Plano Completo Minishell - Divisão em Dupla

- **luarodri**: Responsável pelo parsing, tokenização, expansão de variáveis
- **iwietzke**: Responsável pela execução, pipes, built-ins, redirecionamentos

ESTRUTURA INICIAL DO PROJETO (AMBOS)

1. Setup do Repositório

- Criar estrutura de pastas:

```
minishell/
├── src/
│   ├── parsing/
│   ├── execution/
│   ├── builtins/
│   └── utils/
├── include/
├── libft/
├── obj/
└── Makefile
```

- Copiar libft para o projeto
- Criar header principal `minishell.h`
- Definir estruturas de dados principais
- Configurar Makefile básico

2. Definição das Estruturas de Dados (AMBOS - Discussão)

- Estrutura para tokens
- Estrutura para comandos
- Estrutura para ambiente
- Estrutura para redirecionamentos
- Estrutura para pipelines
- Lista de variáveis globais permitidas

---

luarodri: PARSING E LEXICAL ANALYSIS

Phase 1: Lexer/Tokenizer

1. **Implementar função de leitura de input**
   - Integrar readline()

- Gerenciar histórico
- Tratamento de sinais básicos

2. **Tokenização**
   - Identificar tipos de tokens (comando, argumento, operador, redirecionamento)
   - Separar por espaços e operadores especiais
   - Tratar aspas simples e duplas
   - Expandir variáveis de ambiente ($VAR, $?)
3. **Validação sintática básica**
   - Verificar parênteses balanceados
   - Verificar aspas fechadas
   - Identificar erros de sintaxe básicos

Phase 2: Parser

4. **Construção da árvore de sintaxe**
   - Parsear comandos simples
   - Processar redirecionamentos (<, >, <<, >>)
   - Estruturar pipelines (|)
   - Identificar comandos built-in vs externos
5. **Expansão de variáveis**
   - Expandir $VAR para valores do ambiente
   - Expandir $? para exit status
   - Tratar expansão dentro de aspas duplas
   - Preservar literais em aspas simples
6. **Validação semântica**
   - Verificar redirecionamentos válidos
   - Validar estrutura de comandos
   - Detectar comandos não encontrados no PATH

Phase 3: Here-doc (<<)

7. **Implementar here-document**
   - Ler input até delimiter
   - Armazenar em arquivo temporário
   - Conectar ao stdin do comando
   - Limpar arquivos temporários

---

iwietzke: EXECUTION E BUILT-INS

Phase 1: Execução de Comandos

1. **Sistema de execução básico**
   - fork() para criar processos filhos
   - execve() para executar comandos
   - wait() para aguardar processos
   - Gerenciar exit status
2. **PATH resolution**
   - Buscar executáveis no PATH
   - Verificar se comando é executável
   - Tratar caminhos relativos e absolutos
   - Gerenciar erros de "command not found"
3. **Redirecionamentos**
   - Implementar < (input redirect)
   - Implementar > (output redirect)
   - Implementar >> (append mode)
   - Duplicar file descriptors com dup2()
   - Gerenciar abertura/fechamento de arquivos

Phase 2: Pipelines

4. **Sistema de pipes**
   - Criar pipes com pipe()
   - Conectar stdout de um comando ao stdin do próximo
   - Coordenar múltiplos processos em pipeline
   - Aguardar todos os processos da pipeline
5. **Gerenciamento de processos**
   - Controlar grupos de processos
   - Tratar sinais em pipelines
   - Gerenciar foreground/background

Phase 3: Built-in Commands

6. **Comandos internos obrigatórios**
   - `echo` (com opção -n)

- `cd` (apenas paths relativos/absolutos)
- `pwd` (sem opções)
- `export` (sem opções)
- `unset` (sem opções)
- `env` (sem opções ou argumentos)
- `exit` (sem opções)

7. **Gerenciamento de ambiente**
   - Manter tabela de variáveis de ambiente
   - Implementar export/unset
   - Herdar ambiente para processos filhos

INTEGRAÇÃO E FINALIZAÇÃO (AMBOS)

Phase 4: Signal Handling

8. **Tratamento de sinais**
   - ctrl-C (SIGINT): novo prompt
   - ctrl-D (EOF): sair do shell
   - ctrl-\ (SIGQUIT): ignorar
   - Usar apenas uma variável global para sinais

Phase 5: Modo Interativo

9. **Interface do shell**
   - Exibir prompt
   - Manter histórico de comandos
   - Modo interativo vs não-interativo
   - Limpeza adequada na saída

Phase 6: Error Handling

10. **Tratamento de erros robusto**
    - Memory leaks (exceto readline)
    - Erro de sintaxe
    - Comandos não encontrados
    - Permissões de arquivo
    - Recursos limitados (file descriptors)

Phase 7: Testing

11. **Testes extensivos**
    - Comandos simples
    - Comandos com argumentos
    - Redirecionamentos individuais
    - Pipelines simples e complexos
    - Built-ins
    - Casos edge (aspas, variáveis, sinais)
    - Comparar comportamento com bash

Phase 8: Code Review e Cleanup

12. **Finalização**
    - Code review entre dupla
    - Verificar Norma 42
    - Otimizar performance
    - Documentar código
    - Preparar defesa

CRONOGRAMA DETALHADO (8 SEMANAS) COM PRAZOS

Semana 1: Fundações (27/05 - 02/06)

Ambos:

- Setup completo do projeto (Dia 1-2)
- Definir structs principais (Dia 2)
- Configurar Makefile (Dia 2)
- Primeiro commit funcional (Dia 3)

luarodri:

- Implementar readline básico (Dia 3-4)
- Tokenizer simples (sem expansão) (Dia 4-5)
- Parsing de comandos únicos (Dia 6-7)

iwietzke:

- Fork/execve básico (Dia 3-4)
- Execução de comandos simples (Dia 5-6)

- Estrutura dos built-ins (Dia 6-7)

Prazo Entrega Phase 1 (Fundações): 02/06

Semana 2: Core Parsing e Execution (03/06 - 09/06)

luarodri:

- Tratar aspas simples/duplas (Dia 1-3)
- Expansão básica de variáveis (Dia 4-5)
- Parsing de redirecionamentos (Dia 6-7)

iwietzke:

- Implementar redirecionamentos básicos (Dia 1-3)
- Built-ins: echo, pwd, env (Dia 4-6)
- Error handling inicial (Dia 6-7)

Prazo Entrega Phase 2 (Core): 09/06

Semana 3: Pipes e Expansão Avançada (10/06 - 16/06)

luarodri:

- Parser completo com pipes (Dia 1-3)
- Expansão de $? (Dia 4-5)
- Validação sintática (Dia 6-7)

iwietzke:

- Sistema de pipes funcional (Dia 1-4)
- Built-ins: cd, export, unset (Dia 4-6)
- PATH resolution (Dia 6-7)

Prazo Entrega Phase 3 (Pipes): 16/06

Semana 4: Here-doc e Built-ins Finais (17/06 - 23/06)

luarodri:

- Implementar here-document (Dia 1-4)
- Refinamento do parser (Dia 5-6)
- Testes de parsing (Dia 6-7)

iwietzke:

- Built-in: exit (Dia 1-2)
- Gerenciamento de ambiente (Dia 3-5)
- Testes de execução (Dia 6-7)

Prazo Entrega Phase 3 (Here-doc/Built-ins): 23/06

Semana 5: Integração Principal (24/06 - 30/06)

Ambos:

- Primeira integração completa (Dia 1-3)
- Interface parser ↔ executor (Dia 3-4)
- Resolver conflitos de design (Dia 4-5)
- Testes básicos end-to-end (Dia 6-7)

Prazo Entrega Phase 4 (Integração): 30/06

Semana 6: Signal Handling e Polimento (01/07 - 07/07)

Ambos:

- Implementar signal handling (Dia 1-3)
- Modo interativo completo (Dia 3-4)
- Histórico de comandos (Dia 5)
- Memory management (Dia 6-7)

Prazo Entrega Phase 5 (Signals): 07/07

Semana 7: Testing Extensivo (08/07 - 14/07)

Ambos:

- Bateria de testes automáticos (Dia 1-3)
- Comparação com bash (Dia 3-4)
- Corner cases (Dia 4-5)
- Performance testing (Dia 6)
- Norm compliance (Dia 7)

Prazo Entrega Phase 6/7 (Testing): 14/07

Semana 8: Finalização e Defesa (15/07 - 21/07)

Ambos:

- Bug fixes finais (Dia 1-3)

- Code review completo (Dia 3-4)
- Documentação (Dia 5)
- Preparar apresentação (Dia 6)
- Submissão final (Dia 7)

Prazo Entrega Final: 21/07

Considerações sobre o cronograma:

- As datas são sugestivas e podem ser ajustadas conforme disponibilidade
- Cada phase tem um prazo claro de entrega no domingo
- Recomenda-se manter 1-2 dias de buffer por semana para imprevistos
- As integrações devem começar mais cedo se possível
- Semanas 7-8 (testing e finalização) são cruciais e não devem ser comprimidas

---

BONUS (SE TEMPO PERMITIR)

luarodri: Logical Operators

- Implementar && e ||
- Suporte a parênteses
- Precedência de operadores
- Avaliação condicional

iwietzke: Wildcards

- Expansão de * no diretório atual
- Pattern matching
- Ordenação alfabética
- Integração com pipes

ESTRATÉGIAS DE COORDENAÇÃO

Comunicação Diária:

- Daily standup (5-10 min)
- Status: o que fez ontem, vai fazer hoje, blockers
- Sync design decisions
- Planning da integração

Git Workflow:

- Branch main sempre funcional
- Feature branches para desenvolvimento
- Pull requests com review
- Commits atômicos e bem documentados

Divisão de Responsabilidades:

luarodri (Frontend/Parsing):

- Toda análise de texto de entrada
- Tokenização e parsing
- Expansão de variáveis
- Validação sintática
- Here-documents

iwietzke (Backend/Execution):

- Sistema de execução
- Redirecionamentos
- Pipes e processos
- Built-ins commands
- Signal handling (implementação)

Pontos de Integração Críticos:

1. **Interface Parser → Executor:**
   - Estrutura de dados do comando parseado
   - Lista de redirecionamentos
   - Array de argumentos expandidos
2. **Signal Handling:**
   - Variável global compartilhada
   - Comportamento em modo interativo
   - Cleanup de recursos
3. **Environment Management:**
   - Tabela de variáveis shared
   - Export/unset from parser
   - Inheritance para child processes

Testing Strategy:

1. **Unit Tests (Individual):**
   - Cada função testada isoladamente
   - Mock interfaces quando necessário
   - Edge cases específicos de cada módulo
2. **Integration Tests (Conjunto):**
   - Parser + Executor funcionando junto
   - Pipelines complexos
   - Redirecionamentos múltiplos
3. **System Tests (Comportamental):**
   - Comparação 1:1 com bash
   - Scripts automatizados
   - Casos de uso reais

Risk Mitigation:

Riscos Identificados:

- Incompatibilidade de design entre módulos
- Memory leaks complexos
- Signal handling em pipelines
- Performance em comandos grandes

Mitigações:

- Design sessions semanais
- Code review obrigatório
- Profiling regular
- Testes stress contínuos

DELIVERABLES POR SEMANA

Semana 1:

- [ ] Estrutura completa do projeto
- [ ] Makefile funcional
- [ ] Structs principais definidas
- [ ] Tokenizer básico (luarodri)
- [ ] Execução simples (iwietzke)

Semana 2:

- [ ] Parser de comandos simples (luarodri)
- [ ] Redirecionamentos básicos (iwietzke)
- [ ] 3 built-ins funcionais (iwietzke)
- [ ] Expansão de variáveis (luarodri)

Semana 3:

- [ ] Pipeline parsing (luarodri)
- [ ] Pipeline execution (iwietzke)
- [ ] Todos built-ins (iwietzke)
- [ ] Expansion completa (luarodri)

Semana 4:

- [ ] Here-doc funcional (luarodri)
- [ ] Environment completo (iwietzke)
- [ ] Validação robusta (luarodri)
- [ ] Error handling (iwietzke)

Semana 5-6:

- [ ] Integração completa
- [ ] Signal handling
- [ ] Interactive mode
- [ ] Memory management

Semana 7-8:

- [ ] Testing suite completa
- [ ] Norm compliance
- [ ] Performance optimization
- [ ] Documentação final

Este plano garante que ambos tenham trabalho equivalente e complementar, com pontos claros de sincronização e integração ao longo do desenvolvimento.