Rodeo VM

Linguagem de Programação para Controle de Touro Mecânico

Por que criar uma linguagem específica?

- **Domínio Específico**: Controle de sistemas mecânicos requer comandos especializados
- **Segurança**: Linguagem projetada com limites e verificações de segurança embutidas
- **Simplicidade**: Sintaxe intuitiva para operadores de parques de diversão
- **Real-time**: Execução direta e eficiente para controle em tempo real

Aplicações Práticas

- Parques de diversão e eventos
- Simuladores de rodeio
- Treinamento de atletas
- Entretenimento interativo

Características Principais

1. Comandos de Controle Direto

```
speed(75);      // Define velocidade (0-100%)
torque(80);     // Define torque (0-100%)
yaw(10);        // Define rotação (graus/passo)
brake(1);       // Ativa/desativa freio (0/1)
wait(1000);     // Aguarda tempo em milissegundos
```

2. Modos de Operação Pré-definidos

```
pattern(CALM);        // Modo suave
pattern(SWIRL);       // Modo giratório
pattern(AGGRESSIVE);  // Modo agressivo
```

3. Leitura de Sensores

```
read(rider) -> has_rider;      // Presença do piloto
read(tilt) -> angle;           // Ângulo de inclinação
read(rpm) -> rotation;         // Rotações por minuto
read(emergency) -> stop;       // Botão de emergência
read(time_ms) -> elapsed;      // Tempo decorrido
```

4. Estruturas de Controle

```
// Condicional
if (tilt > 30) {
    brake(1);
    speed(0);
}

// Loop
counter = 0;
while (counter < 10) {
    speed(counter * 10);
    counter = counter + 1;
}
```

5. Operações Aritméticas

```
velocity = base_speed + increment;
torque_adjust = max_torque - safety_margin;
total = value * multiplier / divisor;
```

Arquitetura

Pipeline de Compilação

```
Código Rodeo (.rodeo)
       ↓
  [FLEX] Análise Léxica
       ↓
  [BISON] Análise Sintática
       ↓
  AST (Abstract Syntax Tree)
       ↓
  VM Personalizada
       ↓
  Execução com Simulação
```

Componentes

1. **Lexer (Flex)**: Tokenização do código fonte
2. **Parser (Bison)**: Análise sintática e construção da AST
3. **AST**: Representação intermediária do programa
4. **VM**: Máquina virtual personalizada para execução

Detalhes Técnicos

1. Sistema de Tipos Simples

- Apenas inteiros (adequado para controle de hardware)
- Sem necessidade de declaração de tipo
- Inferência automática

2. Proteções de Segurança

- Limites automáticos (speed 0-100%, torque 0-100%)
- Proteção contra loops infinitos (máximo 10.000 iterações)
- Verificação de divisão por zero
- Leitura de sensores simulados

3. VM Personalizada

- Stack-based execution
- Tabela de símbolos com até 100 variáveis
- Estado do touro mecânico mantido em contexto
- Simulação de sensores baseada no estado atual

4. Características da Linguagem

- Comentários de linha única (`//`)
- Case-sensitive
- Ponto e vírgula obrigatório
- Sintaxe inspirada em C/JavaScript

Gramática EBNF

```
Program = { Statement } ;

Statement = Assignment | IfStmt | WhileStmt | Command ;

Assignment = Identifier "=" Expression ";" ;

IfStmt = "if" "(" Condition ")" "{" { Statement } "}"
         [ "else" "{" { Statement } "}" ] ;

WhileStmt = "while" "(" Condition ")" "{" { Statement } "}" ;

Command = SpeedCmd ";" | TorqueCmd ";" | YawCmd ";"
        | BrakeCmd ";" | WaitCmd ";" | PatternCmd ";"
        | SensorCmd ";" ;

SpeedCmd = "speed" "(" Expression ")" ;
TorqueCmd = "torque" "(" Expression ")" ;
YawCmd = "yaw" "(" Expression ")" ;
BrakeCmd = "brake" "(" Expression ")" ;
WaitCmd = "wait" "(" Expression ")" ;
PatternCmd = "pattern" "(" Mode ")" ;
SensorCmd = "read" "(" Sensor ")" "->" Identifier ;

Expression = Term { ("+" | "-" | "*" | "/") Term } ;
Term = Number | Identifier | "(" Expression ")" ;

Condition = Expression RelOp Expression ;
RelOp = "==" | "!=" | ">" | "<" | ">=" | "<=" ;

Mode = "CALM" | "SWIRL" | "AGGRESSIVE" ;
Sensor = "rider" | "tilt" | "rpm" | "emergency" | "time_ms" ;

Identifier = Letter { Letter | Digit | "_" } ;
Number = Digit { Digit } ;
```

Exemplos Práticos

Exemplo 1: Aquecimento Gradual

```
// Inicia em modo calmo
pattern(CALM);
brake(0);

// Aumenta velocidade gradualmente
speed_level = 0;
while (speed_level < 80) {
    speed_level = speed_level + 10;
    speed(speed_level);
    torque(60);
    wait(2000);
}

// Para suavemente
speed(0);
brake(1);
```

Exemplo 2: Sistema de Segurança

```
// Verifica presença do piloto
read(rider) -> has_rider;
if (has_rider == 0) {
    brake(1);
    speed(0);
}

// Monitora inclinação
read(tilt) -> angle;
if (angle > 35) {
    // Emergência: inclinação perigosa
    brake(1);
    speed(0);
    wait(500);
}

// Verifica botão de emergência
read(emergency) -> em_pressed;
if (em_pressed == 1) {
    brake(1);
    speed(0);
}
```

Exemplo 3: Rotina Completa

```
pattern(SWIRL);
speed(40);
brake(0);

counter = 0;
while (counter < 10) {
    // Aumenta intensidade
    speed(counter * 10);
    torque(70);
    yaw(5);
    wait(200);

    // Verifica inclinação
    read(tilt) -> t;
    if (t > 25) {
        brake(1);
        speed(0);
        wait(500);
        brake(0);
    }

    // Verifica emergência
    read(emergency) -> em;
    if (em == 1) {
        speed(0);
        brake(1);
        counter = 100;  // Força saída do loop
    }

    counter = counter + 1;
}

// Finalização segura
speed(0);
brake(1);
```

Como Usar

Compilação

```
make
```

Execução

```
./rodeo-vm programa.rodeo
```

Execução dos Exemplos

```
# Teste básico
./rodeo-vm examples/test_basic.rodeo

# Teste de operações aritméticas
./rodeo-vm examples/test_arithmetic.rodeo

# Teste de loops
./rodeo-vm examples/test_while.rodeo

# Sistema de segurança completo
./rodeo-vm examples/test_safety.rodeo
```

Saída da VM

A VM fornece visualização detalhada da execução:

```
╔═══════════════════════════════════════╗
║    RODEO VM - Mechanical Bull Simulator   ║
╚═══════════════════════════════════════╝

▶ Starting program execution...

  [SPEED] set to 50%
  [TORQUE] set to 60%
  [PATTERN] set to CALM
  [SENSOR] rider -> has_rider = 1
  [IF] condition = TRUE
  ...

✓ Program execution completed.

┌─────────────────────────────────────┐
│          FINAL RODEO STATE          │
├─────────────────────────────────────┤
│ Speed:    50%  ⚡ ACTIVE            │
│ Torque:   60%                       │
│ Yaw:       5° per step              │
│ Brake:   ✓ OFF                      │
│ Pattern: ✓ CALM                     │
├─────────────────────────────────────┤
│          SENSOR READINGS            │
├─────────────────────────────────────┤
│ Rider:     ✓ PRESENT                │
│ Tilt:        0°                     │
│ RPM:        500                     │
│ Emergency: ✓ OK                     │
└─────────────────────────────────────┘
```

Detalhes de Implementação

Tokens Reconhecidos

- **Palavras-chave**: `if`, `else`, `while`, `speed`, `torque`, `yaw`, `brake`, `wait`, `pattern`, `read`
- **Modos**: `CALM`, `SWIRL`, `AGGRESSIVE`
- **Sensores**: `rider`, `tilt`, `rpm`, `emergency`, `time_ms`
- **Operadores**: `+`, `-`, `*`, `/`, `==`, `!=`, `>`, `<`, `>=`, `<=`, `=`
- **Delimitadores**: `(`, `)`, `{`, `}`, `;`, `->`

Estrutura da AST

```
// Tipos de nós
- STMT_ASSIGNMENT: Atribuição de variável
- STMT_IF: Condicional
- STMT_WHILE: Loop
- STMT_SPEED: Comando de velocidade
- STMT_TORQUE: Comando de torque
- STMT_YAW: Comando de rotação
- STMT_BRAKE: Comando de freio
- STMT_WAIT: Comando de espera
- STMT_PATTERN: Seleção de padrão
- STMT_SENSOR_READ: Leitura de sensor
- STMT_BLOCK: Bloco de comandos
```