer com que todos os testes contidos no arquivo sejam executados. Para se executar todos os testes presentes no caso de teste *AvaliaTest* pode-se usar o comando:$./teste1 --gtest\_filter = AvaliaTest.*$.

Pode-se desabilitar um teste ou todo um caso de teste adicionando a palavra *DISABLE* antes do teste ou do caso de teste conforme exemplo da Figura 3.3

```
// Desabilitando teste DoesABC
TEST(FooTest, DISABLED_DoesAbc) { ... }
//Desabilitando caso de teste BarTest
class DISABLED_BarTest : public ::testing::Test { ... };.
TEST_F(DISABLED_BarTest, DoesXyz) { ... }
```

Figura 3.3: Desabilitando testes e casos de teste.

O *Google Test* gera a saída referente aos resultados dos testes para a saída padrão. Pode-se direcionar a saída dos resultados para um arquivo *xml* através do uso do parâmetro $--gtest\_output$.

O *Google Test* tem o objetivo de trabalhar com a execução de testes através de *threads*, mas até o momento da escrita deste documento, ainda não é possível utilizar estas funcionalidades.

As informações presentes nesta seção foram baseadas nos documentos disponíveis na *wiki* do projeto do *Google Test*. Maiores informações podem ser obtidas em (GUIDE, 2010; PRIMER, 2010)

3.2 Hospedagem do projeto open source

O ambiente escolhido para hospedar os códigos e documentações do projeto é o ambiente disponibilizado pela Google (GOOGLE2010, 2010), o *Google Code* (CODE, 2010). Através do site do *Google Code* pode-se criar projetos *open source* através de formulários *web*. O *Google Code* oferece um espaço de até 1 GB para armazenamento de códigos que podem ser controlados por ferramentas de controle

de versão com o *Subversion* (COLLINS-SUSSMAN BRIAN W. FITZPATRICK, 2004), 2 GB de espaço de armazenamento de conteúdo para download, ferramenta de acompanhamento de erros no código e suporte na construção de *wikis*.

Após a criação do projeto, o dono do projeto pode configurar quais usuários poderão participar com contribuições e revisões de código e do conteúdo. Com a definição dos papéis dos colaboradores do projeto, recomenda-se a configuração da lista de emails para que os membros do projeto sejam notificados quando for identificado algum erro no *software* ou ocorra uma atualização.

Cada projeto no *Google Code* tem seu próprio repositório, sendo permitido a qualquer usuário acessá-lo no modo de leitura. Os membros do grupo administrador podem alterar o conteúdo de todo o repositório.

A documentação do projeto é geralmente construída através de páginas no formato *wiki* pelos administradores e os erros e atualizações descritos na seção *issue* do projeto.

O projeto de correção automática de exercícios será organizado por área de conhecimento de forma hierárquica com o objetivo de organizar as contribuições recebidas da comunidade *open source* por cursos, disciplinas, tópicos e também a linguagem de programação utilizada.

Capítulo 4

Construção e Aplicação das Listas de Exercícios

Após a construção do enunciado das listas de exercícios de acordo com a ementa da disciplina, foram codificados os testes unitários para verificar os códigos dos alunos. Estas listas foram utilizadas em aulas práticas de laboratório e, posteriormente, os arquivos serão disponibilizados no site do projeto *Easy Testing* (ETESTING, 2010), Figura 4.1, criado para armazenar os arquivos das listas de exercícios. A utilização da lista em uma aula prática permite que os alunos avaliem a corretude do código através das várias possibilidades de entrada e saída para cada problema a ser resolvido. Se algum caso de teste básico não esteja sendo tratado, é provável que alguns alunos possam identificar a falha.

No site do projeto, para cada lista, são disponibilizados os arquivos com o enunciado do exercício, a biblioteca do *Google Test*, um arquivo para ser codificado pelo aluno e um outro arquivo contendo uma das possíveis codificações que resolvem o enunciado. Inicialmente, serão disponibilizados também um *makefile* para utilização da lista no *gcc* do sistema *Linux* e os arquivos para uso no compilador *DevC++* para o *Windows*.

Nessa seção serão descritos detalhes de uma lista de exercícios sobre tipos abstratos de dados aplicada para a turma AEDS II do DCC da UFMG em março de 2010.

O enunciado da lista de exercícios está descrito a seguir:

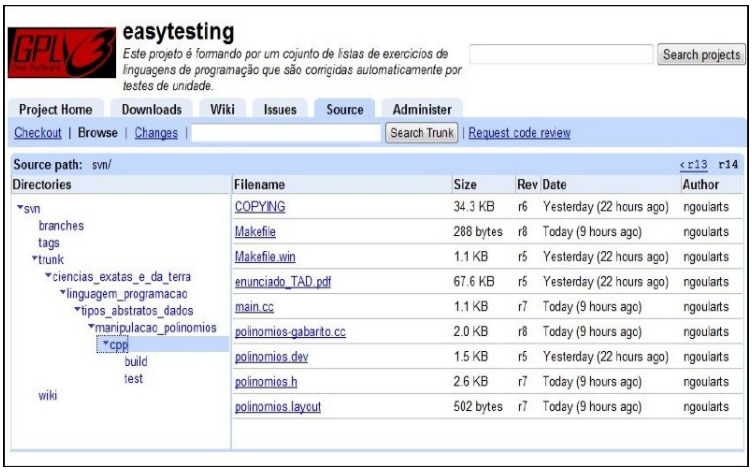

Figura 4.1: Site do projeto Easy Testing

Introdução:

Um polinômio de grau n é uma função do tipo $P(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$, onde $a_0, a_1, a_2, ..., a_n$ pertecem ao conjunto dos números reais e $a_n \neq 0$. Qualquer polinômio de grau n pode ser representado em um programa de computador por um vetor p com n + 1 posições, onde cada posição $[i]$ do vetor armazena o valor do coeficiente $a_i, i \in 0, ..., n$. Por exemplo: o polinômio de grau 4, $Q(x) = 5 + 3x^2 + 2x^4 = 5x^0 + 0x^1 + 3x^2 + 0x^3 + 2x^4$, pode ser representado pelo vetor float q[5] = { 5, 0, 3, 0, 2}

Questão 1. Crie um tipo abstrato de dados - TAD polinômio de acordo com as interfaces presentes no arquivo *polinomio.h*;

Questão 2. Implemente o método float Polinomio::Avaliar(float x) que retorna o valor do polinômio corrente no ponto x.

Questão 3. Implemente o método void Polinomio::Atribuir(Polinomio& q) que faz com que o polinômio corrente fique igual ao polinômio q passado como parâmetro.

Questão 4. Implemente o método void Polinomio::Somar(Polinomio& p1, Polinomio &p2) que atribui ao polinômio corrente a soma dos polinômios p1 e p2 de mesmo grau passados como parâmetro.

Questão 5. Implemente o método void Polinomio::Derivar(Polinomio& q) que faz com que o polinômio corrente fique igual a derivada do polinômio q passado como parâmetro.

DICA: A derivada de um polinômio de grau n > 1, $P(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$, é um polinômio de grau n - 1 e pode ser calculada da seguinte forma: $P'(x) = a_1 + 2a_2x + 3a_3x^2 + ... + na_nx^(n-1)$. Por exemplo: $Q'(x) = 2*3*x(2-1) + 4*2*x(4-1) = 6x + 8x3$.

Questão 6. Implemente o método void Polinomio::Integrar(Polinomio& q) que faz com que o polinômio corrente fique igual a integral do polinômio q passado como parâmetro.

DICA: A integral de um polinômio de grau n, $P(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$, é um polinômio de grau n + 1 e pode ser calculada da seguinte forma: $integral[P(x)] = c + a_0x + (a_1/2)x^2 + (a_2/3)x^3 + ... + (a_n/(n+1))x^n + 1$. Assuma que a constante c é igual a 0. Por exemplo: $integral[Q(x)] = 5x + (3/3)x^3 + (2/5)x^5 = 5x + x^3 + 0.4x^5$.

O pacote de arquivos que formam a lista de exercícios é composto de um arquivo com o enunciado da lista de tipos abstratos de dados chamado *ListaTAD.pdf*, um diretório chamado *test* com a biblioteca do *Google Test* e a codificação dos testes unitários através dos arquivos *teste.h* e *teste.cc*. No diretório raiz da lista há um arquivo chamado *main.cc* que é o arquivo que irá gerar o executável que exibe os resultados do algoritmo, além deste arquivo, esta pasta contém o arquivo *polinomio.h* com as interfaces da classe Polinomio e o arquivo *polinomio.cc* que deve ter a codificação que resolve as questões.

Os testes unitários foram codificados através do arquivo *test.h* que contém a função *ExecutarTodosOsTestes* que executa todos os testes unitários que validam a codificação dos alunos e os códigos de teste. Na aplicação desta lista, para facilitar o uso pelos alunos e deixar transparente a utilização dos testes unitários, dentro do arquivo *main.cc* há uma chamada para a função *ExecutarTodosOsTestes* que irá executar todos os testes de uma única vez. Desta forma, na aplicação desta lista não foram necessários conhecimentos por parte dos alunos das técnicas de uso dos testes unitários.

A Figura 4.2 tem o código do arquivo *test.cc* com a chamada para a função *ExecutarTodosOsTestes*.

A Figura 4.3 contém uma das possíveis soluções para a rotina soma de polinômios descrito no enunciado da lista. Na Figura 4.4 há uma parte do código do

```
#include "test.h"
int ExecutarTodosOsTestes() {
  int argc = 0;
  char** argv = NULL;
  ::testing::InitGoogleTest(&argc, argv);
  ::testing::GTEST_FLAG(print_time) = false;
  int success = RUN_ALL_TESTS();
  return success;
}
```

Figura 4.2: Arquivo test.cc com a função para a execução de todos os testes.

arquivo *test.h* que faz testes sobre a rotina soma de polinômios. A Figura apresenta um caso de teste chamado *Teste* e um dos testes sobre a soma dos polinômios que verifica a soma de polinômios com coeficiente não nulos.

```
void Polinomio::Somar(Polinomio& p1, Polinomio &p2) {
    n = p1.grau() + 1;
    for (int i = 0; i < n; i++) {
        a[i] = p1.get(i) + p2.get(i);
    }
}
```

Figura 4.3: Uma possível solução para a soma de polinômios

Na primeira linha é iniciado um vetor $coef\_p$ com os valores $[1,2,3,4]$, na linha 2 é instanciado o objeto polinômio p de grau 4 com os seus coeficientes. Na linha 3 é iniciado o vetor $coef\_q$ $[4,3,2,1]$ e o polinômio q na linha 4. O vetor $coef\_esperado$ com os valores do resultado da soma é iniciado na linha 5 e o objeto esperado e instanciado na linha 6. Na linha 8 é instanciado o polinômio soma e o método Somar é executado na linha 9, passando os polinômios p e q como parâmetros. Na linha 10 é verificado se a função *MostrarComoVetor* que recebe o polinômio esperado é igual a função que recebe o polinômio soma. Se os valores forem iguais, nada será exibido por este teste, caso contrário, se não forem iguais, será exibido a mensagem presente nas linhas 11 a 27.

No apêndice, capítulo A, está disponível a codificação de todos os casos de teste para a lista de exercícios.

```
TEST_F(Teste, Somar_polinomios_com_coeficientes_nao_nulos) {
1-   float coef_p[] = {1, 2, 3, 4};
2-   Polinomio p(4, coef_p);
3-   float coef_q[] = {4, 3, 2, 1};
4-   Polinomio q(4, coef_q);
5-   float coef_esperado[] = {5, 5, 5, 5};
6-   Polinomio esperado(4, coef_esperado);
7-   // Soma p1 e p2.
8-   Polinomio soma;
9-   soma.Somar(p, q);
10-  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(soma))
11-    << "-----------------------------------------------------\n"
12-    << "Erro na chamada da funcao: "
13-    << "soma.Somar(p, q)\n"
14-    << "   onde:\n"
15-    << "      p = " << MostrarComoVetor(p) << "\n"
16-    << "      q = " << MostrarComoVetor(q) << "\n"
17-    << "      'soma' eh o polinomio que armazena a soma.\n"
18-    << "Resultado esperado:soma=="<< MostrarComoVetor(esperado)
19-    << "Resultado da funcao:soma=="<< MostrarComoVetor(soma)<<"\n"
20-    << "Exemplo:\n"
21-    << "   P(x) = " << MostrarPorExtenso(p) << "\n"
22-    << "   Q(x) = " << MostrarPorExtenso(q) << "\n"
23-    << "   P(x) + Q(x) = "
24-    << MostrarPorExtenso(esperado) << "\n"
25-    << "   Sua funcao retornou o polinomio: "
26-    << MostrarPorExtenso(soma) << "\n"
27-    << "-----------------------------------------------------\n";
}
```

Figura 4.4: Parte do arquivo test.h com o teste *Somar_polinomios_com_coeficientes_nao_nulos*

4.1 Resultados e Discussão

Como estudo de caso para esta proposta, foi utilizada a turma AEDS II do curso de Engenharia de Controle e Automação oferecida no departamento de Ciência da Computação da UFMG. Esta turma tem 50 alunos matriculados, tendo participado das aulas práticas em média 35 alunos, de acordo com a lista de presença da turma. Foram inicialmente entregues os enunciados com as questões sem nenhuma informação adicional sobre o uso de testes unitários. A função que executava os testes unitários estava comentada dentro do arquivo *main.cc* presente na lista. Após os

alunos terem codificado alguns dos métodos que resolvem as listas, foi informado que eles poderiam utilizar a função que executava os testes para avaliar os seus códigos. Pôde ser verificado junto a alguns alunos, através do acompanhamento individual em cada computador, que os testes auxiliavam na correção dos erros, principalmente através das mensagens personalizadas que exibiam exemplos de entrada e saída para cada método. Ainda através deste acompanhamento, pôde ser verificado que o relatório exibido pelo *Google Test* auxiliou na resolução de todas as questões da lista.

Após a aplicação de duas listas de exercícios observou-se, através de entrevistas informais com os alunos que o uso de testes unitários foi um agente motivador para o aprendizado das técnicas de linguagem de programação, permitindo que o professor direcionasse sua atenção para as questões mais complexas e sobre tópicos adicionais da disciplina. Foi avaliada a possibilidade de aplicar uma questionário para a turma deste estudo de caso a fim de obter algumas informações sobre a metodologia, mas o professor da disciplina, juntamente com outros professores do DCC da UFMG entenderam que devido ao pouco tempo de uso, os resultados não seria relevantes, ficando o questionário para um trabalho futuro.

Dentre os pontos positivos observados no uso dos testes unitários pode-se descrever:

- não foi necessário ensinar aos alunos as técnicas de como construir e utilizar os testes unitários. Estes geraram relatórios de erro que permitiam aos alunos a construção do algoritmo sem a necessidade de conhecer detalhes de funcionamento dos testes;
- através da codificação de mensagens personalizadas para os testes, os alunos conseguiram identificar e resolver os problemas ocorridos;
- como as dúvidas simples eram resolvidas pelos relatórios dos testes, o professor pôde dar atenção para as questões mais complexas.

Alguns pontos negativos foram observados no uso da metodologia:

- a metodologia não permite o teste da função principal de cada programa, *main*, e as suas entradas e saídas. Para esta verificação pode-se fazer uso complementar de sistemas como os juízes *online*.
- na construção de uma nova lista pode ocorrer que determinados casos de teste não sejam codificados.

- outro ponto importante já citado neste documento é que as interfaces das classes são previamente definidas na entrega do exercício e não podem ser alteradas pelos alunos.

Após a aplicação das listas observou-se que o uso de testes unitários alcançou os objetivos propostos que são, principalmente, o auxílio no ensino de linguagens de programação através da correção automática e a possibilidade de que os professores possam dar uma adequada atenção para uma turma com grande número de alunos. O uso dos casos de teste auxiliam os professores e monitores no ensino e correção de listas de exercícios, permitindo que estes possam dar atenção para os erros mais complexos. Além disso, foi criado o site para o projeto *Easy Testing* (ETESTING, 2010), onde os arquivos serão disponibilizados.

Capítulo 5

Considerações Finais

Neste trabalho foi proposta uma metodologia para correção automática de listas de exercícios utilizando testes unitários. O *framework Google Test* foi utilizado na construção de casos de teste que verificam funções presentes no enunciado das listas de exercício. Através da aplicação deste método em uma turma da disciplina AEDS II do DCC da UFMG, observou-se que o método auxilia no ensino de linguagens de programação, principalmente para turmas com um grande número de alunos. O uso dos testes codificados, com mensagens de erros personalizadas e objetivas, permitiram que os alunos corrigissem os problemas em seu código sem a ajuda dos professores e monitores.

Uma proposta de trabalho futuro é utilizar os testes unitários juntamente com um sistema juiz *Online* e ferramentas de verificação de cópias de trabalho para uma avaliação completa de listas de exercícios. Uma das opções de uso desta integração é o *plugin* para o *Moodle* citado anteriormente neste documento. Com esta integração, pode-se fazer uso também de ferramentas de verificação de estilos como a apresentada neste documento. Deve-se também, após a utilização da metodologia em algumas turmas, aplicar questionários que possibilitem uma análise das informações dos alunos.

Espera-se que ao longo do tempo, sugestões quanto a metodologia e colaborações com outras listas de exercícios possam ocorrer no site do projeto de forma que a qualidade e abrangência desta metodologia possa ser expandida.

Este trabalho terá prosseguimento através de um projeto de pesquisa coordenado pelo prof. Thiago Noronha da Universidade Federal de Minas Gerais com bolsas fornecidas pelo Programa Especial de Graduação - PEG 2010.

Referências Bibliográficas

BEYDEDA, S.; GRUH, V.; STACHORSKI, M. A graphical class representation for integrated black- and white-box testing. In: . [S.l.: s.n.], 2001. p. 706 –715.

CHEANG, B.; KURNIA, A.; OON, A. L. W. On automated grading of programming assignments in an academic institution. *Computers & Education*, v. 41, p. 121–131, dez. 2003. Disponível em: http://www.elsevier.com/wps/locate/compedu.

CODE, G. *Google Code*. Último acesso em 23 de Março de 2010. Http://code.google.com/p/support/ (referência on-line).

COLLINS-SUSSMAN BRIAN W. FITZPATRICK, C. M. P. B. *Version Control with Subversion - Next Generation Open Source Version Control*. Reading: O'Reilly Media, 2004.

CPPLINT. *CPPLINT*. Último acesso em 11 de Março de 2009. Http://google-styleguide.googlecode.com/svn/trunk/cpplint/README (referência on-line).

DUSTIN, E.; RASHKA, J.; PAUL, J. *Automated Software Testing - Introduction, Management and Performance*. Reading: Addison-Wesley, 1999.

EDWARDS, S. H. Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In: *Proc. Int. Conf. Education and Information Systems: Technologies and Applications (EISTA03)*. [S.l.: s.n.], 2003.

ETESTING. *Site do Projeto*. Último acesso em 10 de abril de 2010. Http://code.google.com/p/easytesting/ (referência on-line).

FAQ, G. T. *Google Test Faq*. Último acesso em 11 de Março de 2010. Http://code.google.com/p/googletest/wiki/GoogleTestFAQ (referência on-line).

FDL, G. *GNU FDL*. Último acesso em 11 de Março de 2008. Http://www.gnu.org/licenses/fdl.html (referência on-line).

GOOGLE2010. *Google*. Último acesso em 11 de Março de 2010. Http://www.google.com.br (referência on-line).

GPL1. *GPL 1*. Último acesso em 11 de Março de 2008. Http://www.gnu.org/licenses/old-licenses/gpl-1.0.html (referência on-line).

GPL3. *GPL 3*. Último acesso em 11 de Março de 2010. Http://www.gnu.org/licenses/quick-guide-gplv3.html (referência on-line).

GPLHT. *GPL How To*. Último acesso em 11 de Março de 2009. Http://www.gnu.org/licenses/gpl-howto.html (referência on-line).

GUIDE, G. A. *Google Advanced Guide*. Último acesso em 23 de Março de 2010. Http://code.google.com/p/googletest/wiki/GoogleTestAdvancedGuide (referência on-line).

HAMILL, P. *Unit Test Frameworks - Tools for High-Quality Software Development*. Reading: O'Reilly Media, 2004.

IBM2010. *ACM-ICPC*. Último acesso em 11 de Março de 2010. Http://cm2prod.baylor.edu/welcome.icpc (referência on-line).

JUNIT2010. *JUNIT*. Último acesso em 11 de Março de 2010. Http://www.junit.org/ (referência on-line).

KURNIA, A.; LIM, A.; OON, W.-C. Online Judge. *Computers & Education*, v. 36, p. 299–315, 2001. Disponível em: http://www.elsevier.com/wps/locate/compedu.

KURNIA, A.; MALAFIEJSK, M.; NOINSK, T. Application of an Online Judge and Contester System in Academic Tuition. *Advances in Web Based Learning – ICWL 2007*, v. 4823-2008, p. 343–354, 2008. Disponível em: http://www.springerlink.com/content/y720586585x88622.

LGPL. *Why not LGPL*. Último acesso em 11 de Março de 2010. Http://www.gnu.org/licenses/why-not-lgpl.html (referência on-line).

LICENSE, B. *New BSD License*. Último acesso em 11 de Março de 2010. Http://www.opensource.org/licenses/bsd-license.php (referência on-line).

MISRA, S. Evaluating four white-box test coverage methodologies. In: . [S.l.: s.n.], 2003. v. 3, p. 1739 – 1742 vol.3. ISSN 0840-7789.

MOODLE2010. *MOODLE*. Último acesso em 11 de Março de 2010. Http://moodle.org/ (referência on-line).

MOSS2010. *MOSS Plugin*. Último acesso em 11 de Março de 2010. Http://code.google.com/p/sunner-projects/w/list (referência on-line).

PRIMER, G. T. *Google Test Primer*. Último acesso em 23 de Março de 2010. Http://code.google.com/p/googletest/wiki/GoogleTestPrimer (referência on-line).

RIEHLE, D. Junit 3.8 documented using collaborations. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 33, n. 2, p. 1–28, 2008. ISSN 0163-5948.

SCHLEIMER, S.; WILKERSON, D. S.; AIKEN, A. Winnowing: local algorithms for document fingerprinting. In: *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2003. p. 76–85. ISBN 1-58113-634-X.

STALLMAN. *Stallman*. Último acesso em 11 de Março de 2010. Http://stallman.org/ (referência on-line).

STYLE, G. *Google Style*. Último acesso em 11 de Março de 2010. Http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml (referência on-line).

TEST, G. *Google Test*. Último acesso em 11 de Março de 2010. Http://code.google.com/p/googletest/w/list (referência on-line).

TORVALDS. *Torvalds*. Último acesso em 11 de Março de 2007. Http://www.linux.org/info/linus.html (referência on-line).

UCHOA, K. C. A. *Cibercultura e Software Livre*. [S.l.]: UFLA/FAEPE, 2008.

WICK, M.; STEVENSON, D.; WAGNER, P. Using testing and junit across the curriculum. In: *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*. New York, NY, USA: ACM, 2005. p. 236–240. ISBN 1-58113-997-7.

WOO, Y.-H.; HONG, S.-W.; KIM, S.-B. A study on the self-directed learning management system. In: *NCM '08: Proceedings of the 2008 Fourth International Conference on Networked Computing and Advanced Information Management*. Washington, DC, USA: IEEE Computer Society, 2008. p. 119–122. ISBN 978-0-7695-3322-3-01.

Apêndice A

Códigos fontes

A.1 Arquivo *test.h*

Arquivo com a codificação de todos os testes. Utiliza a biblioteca do *Google Test* através do arquivo *gtest.h*.

```
#include "../polinomios.h"
#include "gtest.h"
#include <sstream>
#include <string>
using std::string;
using std::stringstream;
namespace {
// Classe base dos testes.
class Teste : public ::testing::Test {
 protected:
  // Retorna uma string no formato a + bx + cx^2 + dx^3 + ... + zx^n.
  string MostrarPorExtenso(Polinomio& p) {
    stringstream s;
    bool primeiro = true;
    for (int i = 0; i <= p.grau(); i++) {
      // TODO(tfn): Tratar o caso P(x) = 0.
      if (p.get(i) != 0.0) {
              if (primeiro) {
                s << p.get(i);
                primeiro = false;
```

```
              } else {
                if (p.get(i) == 1.0) {
                  s << " + ";       ;
                } else if (p.get(i) > 0.0) {
                  s << " + " << p.get(i);
                } else if (p.get(i) < 0.0) {
                  s << " - " << -p.get(i);
                }
              }
              if (i == 1) {
                s << "x";
              } else if (i > 1) {
                s << "x^" << i;
              }
      }
    }
    return s.str();
  }
    string MostrarComoVetor(Polinomio& p) {
    stringstream output;
    output << "{" << p.get(0);
    for (int i = 1; i <= p.grau() ; i++) {
      output << ", " << p.get(i);
    }
    output << "}";
    return output.str();
  }
};
// TODO(tfn): Acrescentar o caso para P(x) = 0.
TEST_F(Teste, Testa_o_metodo_MostrarPorExtenso) {
  float coef_p[] = {4, -2.7, 3.8, -5, 1};
  Polinomio p(5, coef_p);
  ASSERT_EQ("4 - 2.7x + 3.8x^2 - 5x^3 + x^4", MostrarPorExtenso(p));

  float coef_q[] = {0, -0.2, 0, 3, -1};
  Polinomio q(5, coef_q);
  ASSERT_EQ("-0.2x + 3x^3 - 1x^4", MostrarPorExtenso(q));

  float coef_t[] = {0, 0, 0, 3, -1};
  Polinomio t(5, coef_t);
  ASSERT_EQ("3x^3 - 1x^4", MostrarPorExtenso(t));
}
<< "\n"
TEST_F(Teste, Avaliar_polinomios_com_coeficientes_nao_nulos) {
  float coef_p[]={4,-2.0,3,-5,1};// P(x)=4 - 2x + 3x^2 - 5x^3 + x^4.
  Polinomio p(5, coef_p);
```

```
  // Avalia P(x = 0.0).
  float x = 0.0;
  ASSERT_EQ(4, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(x)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      x = " << x << "\n"
    << "   Valor esperado : " << 4 << "\n"
    << "   Valor retornado: " << p.Avaliar(x)
<< "Exemplo:\n"
    << "P(x) = 4 - 2x + 3x^2 - 5x^3 + x^4\n"
    << "P(0.0) = 4 - 2*0 + 3*0^2 - 5*0^3 + 0^4 = 4.0\n"
    << "-------------------------------------------------------\n";

  // Avalia P(x = 1.0)
  x = 1.0;
  ASSERT_EQ(1, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(x)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      x = " << x << "\n"
    << "   Valor esperado : " << 1.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
    << "Exemplo:\n"
    << "P(x) = 4 - 2x + 3x^2 - 5x^3 + x^4\n"
    << "P(1.0) = 4 - 2*1 + 3*1^2 - 5*1^3 + 1^4 = 1.0\n"
    << "-------------------------------------------------------\n";

  // Avalia P(x = -1)
  x = -1.0;
  ASSERT_EQ(15, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(x)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      x = " << x << "\n"
    << "   Valor esperado : " << 15.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
```

```
    << "Exemplo:\n"
    << "P(x) = 4 - 2x + 3x^2 - 5x^3 + x^4\n"
    << "P(-1.0) = 4 - 2*(-1) + 3*(-1)^2 - 5*(-1)^3 + (-1)^4 = 15.0\n"
    << "-------------------------------------------------------\n";
}


TEST_F(Teste, Avaliar_polinomios_com_alguns_coeficientes_nulos) {
  float coef_p[] = {5, 0, 3, 0, 2};  // P(x) = 5 + 3x^2 + 2x^4.
  Polinomio p(5, coef_p);

  // Avalia P(x = 0).
  float x = 0.0;
  ASSERT_EQ(5, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(x)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      x = " << x << "\n"
    << "   Valor esperado : " << 5.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
    << "Exemplo:\n"
    << "P(x) = 5 + 3x^2 + 2x^4\n"
    << "P(0.0) = 5 + 3*0^2 + 2*0^4 = 5.0\n"
    << "-------------------------------------------------------\n";

  // Avalia P(x = 1)
  x = 1.0;
  ASSERT_EQ(10, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(x)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      x = " << x << "\n"
    << "   Valor esperado : " << 10.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
    << "Exemplo:\n"
    << "P(x) = 5 + 3x^2 + 2x^4\n"
    << "P(1.0) = 5 + 3*1^2 + 2*1^4 = 10.0\n"
    << "-------------------------------------------------------\n";

  // Avalia P(x = -1)
  x = -1.0;
```

```
  ASSERT_EQ(10, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(x)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      x = " << x << "\n"
    << "   Valor esperado : " << 10.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
    << "Exemplo:\n"
    << "P(x) = 5 + 3x^2 + 2x^4\n"
    << "P(-1.0) = 5 + 3*(-1)^2 + 2*(-1)^4 = 10.0\n"
    << "-------------------------------------------------------\n";
}


TEST_F(Teste, Avaliar_polinomios_nas_raizes) {
  float coef_p[] = {6, -5, 1};  // P(x) = 6 - 5x + x^2.
  Polinomio p(3, coef_p);

  // Avalia P(x = 2).
  float x = 2.0;
  ASSERT_EQ(0, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(x)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      x = " << x << "\n"
    << "   Valor esperado : " << 0.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
    << "Exemplo:\n"
    << "P(x) = 6 - 5x + x^2\n"
    << "P(2.0) = 6 - 5*2 + 2^2 = 0.0\n"
    << "-------------------------------------------------------\n";

  // Avalia P(x = 3)
  x = 3.0;
  ASSERT_EQ(0, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(x)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      x = " << x << "\n"
```

```
    << "   Valor esperado : " << 0.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
    << "Exemplo:\n"
    << "P(x) = 6 - 5x + x^2\n"
    << "P(3.0) = 6 - 5*3 + 3^2 = 0.0\n"
    << "-------------------------------------------------------\n";
}

// TODO(tfn): Padronizar mensagem de erro.
TEST_F(Teste, Avaliar_polinomios_constantes) {
  float coef_p[] = {7};  // P(x) = 7.
  Polinomio p(1, coef_p);
  // Avalia P(x = 0).
  float x = 0.0;
  ASSERT_EQ(7, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(0.0)\n"
    << "   Valor esperado : " << 7.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
    << "Exemplo:\n"
    << "P(x) = 7.0\n"
    << "P(0.0) = 7.0\n"
    << "-------------------------------------------------------\n";

  // Avalia P(x = 13)
  x = 13.0;
  ASSERT_EQ(7, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "p.Avaliar(0.0)\n"
    << "   Valor esperado : " << 7.0 << "\n"
    << "   Valor retornado: " << p.Avaliar(x) << "\n"
    << "Exemplo:\n"
    << "P(x) = 7.0\n"
    << "P(13.0) = 7.0\n"
    << "-------------------------------------------------------\n";

  // Avalia P(x = -13)
  x = -13.0;
  ASSERT_EQ(7, p.Avaliar(x))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "AvaliarPolinomio(0, {7}, 0.0)\n"
    << "   Valor esperado : " << 7.0 << "\n"
    << "   Valor retornado: " <<  p.Avaliar(x) << "\n"
```

```
    << "Exemplo:\n"
    << "P(x) = 7.0\n"
    << "P(-13.0) = 7.0\n"
    << "-------------------------------------------------------\n";
}

TEST_F(Teste, Atribuir_polinomio_de_grau_maior_que_um) {
  float coef_p[] = {5, 0, 3, 0, 2};
  Polinomio p(5, coef_p);

  float coef_esperado[] = {5, 0, 3, 0, 2};
  Polinomio esperado(5, coef_esperado);

  // Calcula a deriva de p.
  Polinomio novo;
  novo.Atribuir(p);
  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(novo))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "novo.Atribuir(p)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      'novo' eh o polinomio que deve ficar igual a p.\n"
    << "   Resultado esperado : novo == "
    << MostrarComoVetor(esperado) << "\n"
    << "   Resultado da funcao: novo == "
    << MostrarComoVetor(novo) << "\n"
    << "Exemplo:\n"
    << "   P(x) = " << MostrarPorExtenso(p) << "\n"
    << "   Q(x) = P(X) = " << MostrarPorExtenso(esperado) << "\n"
    << "   Sua funcao retornou o polinomio: "
    << MostrarPorExtenso(novo) << "\n"
    << "-------------------------------------------------------\n";
}

TEST_F(Teste, Somar_polinomios_com_coeficientes_nao_nulos) {
  float coef_p[] = {1, 2, 3, 4};
  Polinomio p(4, coef_p);

  float coef_q[] = {4, 3, 2, 1};
  Polinomio q(4, coef_q);

  float coef_esperado[] = {5, 5, 5, 5};
  Polinomio esperado(4, coef_esperado);
```

```
  // Soma p1 e p2.
  Polinomio soma;
  soma.Somar(p, q);
  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(soma))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "soma.Somar(p, q)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      q = " << MostrarComoVetor(q) << "\n"
    << "      'soma' eh o polinomio que armazena a soma.\n"
    << " Resultado esperado :soma==" << MostrarComoVetor(esperado)
    << " Resultado da funcao:soma==" << MostrarComoVetor(soma)<<"\n"
    << "Exemplo:\n"
    << "   P(x) = " << MostrarPorExtenso(p) << "\n"
    << "   Q(x) = " << MostrarPorExtenso(q) << "\n"
    << "   P(x) + Q(x) = "
    << MostrarPorExtenso(esperado) << "\n"
    << "   Sua funcao retornou o polinomio: "
    << MostrarPorExtenso(soma) << "\n"
    << "-------------------------------------------------------\n";
}

TEST_F(Teste, SomarPolinomios_com_alguns_coeficientes_nulos) {
  float coef_p[] = {0, 2, 0, 2};
  Polinomio p(4, coef_p);
  float coef_q[] = {1, 0, 3, 2};
  Polinomio q(4, coef_q);
  float coef_esperado[] = {1, 2, 3, 4};
  Polinomio esperado(4, coef_esperado);
  // Soma p1 e p2.
  Polinomio soma;
  soma.Somar(p, q);
  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(soma))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "soma.Somar(p, q)\n"
    << "   onde:\n"
    << "      p = " << MostrarComoVetor(p) << "\n"
    << "      q = " << MostrarComoVetor(q) << "\n"
    << "      'soma' eh o polinomio que armazena a soma.\n"
    << " Resultado esperado:soma==" << MostrarComoVetor(esperado)
    << " Resultado da funcao:soma==" << MostrarComoVetor(soma)<<"\n"
```

```
    << "Exemplo:\n"
    << "   P(x) = " << MostrarPorExtenso(p) << "\n"
    << "   Q(x) = " << MostrarPorExtenso(q) << "\n"
    << "   P(x) + Q(x) = "
    << MostrarPorExtenso(esperado) << "\n"
    << "   Sua funcao retornou o polinomio: "
    << MostrarPorExtenso(soma) << "\n"
    << "-------------------------------------------------------\n";
}

TEST_F(Teste, Derivar_polinomio_de_grau_um) {
  float coef_p[] = {1, 2};
  Polinomio p(2, coef_p);
  float coef_esperado[] = {2};
  Polinomio esperado(1, coef_esperado);
  // Calcula a deriva de p.
  Polinomio derivada;
  derivada.Derivar(p);
  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(derivada))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "derivada.Derivar(p)\n"
    << "   onde:\n"
    << " p = " << MostrarComoVetor(p) << "\n"
    << " 'derivada' eh o polinomio que armazena a derivada de p.\n"
    << "   Resultado esperado : derivada == "
    << MostrarComoVetor(esperado) << "\n"
    << "   Resultado da funcao: derivada == "
    << MostrarComoVetor(derivada) << "\n"
    << "Exemplo:\n"
    << "   P(x) = " << MostrarPorExtenso(p) << "\n"
    << "   P'(x) = " << MostrarPorExtenso(esperado) << "\n"
    << "   Sua funcao retornou o polinomio: "
    << MostrarPorExtenso(derivada) << "\n"
    << "-------------------------------------------------------\n";
}

TEST_F(Teste, Derivar_polinomio_de_grau_maior_que_um) {
  float coef_p[] = {5, 0, 3, 0, 2};
  Polinomio p(5, coef_p);
  float coef_esperado[] = {0, 6, 0, 8};
  Polinomio esperado(4, coef_esperado);
  // Calcula a deriva de p.
  Polinomio derivada;
  derivada.Derivar(p);
```

```
  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(derivada))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "derivada.Derivar(p)\n"
    << "   onde:\n"
    << "  p = " << MostrarComoVetor(p) << "\n"
    << " 'derivada' eh o polinomio que armazena a derivada de p.\n"
    << "   Resultado esperado : derivada == "
    << MostrarComoVetor(esperado) << "\n"
    << "   Resultado da funcao: derivada == "
    << MostrarComoVetor(derivada) << "\n"
    << "Exemplo:\n"
    << "   P(x) = " << MostrarPorExtenso(p) << "\n"
    << "   P'(x) = " << MostrarPorExtenso(esperado) << "\n"
    << "   Sua funcao retornou o polinomio: "
    << MostrarPorExtenso(derivada) << "\n"
    << "-------------------------------------------------------\n";
}

TEST_F(Teste, Integrar_polinomio_de_grau_zero) {
  float coef_p[] = {7};
  Polinomio p(1, coef_p);
  float coef_esperado[] = {0, 7};
  Polinomio esperado(2, coef_esperado);

  // Calcula a integral de p.
  Polinomio integral;
  integral.Integrar(p);
  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(integral))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "integral.Integrar(p)\n"
    << "   onde:\n"
    << "  p = " << MostrarComoVetor(p) << "\n"
    << " 'integral' eh o polinomio que armazena a integral de p.\n"
    << "   Resultado esperado : integral == "
    << MostrarComoVetor(esperado) << "\n"
    << "   Resultado da funcao: integral == "
    << MostrarComoVetor(integral) << "\n"
    << "Exemplo:\n"
    << "   P(x) = " << MostrarPorExtenso(p) << "\n"
    << "   integral[P](x) = " << MostrarPorExtenso(esperado) << "\n"
    << "   Sua funcao retornou o polinomio: "
    << MostrarPorExtenso(integral) << "\n"
    << "-------------------------------------------------------\n";
}
```

```
TEST_F(Teste, Integrar_polinomio_de_grau_um) {
  float coef_p[] = {1, 2};
  Polinomio p(2, coef_p);
  float coef_esperado[] = {0, 1, 1};
  Polinomio esperado(3, coef_esperado);

  // Calcula a integral de p.
  Polinomio integral;
  integral.Integrar(p);
  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(integral))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "integral.Integrar(p)\n"
    << "   onde:\n"
    << "   p = " << MostrarComoVetor(p) << "\n"
    << "  'integral' eh o polinomio que armazena a integral de p.\n"
    << "   Resultado esperado : integral == "
    << MostrarComoVetor(esperado) << "\n"
    << "   Resultado da funcao: integral == "
    << MostrarComoVetor(integral) << "\n"
    << "Exemplo:\n"
    << " P(x) = " << MostrarPorExtenso(p) << "\n"
    << " integral[P](x) = " << MostrarPorExtenso(esperado) << "\n"
    << "   Sua funcao retornou o polinomio: "
    << MostrarPorExtenso(integral) << "\n"
    << "-------------------------------------------------------\n";
}

TEST_F(Teste, Integrar_polinomio_de_grau_maior_que_um) {
  float coef_p[] = {5, 0, 3, 0, 2};
  Polinomio p(5, coef_p);
  float coef_esperado[] = {0, 5, 0, 1, 0, 0.4};
  Polinomio esperado(6, coef_esperado);

  // Calcula a integral de p.
 Polinomio integral;
  integral.Integrar(p);
  ASSERT_EQ(MostrarComoVetor(esperado), MostrarComoVetor(integral))
    << "-------------------------------------------------------\n"
    << "Erro na chamada da funcao: "
    << "integral.Integrar(p)\n"
    << "   onde:\n"
    << "   p = " << MostrarComoVetor(p) << "\n"
    << "  'integral' eh o polinomio que armazena a integral de p.\n"
```

```
<< "   Resultado esperado : integral == "
    << MostrarComoVetor(esperado) << "\n"
    << "   Resultado da funcao: integral == "
    << MostrarComoVetor(integral) << "\n"
    << "Exemplo:\n"
    << "   P(x) = " << MostrarPorExtenso(p) << "\n"
    << "   integral[P](x) = " << MostrarPorExtenso(esperado) << "\n"
    << "   Sua funcao retornou o polinomio: "
    << MostrarPorExtenso(integral) << "\n"
    << "-------------------------------------------------------\n";
}
} // End namespace

#endif  // TEST_H_
```