Especificação do Trabalho Final de MLP

Prof. Lucas Mello Schnorr (INF/UFRGS)

schnorr@inf.ufrgs.br

1 Sumário

1.1 Geral

Item	Descrição
Grupos	Três alunos
Linguagem	Uma dentre aquelas listadas
Problema	Um dentre aqueles sugeridos
Implementação	Duas: OO e funcional
Relatório	Um com os requisitos obrigatórios (`latex` e `bibtex`) – Modelo
Submissão	ZIP ou TAR.GZ, por e-mail, com FileSender da RNP

1.2 Prazos

Os prazos são até às 23:59 do dia informado.

Item	Prazo
Escolha da LP, Problema, Grupos	2017-09-25
Projeto inicial	2017-10-30
Entrega **parcial** (OO ou funcional)	2017-12-01
Entrega **final**	2018-01-01
Apresentações Sessão #1	2018-01-15
Apresentações Sessão #2	2018-01-17

2 Visão Geral

O objetivo deste trabalho consiste em fornecer aos alunos a oportunidade de **estudar uma linguagem de programação moderna com características híbridas** (i.e., multiparadigma). O trabalho permitirá aos alunos **demonstrarem que aprenderam os princípios de programação relacionados com os diferentes paradigmas** estudados ao longo do semestre, demonstrando, ainda, a **capacidade de analisar e avaliar linguagens de programação, seguindo os critérios abordados em aula**.

Cada grupo **deve selecionar uma linguagem de programação da lista abaixo**. Cada linguagem poderá ser escolhida por no máximo dois grupos. A ordem de preferência da escolha da linguagem será baseada no recebimento de e-mails pelo professor, que informará através de meio hábil qual o mapeamento grupo e linguagem. A escolha do grupo deve ser realizada dentro do prazo estabelecido.

C++ 17	ISO-CPP
C# 7.1	C#@Microsoft
Clojure	`http://clojure.org/`
F#	`http://fsharp.org/`
Groovy	`http://www.groovy-lang.org/`
Java 8	Java8@java.com
Julia	`http://julialang.org/`
Objective-C	ObjC@Apple
Objective CAML	`http://ocaml.org/`
Python	`http://python.org/`
R	`https://www.r-project.org/`
Ruby	`http://ruby-lang.org/`
Scala	`http://www.scala-lang.org/`
Swift 3.0	`http://swift.org/`

A tarefa principal do trabalho consiste em experimentar e comparar as características e funcionalidades orientadas a objeto e funcionais da linguagem de programação escolhida. Após definir um grupo e selecionar uma linguagem, é necessário escolher um problema a ser solucionado com ela. O problema será, então, implementado **duas vezes na mesma linguagem**: uma delas usando somente Orientação a Objetos e a outra usando somente características funcionais. Interfaces (gráficas ou textuais) podem ser feitas em qualquer paradigma ou plataforma, podendo inclusive serem compartilhadas entre as duas versões, visto que o foco não está na interface do programa.

3 Definição dos grupos, do problema e da linguagem

Cada grupo é formato com três participantes e elege um líder. O líder informa ao professor, por e-mail, qual o problema escolhido, a linguagem, e os membros do grupo conforme instruções a seguir. Grupos com outra quantidade de membros só serão aceitos para fins de arrendondamento da turma. O e-mail deve ter o título `MLP: Grupo TF` e ter no corpo da mensagem em texto puro (sem HTML) somente os seguintes dados.

Linha	Descrição
1	Nome do grupo (seja criativo)
2	Nome completo do líder
3	Código do líder
4	Nome completo do membro #2
5	Código do membro #2
6	Nome completo do membro #3
7	Código do membro #3
8	Linguagem
9	Problema

4 Problemas disponíveis

4.1 War

A ideia é desenvolver um jogo de batalha por turnos estilo o jogo americano Risk (`http://en.wikipedia.org/wiki/Risk_(game)/`) ou a versão Brasileira War (`http://pt.wikipedia.org/wiki/War`). A fim de tornar o jogo menos complexo e menos demorado, sua versão pode envolver somente dois adversários (seja outro ser humano ou o computador). Preferencialmente, o jogo pode utilizar os tiles do OpenStreetMap para desenhar o mapa e posicionar exércitos.

4.2 MarioBrosAI

A ideia consiste em desenvolver um software (bot ou agente) que controle o personagem Mário Bros para o ambiente InfiniteMarioBros, utilizado na Mario AI Competition. O Mário deve coletar o maior número de moedas no menor espaço de tempo, sem morrer.

4.3 Starcraft

A ideia consiste em desenvolver um software (bot ou agente) que controle um exército para competir no jogo Starcraft Broodwar. Maiores detalhes em: `http://sscaitournament.com/` e `https://github.com/bwapi/bwapi`.

4.4 BatalhaNaval

O computador deve sortear uma configuração ao inicial do jogo, em que estarão colocados num tabuleiro de tamanho 15x15 os seguintes itens: 4 submarinos (2 casas), 3 navios (3 casas) e 5 minas (1 casa cada). Em cada jogada, o computador lê as coordenadas (linha e coluna) da casa em que o usuário quer atingir e indica o resultado, ou seja, se acertou na água ou em parte de um navio (navio inteiro se for uma mina). O jogo termina quando o usuário afundar toda a frota, ou quando indicar que não quer continuar a jogar. Ao invés de solicitar as coordenadas, você pode usar o mouse como entrada de dados.

4.5 TowerDefence

Neste tipo de jogo você precisa defender algum elemento ou posição na tela, normalmente em algum cenário composto de uma ou mais estradas ou caminhos que são percorridos por uma série de inimigos (por rounds). A cada round você tem um saldo a gastar em torres ou elementos de defesa (ou ainda em `upgrades`), que podem ser posicionados em locais fixos ou abertos ao longo do cenário. Esses elementos de defesa devem atacar os inimigos,

destruindo-os antes que cheguem ao alvo. Cada inimigo tem um poder de ataque, cura ou quantidade de vida específico, o qual diminui cada vez que recebe algum tiro de defesa. Cada vez que um inimigo é acertado ou morto, você ganha créditos. O jogo termina quando uma quantidade x de inimigos chega no objetivo ou quando seu ponto de defesa fica muito fraco. Maiores detalhes em: `http://en.wikipedia.org/wiki/Tower_defense/`.

4.6 Escopo

Ou seja, desenvolver um simulador capaz de aceitar definições de subprogramas e variáveis locais, utilizando uma pseudolinguagem simples. Com base nisso, demonstrar como ficaria sua pilha de chamadas (call-stack) e o conteúdo das variáveis locais a cada passo de execução.

4.7 Galáxias

Implementar um simulador de partículas, considerando forças físicas de repulsão e atração. Uma possibilidade é utilizar as leis gravitacionais para construir um simulador de órbitas para estrelas e planetas. Outra possibilidade é utilizar uma força elétrica de repulsão (todas as partículas com carga positiva, por exemplo), e forças de atração baseadas em molas. Deve-se ter cuidado com a escalabilidade do algoritmo utilizando, dando preferências para o algoritmo de Barnes-Hut. Um exemplo utilizando a linguagem C já está disponível em `http://github.com/schnorr/viva/tree/master/src/libtupi`, e pode ser utilizado como inspiração para o projeto.

4.8 Pessoal

No caso, o grupo deve encaminhar sua ideia ao professor, descrita em detalhes, que avaliará sua viabilidade.

5 Recursos Necessários (critérios mínimos)

O trabalho realizado **deve considerar os aspectos especificados nesta seção**, sendo um conjunto específico de recursos para a solução orientada a objetos e outro para a solução funcional. Caso um recurso não esteja disponível na linguagem, **explique e justifique** no relatório os motivos para ele não existir **e utilize um mecanismo alternativo**.

5.1 Requisitos de orientação a objetos

- Especificar e utilizar classes (utilitárias ou para representar as estruturas de dados utilizadas pelo programa).
- Fazer uso de encapsulamento e proteção dos atributos, com os devidos métodos de manipulação (setters/getters) ou propriedades de acesso, em especial com validação dos valores (parâmetros) para que estejam dentro do esperado ou gerem exceções caso contrário.
- Especificação e uso de construtores-padrão para a inicialização dos atributos e, sempre que possível, de construtores alternativos.
- Especificação e uso de destrutores (ou métodos de finalização), quando necessário.
- Organizar o código em espaços de nome diferenciados, conforme a função ou estrutura de cada classe ou módulo de programa.
- Usar mecanismo de herança, em especial com a especificação de pelo menos três níveis de hierarquia, sendo pelo menos um deles correspondente a uma classe abstrata, mais genérica, a ser implementada nas classes-filhas.
- Utilizar polimorfismo por inclusão (variável ou coleção genérica manipulando entidades de classes filhas, chamando métodos ou funções específicas correspondentes).
- Usar polimorfismo paramétrico
  - através da especificação de *algoritmo* (método ou função genérico) utilizando o recurso oferecido pela linguagem (i.e., generics, templates ou similar)
  - e da especificação de *estrutura de dados* genérica utilizando o recurso oferecido pela linguagem.
- Usar polimorfismo por sobrecarga (vale construtores alternativos).
- Especificar e usar delegates.

5.2 Recursos para a solução funcional

- Priorizar o uso de elementos imutáveis e funções puras (por exemplo, sempre precisar manipular listas, criar uma nova e não modificar a original, seja por recursão ou através de funções de ordem maior).
- Especificar e usar funções não nomeadas (ou lambda).
- Especificar e usar funções que usem currying.
- Especificar funções que utilizem pattern matching ao máximo, na sua definição.
- Especificar e usar funções de ordem superior (maior) criadas pelo programador.
- Usar funções de ordem maior prontas (p.ex., map, reduce, foldr/foldl ou similares).
- Especificar e usar funções como elementos de 1ª ordem.
- Usar recursão como mecanismo de iteração (pelo menos em funções de ordem superior que manipulem listas).

6 Relatório

O grupo deve apresentar um relatório técnico com os itens descritos abaixo. O relatório deve ser escrito utilizando a linguagem de marcação `LaTeX`. O modelo do relatório pode ser obtido aqui:

- `https://github.com/schnorr/mlpreport`

Segue a lista dos itens obrigatórios para o relatório:

1. Capa: com identificação do grupo, da linguagem e do problema escolhidos.
2. Visão geral da Linguagem: Apresentação da linguagem escolhida, descrevendo suas características, fundamentos, funcionalidades, benefícios e principais aplicações (inclusive com discussão de sua aplicabilidade em questões práticas).
3. Análise Crítica: uma análise crítica da linguagem estudada, envolvendo uma tabela com os critérios e propriedades estudados em aula (i.e. simplicidade, ortogonalidade, expressividade, adequabilidade e variedade de estruturas de controle, mecanismos de definição de tipos, suporte a abstração de dados e de processos, modelo de tipos, portabilidade, reusabilidade, suporte e documentação, tamanho de código, generalidade, eficiência e custo, e outros que o grupo achem convenientes), com notas/valores justificados (ilustrando com exemplos utilizados no código ou descrevendo situações que contariam como pontos favoráveis ou desfavoráveis para cada critério ou propriedade). Indicar qual paradigma foi mais adequado para resolver o problema e por que.
4. Conclusão: descrevendo as facilidades e dificuldades encontradas, benefícios, problemas e limitações da linguagem estudada.
5. Referências: todo material consultado, incluindo livros, artigos, páginas na Internet, etc., que tenha relação com o assunto. Elaborar a lista usando `bibtex`.

Não serão aceitos trabalhos com indícios de plágio (cópia integral ou parcial de outros trabalhos). Utilizar trechos e exemplos, mesmo que em forma de paráfrase, é permitido e estimulado, desde que a menção (citação) ao autor do original seja feita corretamente.

7 Boas práticas

Sugere-se uma lista de boas práticas para a execução deste trabalho.

- GIT: para gerenciar o desenvolvimento em grupo e manter um repositório único de código, permitindo não só gerenciar versões, mas também controlar a contribuição de cada participante.
- Máquina Virtual: para que você possa configurar todas as bibliotecas, plug-ins e componentes necessários para o desenvolvimento e a execução de seu software.

8 Etapas de Entrega

Todas as etapas de entrega deverão ser encaminhados até a data estipulada pelo professor por e-mail. As entregas devem ser realizadas através de um arquivo compactado (ZIP ou TAR.GZ), contendo o relatório (em PDF) e os códigos-fontes desenvolvidos (não incluir os códigos binários). Utilize o serviço FileSender da RNP para envio de arquivos grandes, mediante login utilizando o cartão do aluno da UFRGS.

8.1 Projeto Inicial

O **projeto inicial**, uma etapa obrigatória, deve vir acompanhada apenas da capa, introdução e da apresentação da linguagem escolhida e do problema. Sugere-se que uma estrutura completa do relatório já esteja igualmente presente.

8.2 Entrega Parcial

A **entrega parcial**, uma etapa obrigatória, deve vir acompanhada da implementação e relatório a respeito da solução utilizando um dos paradigmas (OO ou funcional), a critério do grupo. O professor utilizará esta oportunidade para formar um parecer rápido do relatório e da implementação; sugerindo ao grupo melhorias caso necessário.

9 Apresentação

A apresentação do trabalho prático será feita diante da turma e do professor nas aulas especificadas no cronograma da disciplina. Cada grupo terá 10 minutos para a apresentação. Dentro desse tempo, os alunos deverão: apresentar o problema, apresentar a linguagem escolhida para a implementação, apresentar quais foram as vantagens e desvantagens da abordagem OO e da funcional para a implementação da solução do problema e, por fim, fazer uma breve demonstração. O professor fará perguntas pontuais direcionadas para cada um dos membros do grupo. A apresentação faz parte da nota. Pontualidade também.

10 Avaliação

A avaliação geral do trabalho incluirá os seguintes critérios: desenvolvimento e detalhamento dos itens do relatório, aplicação dos conceitos de programação estudados, utilização correta dos recursos da linguagem escolhida, correção, legibilidade, confiabilidade e originalidade, uso de referências, formatação e estilo do texto. Outros aspectos de avaliação poderão ser incluídos a critério do professor. O peso deste trabalho corresponde ao valor especificado no plano da disciplina disponível na plataforma de apoio pedagógico.

Atenção: conforme instruções presentes no plano de ensino da disciplina, todas as etapas do trabalho devem ser cumpridas para que a sua nota de trabalho seja contabilizada!