Ministério da Educação
Universidade Federal de Itajubá - Campus Itabira
Rua Irmã Ivone Drummond, 200, Distrito Industrial II
Itabira - MG - 35903-087

Instituto de Ciências Tecnológicas - ICT
Engenharia de Computação
ECOI26 - Compiladores
Prof. Walter Aoiama Nagai

Projeto Final: TermIA - Terminal Inteligente

Data de entrega parcial: **25/10** (sábado) **às 23h59**

Data de entrega final: 29/11 (sábado) **às 23h59**

Data da apresentação: 01/12 (segunda-feira) **às 15h45**

1. Visão Geral

Este projeto integra os conceitos principais da disciplina de **Compiladores** com práticas de **engenharia de software** e **IA aplicada**. O estudante deverá projetar e implementar um **shell (terminal) inteligente** denominado **`TermIA`**, capaz de interpretar comandos definidos pelo usuário, executar ações no sistema operacional e oferecer suporte básico a comandos de Inteligência Artificial.

2. Resultados de Aprendizagem Esperados

Ao concluir o projeto, o aluno será capaz de:

1. **Modelar gramáticas** simples para descrever comandos de terminal.
2. **Construir analisadores léxicos e sintáticos** funcionais para interpretar esses comandos.
3. **Executar comandos do sistema operacional** de forma controlada e segura.
4. **Integrar recursos de IA** via API REST para oferecer funcionalidades como ajuda, autocompletar ou geração de respostas simples.
5. **Documentar e apresentar** um software funcional, justificando as escolhas técnicas.

2.1 Evidências de Aprendizagem

- **Projeto funcional** (*shell* executável) contendo:
    - Lexer e parser para comandos definidos pelo aluno.
    - Mecanismo de execução de comandos de SO, pelo menos, três comandos de SO.
    - Pelo menos, três recursos de IA integrado, por exemplo:
        - **`ia ask`** “`Qual é a capital da França?`”
        - **`ia summarize`** “`Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cities of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.`”
        - `ia codeexplain “arquivo.txt”`
        - Podem ser outros comandos de IA, que devem estar descritos na documentação.
    - Recursos extras (histórico, autocomplete, mensagens de erro claras).
- **Repositório Git documentado**, incluindo README e gramática usada.
- **Demonstração prática** ao final do curso, apresentando a arquitetura e funcionalidades do shell.

2.2 Atividades e Caminho de Aprendizagem (8 Semanas)

Semana	Tópico	Produto Intermediário
1	Fundamentos de compiladores e linguagens formais aplicados a CLIs; definição da gramática	Gramática inicial
2	Implementação do analisador léxico (tokens)	Lexer funcional
3	Implementação do analisador sintático (parser)	Parser integrado
4	Conexão com SO (execução de comandos reais)	Shell básico
5	Integração com API de IA (REST simples)	Comando `ia <prompt>`
6	Recursos inteligentes adicionais: histórico, autocomplete, ajuda contextual	Shell “smart” inicial
7	Testes, refino, UX e documentação preliminar	Beta funcional
8	Apresentação e entrega final	Shell final e documentação

2.3 Entregáveis

Entrega parcial: Gramática inicial, Lexer funcional e parser integrado respectivos as Semanas 1, 2 e 3
Entrega final: os demais produtos intermediários.

2.4 Critérios de Avaliação

Critério	Excelente (A)	Bom (B)	Regular (C)	Insuficiente (D)
Lexer & Parser	Funcionam para todos os comandos definidos; bem documentados	Funcionam para casos principais	Funcionam parcialmente	Incompletos ou inexistentes
Execução de comandos do Sistema Operacional	Robusta e segura, cobre vários comandos	Funciona para comandos básicos	Funciona parcialmente	Não funciona
Integração IA	Comandos de IA úteis e estáveis	Integração mínima funcional	Respostas limitadas	Ausente
UX & extras	Histórico, autocomplete e mensagens claras	Alguns recursos extras	Interface mínima	Pouca ou nenhuma UX
Documentação & Apresentação	Clara, organizada e explicando arquitetura	Suficiente, mas não detalhada	Parcial	Deficiente

3. *Product Requirements Document* (PRD)

3.1 Contexto

O **TermIA** é um projeto integrador que tem como meta aplicar os conceitos da disciplina de Compiladores para desenvolver um terminal personalizado. O sistema deve ser capaz de interpretar comandos definidos pelo usuário, executar tarefas reais do sistema operacional e incluir funcionalidades básicas de Inteligência Artificial para apoio e automação.

3.2 Objetivos do Produto

- **Objetivo Geral** Capacitar o aluno a empregar conceitos de análise léxica e sintática, bem como integração de serviços externos, para criar um shell funcional e inteligente.
- **Objetivos Específicos**
  - Criar gramáticas simples e expressivas para comandos de terminal.
  - Implementar analisadores léxicos e sintáticos customizados.
  - Permitir execução de comandos do sistema operacional.
  - Integrar recursos de IA via API REST (ex.: ajuda, sugestão de comandos, geração de texto).
  - Documentar arquitetura, decisões técnicas e instruções de uso.

3.3 Requisitos Funcionais

ID	Requisito	Descrição
RF-01	Lexer	Implementar um analisador léxico que reconheça tokens dos comandos definidos pelo aluno.
RF-02	Parser	Construir um analisador sintático que valide a gramática e crie uma estrutura interpretável (ex.: árvore sintática).
RF-03	Execução de comandos do sistema operacional	Permitir que comandos válidos executem tarefas reais do sistema (listar arquivos, criar diretórios etc.).
RF-04	Integração IA	Oferecer pelo menos um comando de IA, ex.: `ia <prompt>` ou `ia --help`, consumindo uma API externa.
RF-05	Histórico / UX	Armazenar e exibir histórico de comandos, oferecer mensagens de ajuda e erros compreensíveis.
RF-06	Documentação	Disponibilizar README explicando arquitetura, gramática, como executar e testar.

3.4 Requisitos Não Funcionais

- **Linguagem de programação:** Python (PLY) ou C/C++ (Flex/Bison); outras mediante aprovação.
- **Portabilidade:** funcionar em ambiente Linux ou WSL.
- **Segurança:** comandos executados devem ser restritos ao ambiente controlado, evitando ações destrutivas.
- **Versionamento:** uso obrigatório de Git (GitHub ou GitLab).
- **Código limpo:** padrão de nomes claro, comentários e modularização.

3.5 Recursos Recomendados

- **Compiladores:**
  - Aho, Lam, Sethi, Ullman — *Compilers: Principles, Techniques and Tools*
  - Grune et al. — *Modern Compiler Design*
- **Ferramentas:**
  - PLY ou Lark (Python)
  - Flex/Bison (C/C++)
  - ANTLR (Java)
- **IA:**
  - OpenAI GPT-4o-mini, Ollama local, APIs REST simples

3.6 Riscos e Mitigações

Risco	**Mitigação**
Dificuldade com ferramentas de análise	Fornecer exemplos práticos e tutoriais em aula
Integração de IA falhar por instabilidade de API	Permitir fallback local ou exemplo pré-configurado
Segurança ao executar comandos	Restringir comandos suportados e revisar exemplos