

QuadraSimples

Linguagem de Programação para Tênis

APS - Lógica da Computação

Insper - Engenharia da Computação

Daniel Marco Djanikian

Dezembro 2025

Motivação

Por que criar uma linguagem para tênis?

- **Treinos estruturados:** Descrever rotinas de forma programática
- **Simulação de partidas:** Reproduzir jogos com placar oficial
- **Domínio específico:** Comandos naturais para o contexto esportivo
- **Repetição e variação:** Loops e condicionais para padrões táticos
- **Análise de desempenho:** Variáveis para rastrear estatísticas

Diferencial: Combina lógica de programação imperativa com vocabulário de tênis profissional.

Características da Linguagem

Categoria	Recursos
Estruturas Básicas	Variáveis, atribuição, expressões aritméticas e booleanas
Controle de Fluxo	<code>if/else</code> , <code>while</code> , blocos <code>{}</code>
Entrada/Saída	<code>readline()</code> , <code>log()</code> , <code>print()</code>
Comandos de Tênis	<code>sessao</code> , <code>etapa</code> , <code>saque</code> , <code>rally</code> , <code>point</code> , <code>ace</code> , <code>double_fault</code>
Gerenciamento de Jogo	<code>start_match</code> , <code>score</code> , <code>rename</code> , <code>config</code>
Tipos de Dados	Inteiros, booleanos, strings

Exemplo: Treino Básico Completo

```
sessao("Treino de Fundamentos") {  
    etapa("Aquecimento", dur=15);  
  
    contador = 0;  
    while (contador < 10) {  
        rally(8);  
        point(A);  
        contador = contador + 1;  
    };  
  
    descanso(5);  
  
    etapa("Saques", reps=20);  
    saque(A, "deuce");  
    saque(B, "ad");  
}  
  
config(no_ad = true);  
start_match(3, 6);  
point(A);  
ace(B);  
score();
```

Vantagens: Sintaxe clara e legível - Mistura comandos de domínio com lógica imperativa - Fácil de entender mesmo sem conhecer tênis.

Exemplo: Simulação de Partida Completa

```
start_match(3, 6);
config(no_ad = true);
rename(A, "Nadal");
rename(B, "Federer");

saque(A, "deuce");
rally(8);
point(A);
double_fault(B);

saque(B, "ad");
rally(12);
point(B);
ace(B);

score();
```

Saída: Placar profissional com box-drawing Unicode.

Gramática EBNF - Resumo Estrutural

```
program = { statement } ;

statement = expr_stmt | if_stmt | while_stmt | sessao_stmt | block ;

Expressões:
expr_stmt = assignment | log_stmt | print_stmt | etapa_stmt
           | saque_stmt | rally_stmt | point_stmt | ace_stmt
           | score_stmt | rename_stmt | config_stmt ;

Controle de Fluxo:
if_stmt    = "if" "(" bool_expr ")" block [ "else" block ] ";" ;
while_stmt = "while" "(" bool_expr ")" block ";" ;

DSL de Tênis:
sessao_stmt = "sessao" "(" string ")" block ";" ;
etapa_stmt  = "etapa" "(" string "," etapa_type "=" expression ")" ";" ;
rally_stmt  = "rally" "(" expression ")" ";" ;
point_stmt  = "point" "(" player ")" ";" ;

Expressões Booleanas:
bool_expr  = bool_term { "||" bool_term } ;
bool_term   = rel_expr { "&&" rel_expr } ;
rel_expr    = expression [ ("==" | ">" | "<") expression ] ;

Expressões Aritméticas:
expression = term { ("+" | "-") term } ;
term       = factor { ("*" | "/") factor } ;
factor     = number | identifier | sensor | "(" bool_expr ")"
           | "+" factor | "-" factor | "!" factor ;
```

Documentação completa: [docs/gramatica-ebnf.md](#)

Arquitetura do Sistema



QuadraVM - Máquina Virtual

Arquitetura Stack-Based com Registradores

Componente	Descrição
Stack	1024 elementos para operações aritméticas/lógicas
Variáveis	128 slots para armazenamento
Sensores read-only	<code>last_rally</code> , <code>total_points</code>
Registradores	<code>R0</code> , <code>R1</code> (uso geral) + match state (pontos, games, sets)
Labels	512 posições para saltos (if/while)
Match State	Rastreamento completo de placar de tênis

ISA - Instruction Set Architecture

Principais Categorias (40+ instruções)

- **Stack:** PUSH_NUM, POP, DUP
- **Aritmética:** ADD, SUB, MUL, DIV, MOD
- **Lógica:** AND, OR, NOT, EQ, LT, GT, LE, GE
- **Registradores:** LOAD_R0, LOAD_R1, PUSH_R0, PUSH_R1, INC_R0, DEC_R0, INC_R1, DEC_R1
- **Variáveis:** LOAD_VAR, STORE_VAR
- **Controle:** LABEL, JUMP, JZ, JNZ
- **I/O:** READLINE, LOG, PRINT
- **Tênis:** START_MATCH, ADD_POINT, PRINT_SCORE, SET_CONFIG, etc.

Turing-Compleitude

QuadraVM é equivalente a uma Counter Machine

Requisitos satisfeitos:

1. **Memória ilimitada:** Stack (1024) + Variáveis (128)
2. **Registradores:** R0, R1 (mínimo: 2)
3. **Operações primitivas:**
 - INC_R0, INC_R1 (incremento)
 - DEC_R0, DEC_R1 (decremento)
 - JZ (salto se zero)

Conclusão: Com INC, DEC e JZ, a VM pode simular qualquer Máquina de Turing - **Referência:** [Counter Machine - Wikipedia](#)

Ferramentas de Compilação

Stack Tecnológico

Ferramenta	Função	Arquivo
Flex 2.6+	Análise léxica (tokenização)	lexer.l
Bison 3.8+	Análise sintática (parsing)	parser.y
GCC/C11	Geração de código intermediário	codegen.c
Make	Sistema de build	Makefile

Exemplos de Teste (9 programas)

Arquivo	Descrição
demo.qs	Demonstração geral da linguagem com todos os recursos
treino_basico.qs	Sessões de treino estruturadas com etapas e descanso
controle_fluxo.qs	Estruturas condicionais if/else, loops while e entrada readline
partida_completa.qs	Simulação completa de partida com placar visual
loops.qs	Demonstração de loops aninhados e controle de fluxo complexo
match_simulation.qs	Match profissional completo: 6 games entre Nadal e Federer
rally.qs	Treino interativo de rally com entrada do usuário
sensores_demo.qs	Demonstração de sensores read-only (last_rally, total_points)
test_print.qs	Testes do comando print() com diferentes tipos de dados

Curiosidade: Placar Visual

Caracteres Unicode Box-Drawing

PLACAR - SET 1		
Nadal	3	15
Federer	2	30

Implementação: Função `print_score()` em `quadravm.c` - **Vantagem:** Feedback visual profissional durante simulações - **Detalhes:** Suporta sistema de pontuação oficial (0-15-30-40-game)

Curiosidade: Modo No-Ad

Configuração Personalizada de Partida

```
# Modo tradicional (advantage)
config(no_ad = 0);

# Modo rápido (no advantage - ponto de ouro em 40-40)
config(no_ad = 1);
```

No-Ad Tennis: Formato usado em competições rápidas onde em 40-40 não há "vantagem", mas sim um ponto decisivo (sudden death).

Implementação: Flag no `MatchState` da VM que altera o comportamento da função `add_point_to_player()`.

Curiosidade: Comandos de Domínio

Vocabulário Específico de Tênis

Comando	Uso no Tênis Real
ace(player)	Saque direto, sem toque do adversário
double_fault(player)	Duas falhas de saque consecutivas
rally(num_trocas)	Simula troca de bolas no ponto (número de golpes)
descanso(mins)	Pausa entre exercícios
rename(player, nome)	Trocar nome do jogador durante match

Objetivo: Fazer código que "soa" como narração de partida!

Estrutura do Projeto

```
APS-QuadraSimples-LogComp/
├── lexer.l          # Análise léxica (Flex)
├── parser.y         # Análise sintática (Bison)
├── codegen.c/h      # Geração de assembly .qvm
├── main.c           # Entry point do compilador
├── quadravm.c      # Máquina virtual
├── Makefile          # Sistema de build
├── README.md        # Documentação completa
└── docs/
    ├── gramatica-ebnf.md      # EBNF formal
    └── QuadraSimples-APS-slides.html # Apresentação original
└── examples/
    ├── demo.qs
    ├── treino_basico.qs
    ├── controle_fluxo.qs
    ├── partida_completa.qs
    ├── loops.qs
    ├── match_simulation.qs
    ├── rally.qs
    ├── sensores_demo.qs
    └── test_print.qs
```

Como Compilar e Executar

1. Compilar o projeto

```
$ make
```

2. Compilar um programa QuadraSimples

```
$ ./quadrasimples examples/demo.qs
```

3. Executar na VM

```
$ ./quadravm out.qvm
```

Atalho (compilar + executar)

```
$ ./quadrasimples examples/partida_completa.qs && ./quadravm out.qvm
```

Conceito Bônus: QuadraVM - Implementação Própria

Por que criar uma VM do zero?

Em vez de usar máquinas virtuais prontas como LLVM ou Python, desenvolvi a **QuadraVM em C** para ter controle total sobre o comportamento da linguagem:

- **Estado de jogo integrado:** A VM mantém o placar da partida (pontos, games, sets) internamente, como se fosse uma quadra de tênis virtual
- **Comandos nativos de tênis:** Operações como "dar ponto", "sacar" e "mostrar placar" são instruções diretas da máquina, não funções externas
- **Arquitetura flexível:** Combina uma pilha para cálculos matemáticos com registradores para controlar repetições e loops
- **Placar visual personalizado:** A VM desenha o placar de tênis formatado no terminal, algo impossível com VMs genéricas
- **Computacionalmente completa:** Possui instruções de incremento, decremento e salto condicional, permitindo implementar qualquer algoritmo

Resultado: Uma máquina virtual especializada em tênis, que entende o domínio do esporte ao invés de ser apenas um interpretador genérico.

Conclusão

QuadraSimples

Uma DSL completa para o mundo do tênis, com compilador robusto e VM customizada.

Repositório GitHub: [APS-QuadraSimples-LogComp](#) | **Autor:** Daniel Marco Djanikian | **Curso:** Engenharia da Computação - Insper | **Disciplina:** Lógica da Computação

Obrigado!