Especificação do Trabalho Final de MLP

Prof. Leandro Krug Wives

Turma U - 2025/I

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 das fases, entregáveis e prazos

Fase	**Descrição**	**Prazo**
1.Definição	(a) Definir quem faz parte do grupo (três alunos) e (b) escolher um tópico dentre os sugeridos na Seção 2.1	18/05/2025
2.Desenvolvimento	Execução do trabalho, que consiste em estudar, conduzir a pesquisa, desenvolver ou planejar exemplos, além de elaborar os elementos do trabalho (conforme o tópico escolhido)	15/06/2025
3. Relatório	Detalhes na Seção 2.2	15/06/2025
4.Participação no Seminário Final	Detalhes na Seção 2.3	16/06/2025 - 23/06/2025

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 3 alunos. Trabalhos individuais, em duplas ou com 4 alunos devem ser evitados e só serão aceitos em casos especiais – converse

com o professor se for o seu caso. Trabalhos com mais de 4 alunos não serão aceitos.

Os assuntos (tópicos) disponíveis são os listados na seção 2.1. **Os líderes de grupo têm até o dia definido para a Tarefa 1 (ver seção 1.1) para registrar as informações de seu grupo** (i.e., nome do grupo, componentes e tópico escolhido) via Moodle, nas atividades específicas disponibilizadas pelo professor.

Após a definição, cada grupo deve realizar sua pesquisa e/ou implementação e elaborar um relatório (detalhes na seção 2.2). **A apresentação será realizada em um *seminário (workshop)* no final do semestre, e todos farão avaliação dos trabalhos dos colegas durante o período do seminário** (ver seção 2.3).

2.1 Opções de trabalho

Segue a lista de opções de trabalho a serem desenvolvidos, os quais estão **organizados entre trabalhos teóricos (de estudo de linguagens)** e **práticos (de implementação)**.

É importante mencionar que os trabalhos relacionados com o estudo de uma linguagem só podem ser escolhidos por um único grupo (o que escolher primeiro). Os demais (i.e., de implementação) podem ser escolhidos por no máximo dois grupos (por ordem de escolha).

1. **Estudos de Linguagem**

   Nesta opção de trabalho, os grupos devem escolher uma das linguagens da lista a seguir. O trabalho consiste em estudar a linguagem escolhida, na versão mais recente possível, procurando compreender e apresentar os principais comandos, estruturas e funcionalidades da linguagem, dando exemplos de uso. Deve também exemplificar como a linguagem lida com elementos de orientação a Objetos (definição de classes, encapsulamento e proteção; herança simples, herança múltipla ou soluções equivalentes, como interfaces ou *mixins*; composição; organização de código em pacotes, módulos ou elementos equivalentes; polimorfismo *ad-hoc* e universal; *delegates*, se houver) de programação funcional (funções anônimas com lambdas, imutabilidade, recursão, funções de alta ordem e *pattern-matching*) e, se houver, de programação em lógica (definição de fatos, regras e realização de inferências/consultas). Incluir detalhes sobre o sistema de tipos utilizado pela linguagem (i.e., indicar se é estático ou dinâmico, se há inferência de tipos, se tem anotação de tipos, entre outros elementos relevantes), como lida com escopo (estático ou dinâmico) e sobre o sistema de avaliação implementado (vin-

culação profunda ou rasa). Descrever a estrutura de memória e, em especial, se tem algum mecanismo de gestão automática de memória (*garbage collection*).

Lista de linguagens possíveis:

- **C++ (padrão ISO)**[1].
- **C#**[2].
- **Clojure**[3]: Incluir elementos de programação em lógica.
- **F#**[4]: Incluir interfaces gráficas e tratamento de eventos.
- **Java**[5].
- **Javascript**(ECMAScript)[6].
- **Julia**[7]: Incluir elementos de programação em lógica e avaliar sua importância (com exemplos) para *Data Science*.
- **Kotlin**[8].
- **Swift**[9].
- **OCAML**[10].
- **PHP**[11]
- **Ruby**[12].
- **Rust**[13].
- **Scala**[14].
- **Escolha de uma linguagem não listada ou problema de linguagem de programação motivador**. No caso, o grupo

[1] http://isocpp.org/std/the-standard
[2] https://learn.microsoft.com/pt-br/dotnet/csharp/whats-new/csharp-version-history
[3] http://clojure.org/
[4] http://fsharp.org/
[5] https://jdk.java.net/
[6] https://ecma-international.org/publications-and-standards/standards/ecma-262/
[7] http://julialang.org/
[8] https://kotlinlang.org/docs/books.html
[9] https://docs.swift.org/swift-book/
[10] http://ocaml.org/
[11] https://www.php.net/
[12] http://ruby-lang.org/
[13] https://www.rust-lang.org/learn
[14] http://www.scala-lang.org/

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. **Estudos Práticos (implementações)**

   - **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.
   - **Desenvolvimento comparativo de software em dois paradigmas**: escolher uma linguagem que suporte dois paradigmas (obrigatoriamente o funcional e o Orientado a Objetos) ou duas linguagens diferentes (uma de cada paradigma). Você deve implementar o mesmo software (escolha um dos problemas da lista seguinte), mas usando paradigmas diferentes (da maneira mais pura que a(s) linguagem(ns) escolhida(s) suportar(em)). O objetivo é demonstrar como as funcionalidades orientadas a objeto e funcionais das linguagens escolhidas podem ser usadas para a solução do problema, destacando as características (funcionais e O.O) que auxiliam (ou não) na resolução do problema. O seu relatório deve incluir uma descrição das implementações, dando ênfase aos elementos que foram utilizados em cada paradigma para solucionar o problema. Faça uma análise comparativa detalhada sobre esses elementos e indique o que contribuiu ou dificultou a implementação. Para a versão Orientada a Objetos, você deve utilizar, pelo menos, classes, herança, polimorfismo e encapsulamento. Para a versão funcional, deve usar, pelo menos, funções anônimas com lambdas, imutabilidade, recursão e funções de alta-ordem.
     - **Jogo de torre de defesa (***tower defense***)**: O objetivo deste tipo de jogo é tentar parar os inimigos de cruzar um mapa através da construção de prédios-armadilhas e torres que atiram projéteis cinéticos. As construções devem ser feitas de forma a diminuir o avanço dos inimigos. Mais informações sobre este tipo de jogo podem ser obtidas na Wikipedia (`http://en.wikipedia.org/wiki/Tower_defense/`).

- **Simulador de galáxias ou de partículas**: 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 a modelagem.
- **Jogos de artilharia em duas (ou três) dimensões**: Nestes jogos, o combate é feito por turnos onde o objetivo de cada jogador é destruir o seu adversário através do planejamento de uma trajetória correta para um projétil cinético. Técnicas sofisticadas de cálculo balístico, envolvendo força, vento, etc, podem ser empregadas para facilitar ou dificultar o jogo. Mais informações (`https://en.wikipedia.org/wiki/Artillery_game`).
- **Jogo do tipo War**: Segundo a Wikipedia (`http://pt.wikipedia.org/wiki/War`): *O jogo é disputado com um mapa do mundo dividido em 6 regiões (Europa, Ásia, África, América do Norte, América do Sul e Oceania). Cada jogador recebe uma carta com um determinado objetivo e quem completar primeiro o seu e declará-lo cumprido é o vencedor. É disputado em rodadas, nas quais os participantes colocam exércitos [...] e atacam outros oponentes.*
- **Outro problema não listado**: discuta primeiro com o professor...

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), encontra-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). Incluir principais comandos da linguagem, além de exemplos de elementos de Orientação a Objetos (Classes, atributos, herança, polimorfismo de diferentes tipos...) e de programação funcional (funções anônimas, funções de primeira ordem, funções de alta ordem, funções puras, recursão para realizar iteração...). Incluir evolução da linguagem e elementos que foram sendo acrescentados, modificados. Procurar analisar e criticar, analisar essas inclusões ou remoções. Se escolher como tópico o desenvolvimento de algum problema, descrever o que foi implementado, a arquitetura do software, principais funções e algoritmos. Descrever elementos importantes e contrastá-los (no caso de soluções em múltiplos paradigmas).

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 à 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 achar conveniente), 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*.

   Recomenda-se mostrar o relatório ao professor antes da entrega, para *feedback*.

   Se tiver dúvidas, fale com o professor!

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 estabelecido na atividade (ver tabela no início deste documento).

A entrega no prazo é importante porque, além do professor, os colegas avaliarão o seu conteúdo. Essa avaliação ocorrerá durante o período do seminário.

Todo atraso implica em desconto de nota.

2.2.1 Forma da Entrega

A entrega do trabalho é realizada através de um arquivo compactado (zip ou tar.gz), contendo o relatório (em PDF) e os códigos-fontes desenvolvidos (não incluir os executáveis) e Makefiles (se necessário), com um arquivo texto explicando como compilar/lançar o código-fonte. Alternativamente, os grupos podem disponibilizar o código-fonte em um repositório de código (p.ex. github ou similar) e referenciar o link no relatório.

Tal arquivo compactado deverá ser encaminhado até a data estipulada pelo professor e indicada em atividade específica de entrega via Moodle.

2.3 Seminário

No final do semestre, teremos um seminário de apresentação dos trabalhos. Durante o período do seminário, os grupos apresentarão (de forma resumida) o resultado de seus trabalhos e avaliarão os trabalhos dos colegas.

2.3.1 Apresentação

Em dias definidos pelo professor (em cronograma a ser disponibilizado no Moodle), os grupos apresentarão os resultados de seus trabalhos para o professor e para os colegas.

Para tanto, devem elaborar uma apresentação sobre a linguagem escolhida ou demonstrar o software desenvolvido. As apresentações serão de, no máximo, 20 minutos. Portanto, é importante focar nos elementos mais importantes.

Todos os participantes do grupo devem participar da apresentação. Sugere-se que o tempo seja dividido entre os apresentadores.

A realização das apresentações, assim como assisti-las, elaborar comentários, tecer sugestões e fazer perguntas, faz parte da nota.

2.3.2 Avaliação dos trabalhos dos outros grupos

Durante o seminário estará disponível no Moodle uma atividade de avaliação, onde os trabalhos dos grupos (relatório e apresentação) serão avaliados pelos colegas dos outros grupos, tendo como base um formulário de avaliação com critérios definidos pelo professor. A atividade em si de avaliar e o resultado global da avaliação farão parte da nota final do trabalho.

3 Nota final 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 prático na composição da nota final corresponde ao valor especificado no plano da disciplina disponível na plataforma de apoio pedagógico.

Os 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.

Todos os participantes do trabalho devem se envolver, participar de todas as atividades para que a nota do trabalho seja integralizada na média final.

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.