Especificação do Trabalho Final de MLP - Turma B

Prof. Leandro Krug Wives

Outubro de 2020

1 Visão Geral

O **objetivo** deste trabalho consiste em **oportunizar aos alunos o aprofundamento dos conceitos e características abordados na disciplina**, demonstrando que aprenderam os princípios relacionados com os diferentes paradigmas de programação estudados ao longo do semestre, evidenciando, ainda, a capacidade de analisar e avaliar linguagens de programação, seguindo os critérios abordados em aula.

1.1 Resumo

Item	**Descrição**
Grupos	Dois alunos (da mesma turma)
Opções de trabalho	Escolher um tópico dentre os sugeridos na seção 2.1
Vídeo	Detalhes na seção 2.3
Relatório	Detalhes na seção 2.2
Workshop virtual	Detalhes na seção 2.4

1.2 Prazos

Item	**Data**
Anúncio do trabalho	22/10/2020
Definição dos grupos e tópicos	01/11/2020
Entrega de vídeos	16/11/2020
Workshop (virtual)	17 - 24/11/2020
Entrega do relatório final	24/11/2020

2 Detalhamento do Trabalho

O primeiro passo do trabalho consiste em definir um grupo, eleger um líder e definir o tópico do trabalho. Os alunos devem se organizar em grupos de 2 alunos (da mesma turma). Trabalhos individuais ou com 3 alunos devem ser evitados e só serão aceitos em casos especiais (converse com o professor se for o seu caso). Trabalhos com mais

do que 3 alunos não serão aceitos. Trabalhos com alunos de turmas diferentes também não serão aceitos.

Os assuntos (tópicos) disponíveis são os listados na seção 2.1. Cada tópico pode ser escolhido por no máximo 3 grupos. **Os líderes de grupo têm até o dia 01/11/2020 para registrar as informações de seu grupo** (i.e., nome do grupo, componentes e tópico escolhido) via Moodle, na ferramenta Wiki disponibilizada pelo professor.

Após a definição, **cada grupo deve realizar sua pesquisa e elaborar um relatório** (detalhes na seção 2.2) **e um vídeo que apresente uma visão geral do tópico escolhido** (detalhes na seção 2.3). **O vídeo será disponibilizado aos colegas, que farão uma avaliação durante o período do Workshop Virtual** (ver seção 2.4).

Finalmente, **após o workshop, considerando o feedback do professor e dos colegas, os grupos finalizarão e entregarão o relatório final**.

2.1 Opções de trabalho

Segue a lista de opções de trabalho a serem desenvolvidos, lembrando que serão aceitos no máximo três grupos por problema (confirme disponibilidade no Wiki do Moodle da disciplina):

- **Simulador de determinação de escopo (dinâmico versus estático)**: desenvolver um simulador capaz de aceitar definições de subprogramas e variáveis locais, utilizando uma pseudolinguagem simples, que permita a definição de variáveis e escopos em diferentes níveis (global, bloco, subprograma), com aninhamento. Com base nisso, demonstrar como ficaria sua pilha de chamadas (call-stack) e o conteúdo das variáveis locais a cada passo de execução.

- **Evolução da linguagem C++ (ISO C++ 11, 14, 17 e 20)** (`http://isocpp.org/std/the-standard`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades acrescentados nas últimas versões da linguagem (versões 11 até 20). Detalhar o que foi introduzido em cada versão a partir da 11 (inclusive), ilustrando como funcionam esses novos recursos e o que foi feito para eles funcionarem. Dar exemplos (concretos) de uso, explicar como a linguagem fica melhor (ou pior) com eles e contrastar com algum código que faça a mesma coisa em uma versão antiga da linguagem (anterior a 11). Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento) e de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem).

- **Evolução da linguagem C# (focando nas versões 6, 7, 8 e 9)** (`https://docs.microsoft.com/pt-br/dotnet/csharp/whats-new/csharp-9`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades acrescentados nas últimas versões da linguagem (versões 6 até 9). Detalhar o que foi introduzido em cada versão a partir da 6 (inclusive), ilustrando como funcionam esses novos recursos e o que foi feito para eles funcionarem. Dar exemplos (concretos) de uso, explicar como a linguagem fica melhor (ou pior) com eles e contrastar com algum código que faça a mesma coisa em uma versão antiga da linguagem (anterior a 6). Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento) e de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem).

- **Evolução da linguagem Java (ênfase nas versões 8 em diante, até a 15)** (`https://jdk.java.net/15/`, `www.oracle.com/technetwork/java/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades acrescentados nas últimas versões da linguagem (versões 8 até 15). Detalhar o que foi introduzido em cada versão a partir da 8 (inclusive), ilustrando como funcionam esses novos recursos e o que foi feito para eles funcionarem. Dar exemplos (concretos) de uso, explicar como a linguagem fica melhor (ou pior) com eles e contrastar com algum código que faça a mesma coisa em uma versão antiga da linguagem (anterior a 8). Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento) e de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem).

- **Swift e sua evolução (1, 2, 3, 4 e 5)** (`https://docs.swift.org/swift-book/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem Swift desde sua criação (versões 1 até 5.1). Detalhar o que foi modificado ou introduzido a cada versão, ilustrando como esses novos recursos. Dar exemplos (concretos) de uso, explicar como a linguagem fica melhor (ou pior) com eles. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento) e de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem).

- **Linguagem Clojure** (`http://clojure.org/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem. Dar exemplos (concretos) de uso, contrastando com alguma das linguagens vistas em aula. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento) e de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem). Incluir elementos de programação em lógica com Clojure.

- **Linguagem Erlang** (`http://www.erlang.org/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem. Dar exemplos (concretos) de uso, contrastando com alguma das linguagens vistas em aula. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento) e de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem).

- **Linguagem F#** (`http://fsharp.org/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem. Dar exemplos (concretos) de uso, contrastando com alguma das linguagens vistas em aula. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento), de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem) e de programação em lógica. Incluir interfaces gráficas e tratamento de eventos.

- **Linguagem Julia** (`http://julialang.org/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem. Dar exemplos (concretos) de uso, contrastando com alguma das linguagens vistas em aula. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento), de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem) e de programação em lógica.

- **Linguagem OCAML** (`http://ocaml.org/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem. Dar exemplos (concretos) de uso, contrastando com alguma das linguagens vistas em aula. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento) e de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem).

- **Linguagem Ruby (2.7.x)** (`http://ruby-lang.org/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem. Dar exemplos (concretos) de uso, contrastando com alguma das linguagens vistas em aula. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento), de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem) e de programação em lógica.

- **Linguagem Scala (2.12.x)** (`http://www.scala-lang.org/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem. Dar exemplos (concretos) de uso, contrastando com alguma das linguagens vistas em aula. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento), de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem) e de programação em lógica.

- **Linguagem Elixir** (`https://elixir-lang.org/`): estudar, compreender e apresentar os comandos, estruturas e funcionalidades da linguagem. Dar exemplos (concretos) de uso, contrastando com alguma das linguagens vistas em aula. Não esquecer de apresentar elementos de orientação a objetos (classes, herança simples/múltipla, encapsulamento), de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta-ordem) e de programação em lógica (dentro do possível).

- **Escolha de uma linguagem não listada ou problema de linguagem de programação motivador**. No caso, o grupo deve encaminhar sua proposta ao professor, descrita em detalhes, que avaliará sua viabilidade. A priori, qualquer linguagem moderna com características funcionais e orientadas a objetos pode ser relevante.

2.2 Relatório

O grupo deve apresentar um relatório técnico, em PDF, com os itens descritos abaixo. Sugere-se que este relatório seja escrito utilizando recursos do LaTeX. Um modelo de relatório (tanto em LaTeX quanto em .doc), entra-se no Moodle.

Segue a lista dos itens obrigatórios:

1. **Capa**. Com identificação do grupo e da linguagem escolhida.

2. **Visão geral da Linguagem ou problema escolhido**. Apresentação da linguagem escolhida ou do problema, descrevendo suas origens e inspirações, principais características, fundamentos, funcionalidades, benefícios e principais aplicações.

3. **Detalhamento da linguagem ou problema**. Demonstração dos elementos estudados ou desenvolvidos, conforme o tópico escolhido. Elaborar um exemplo para cada elemento e explicá-lo, traçando comentários positivos ou negativos (usando como base os critérios de avaliação de linguagens de programação discutidos nas primeiras aulas).

4. **Análise Crítica**. Uma análise crítica da linguagem estudada ou usada na resolução do problema escolhido. A análise compreende 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 justificadas e comentadas com exemplos ou situações que contariam como pontos favoráveis ou desfavoráveis para cada critério ou propriedade).

5. **Conclusões**. Conclusões, descrevendo as facilidades e dificuldades encontradas, benefícios, problemas e limitações da linguagem estudada.

6. **Bibliografia**. Todo material consultado ou não, incluindo livros, artigos, páginas na Internet, etc., que tenha relação com o assunto. Elaborar a lista preferencialmente usando *Bibtex*.

O PDF resultante do relatório será entregue via Moodle, em tarefa específica, disponível no início da página. O relatório deverá ser entregue até o dia 24/11/2020. Todo atraso implica em desconto de nota.

2.3 Vídeo

Os grupos deverão elaborar um vídeo apresentando a linguagem ou problema estudado (abordando os itens relativos ao tópico escolhido). O vídeo deve ter entre 5 e 10 minutos. Sugere-se utilizar o máximo de recursos o possível (screen cast, vídeo-aula, entrevistas, etc).

O vídeo será encaminhado no prazo estipulado (16/11/2020) via ferramenta a ser disponibilizada pelo professor. Todo atraso implica em desconto de nota.

Os vídeos serão disponibilizados aos colegas, para que possam assistir durante o período do Workshop Virtual, onde farão comentários e sugestões.

2.4 Workshop

Os vídeos dos trabalhos serão disponibilizados para toda a turma no período do Workshop Virtual, para que os colegas possam assistir e comentar. Não haverá atividade presencial, mas o horário de aula poderá ser utilizado para assistir aos vídeos e comentá-los, além elaborar o relatório final e tirar eventuais dúvidas com o professor.

A elaboração de comentários, sugestões e perguntas faz parte da nota.

Os grupos deverão observar esses comentários, sugestões e perguntas a fim de aprimorar seus relatórios.

3 Avaliação e Prazos

A avaliação 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, ortografia, gramática 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.

O peso do trabalho trabalho prático na composição da notal final corresponde ao valor especificado no plano da disciplina disponível na plataforma de apoio pedagógico.

O prazos de cada etapa ocorrem conforme especificado e serão definidos em respectivas atividades disponibilizadas no Moodle, sendo rigorosamente observados. Qualquer atraso implicará em perda de nota.

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!
- 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.

4 Boas Práticas

Sugere-se uma lista de boas práticas para a execução deste trabalho. Elas são opcionais. Caso o grupo tenha interesse em utilizá-las, o professor pode auxiliar no seu uso durante o semestre.

Trello - para manter o registro de cada atividade e seus responsáveis.

Editor online de documentos - para que todos possam criar e editar de maneira colaborativa o relatório final (sugere-se o uso do Overleaf (`http://www.overleaf.com/`).

Github ou Bitbucket - 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, inclusive no relatório.

Máquina Virtual ou Docker - para que você possa configurar todas as bibliotecas, plug-ins e componentes necessários para o desenvolvimento e a execução de seu software, durante o estudo da linguagem.

Jupyter Notebook - inicialmente desenvolvida para Python, é uma ferramenta para a criação e edição de blocos de nota colaborativos, com código-fonte e execução, intercalados com texto. Atualmente aceita várias linguagens, inclusive C++, Java e outras.

5 Dúvidas

Em caso de dúvidas, não hesite em consultar o professor.