Robot Arm VM

Linguagem de Programação para Controle de Braços Robóticos

Autor: Antonio Lucas Michelon de Almeida
Disciplina: Lógica da Computação - 2025.2
Instituto: Insper
Professor: Raul Ikeda

Motivação

Por que criar uma linguagem para braços robóticos?

Problema Atual:

- Programação de robôs industriais é **complexa e inacessível**
- Linguagens existentes exigem conhecimento profundo de hardware
- Assembly e linguagens de baixo nível são **difíceis de ler e manter**
- Curva de aprendizado alta impede prototipagem rápida
- Falta de ferramentas educacionais para ensinar robótica

Nossa Solução:

- Linguagem de **alto nível** com sintaxe **intuitiva em português**
- Programação **declarativa** focada em "o que fazer", não "como fazer"
- Compilação automática para Assembly otimizado e eficiente
- Feedback visual em tempo real facilita aprendizado e debug
- Simulador integrado permite testes sem necessidade de hardware físico

Aplicações Práticas:

- **Automação Industrial:** Programação rápida de linhas de produção
- **Educação:** Ensino de robótica para iniciantes
- **Prototipagem:** Desenvolvimento ágil de soluções automatizadas
- **Controle de Qualidade:** Inspeção automatizada de produtos
- **Pesquisa**: Base para experimentos em compiladores e linguagens

---

Características da Linguagem

1. Sintaxe em Português - Natural e Intuitiva

Diferente de linguagens tradicionais em inglês, nossa linguagem usa **comandos em português** que facilitam o entendimento:

```
// Programa completo em apenas 5 linhas!
mover cima 20        // Move braço para cima
mover direita 30     // Move braço para direita
pegar                // Segura objeto
esperar 1000         // Aguarda 1 segundo
soltar               // Libera objeto
```

Vantagens:

- Ideal para estudantes brasileiros
- Reduz barreira linguística no aprendizado
- Código autodocumentado e legível
- Facilita trabalho em equipe

---

2. Estruturas de Controle Completas

Variáveis - Armazenamento de Dados

```
var contador = 0
var distancia = 50
var peso_maximo = 10
```

Condicionais - Tomada de Decisão

```
se objeto: {
    pegar
    mover esquerda 40
} senao: {
    esperar 500
}
```

Loops - Repetição Controlada

```
enquanto contador < 10: {
    mover cima 5
    contador = contador + 1
}
```

---

3. Operações Matemáticas Expressivas

Operadores Aritméticos: `+`, `-`, `*`, `/`

```
var base = 10
var altura = base * 2
var area = (base + altura) / 2
var perimetro = base + base + altura + altura
```

Operadores Relacionais: `==`, `!=`, `<`, `>`, `<=`, `>=`

```
se peso > 5: {
    // Objeto pesado - cuidado extra
    mover devagar
}

enquanto distancia != 0: {
    mover direita 1
    distancia = distancia - 1
}
```

---

4. Comandos de Controle do Robô

mover [direção] [quantidade]
Move o braço robótico no espaço 2D

- Direções: `cima`, `baixo`, `esquerda`, `direita`
- Quantidade: distância em unidades (0-100)

pegar
Fecha a garra para segurar objetos detectados

soltar
Abre a garra para liberar objetos segurados

esperar [milissegundos]
Pausa a execução por tempo determinado

---

5. Sensores Inteligentes para Interação

objeto - Sensor de Presença
Detecta se há objeto na posição atual do braço

peso - Sensor de Massa
Mede o peso do objeto segurado (escala 0-10)

limite - Sensor de Segurança
Detecta quando o braço atinge os limites físicos de movimento

---

Curiosidades do Projeto

1. Máquina Virtual Própria em Assembly!

Não usamos Python nem JavaScript - criamos uma **VM do zero em Assembly x86**!

Por quê?

- Controle total sobre execução
- Desempenho máximo (código nativo)
- Aprendizado profundo de arquitetura de computadores

Como funciona:

- Registradores customizados simulam o estado do robô
- `R_X` e `R_Y` : Posição do braço no espaço (0-100)
- `R_GARRA` : Estado da garra (0=aberta, 1=fechada)
- Sensores simulados por lógica condicional em Assembly

Sensores Simulados:

- `S_OBJETO` - Detecção de objetos por posição
- `S_PESO` - Peso variável por objeto
- `S_LIMITE` - Proteção contra movimentos inválidos

---

2. Feedback Visual em Tempo Real

Cada operação mostra exatamente o que está acontecendo:

```
 ========================================
[ROBO] Iniciando programa...
========================================
[ROBO] Posicao atual -> X=50, Y=70
[ROBO] Garra FECHADA - Objeto segurado! (Peso=5)
[ROBO] Aguardando 1000 ms...
[AVISO] Limite de movimento atingido!
[ROBO] Garra ABERTA - Objeto liberado!
========================================
[ROBO] Programa finalizado com sucesso!
========================================
```

Benefícios:

- Debug simplificado para estudantes
- Entendimento visual do fluxo de execução
- Validação de lógica em tempo real

---

3. Execução Super Simples - Um Comando Apenas!

Compilar e executar nunca foi tão fácil:

```
./robotarm examples/07_hello_robot.robot
```

Isso substitui 4 comandos complexos e

---

4. Linguagem Turing-Completa

Nossa linguagem pode computar **qualquer função computável**!

Requisitos para Turing-Completude:

- **Armazenamento arbitrário** → Variáveis ilimitadas
- **Operações aritméticas** → +, -, *, /
- **Condicionais** → if/else
- **Loops** → while com condição

Isso significa que podemos:

- Implementar qualquer algoritmo clássico
- Simular outras máquinas de Turing
- Resolver problemas computáveis arbitrariamente complexos

5. Detecção Inteligente de Objetos

Objetos são detectados por **posição no espaço**:

```
// Objeto em (70, 50)
mover direita 20  // X=70
se objeto: {      // Detectado!
    pegar
}
```

Peso varia conforme objeto (simulação realista)!

Exemplos de Código

Exemplo 1: Busca Inteligente com Sensores

```
// Busca objeto até encontrar
var encontrou = 0

enquanto encontrou == 0: {
    mover direita 5

    se objeto: {
        encontrou = 1
        pegar
    }
}
```

Resultado:

```
[ROBO] Posicao atual -> X=55, Y=50
[ROBO] Posicao atual -> X=60, Y=50
[ROBO] Posicao atual -> X=65, Y=50
[ROBO] Posicao atual -> X=70, Y=50  // Objeto encontrado!
[ROBO] Garra FECHADA - Objeto segurado! (Peso=5)
```

O que demonstra:

- Loop com condição de parada (`enquanto`)
- Uso do sensor `objeto`
- Variáveis para controle de estado
- Interação inteligente com ambiente

Exemplo 2: Linha de Montagem Industrial

```
// Processa 3 peças automaticamente
var pecas = 0

enquanto pecas < 3: {
    // Move para área de coleta
    mover direita 30
    mover cima 20
    pegar

    // Move para área de entrega
    mover esquerda 40
    mover baixo 10
    soltar

    // Volta à posição inicial
    mover direita 10
    mover baixo 10

    // Incrementa contador
    pecas = pecas + 1
}
```

O que demonstra:

- Automação completa de processo industrial
- Loop com múltiplas iterações
- Expressões aritméticas (`pecas + 1`)
- Condicionais (`pecas < 3`)
- Todos os comandos do robô (mover, pegar, soltar)

Gramática da Linguagem (EBNF)

Definição Formal

```
Program    = { Statement } ;

Statement  = VarDecl | AssignStmt | MoveStmt
           | GrabStmt | ReleaseStmt | WaitStmt
           | IfStmt | WhileStmt | Block ;

VarDecl    = "var" Identifier "=" Expr ;
AssignStmt = Identifier "=" Expr ;

MoveStmt    = "mover" Direction Number ;
Direction   = "cima" | "baixo" | "esquerda" | "direita" ;
GrabStmt    = "pegar" ;
ReleaseStmt = "soltar" ;
WaitStmt    = "esperar" Number ;

IfStmt   = "se" Condition ":" Block [ "senao" ":" Block ] ;
WhileStmt = "enquanto" Condition ":" Block ;
Block    = "{" { Statement } "}" ;

Condition = Expr RelOp Expr | Sensor ;
RelOp     = "==" | "!=" | "<" | ">" | "<=" | ">=" ;
Sensor    = "objeto" | "peso" | "limite" ;

Expr   = Term { ("+" | "-") Term } ;
Term   = Factor { ("*" | "/") Factor } ;
Factor = Number | Identifier | "(" Expr ")" | Sensor ;
```

Tecnologias Utilizadas

Ferramentas de Desenvolvimento

Ferramenta	Versão	Uso
Flex	2.6+	Análise Léxica (Tokenização)
Bison	3.8+	Análise Sintática (Parser)
NASM	2.15+	Montagem Assembly x86
GCC	11+	Compilação e Linkagem
Make	4.3+	Automação de Build

Conceitos de Compiladores

- **Análise Léxica:** Transformação de texto em tokens
- **Análise Sintática:** Validação de gramática e geração de AST
- **Geração de Código:** Tradução para Assembly
- **Otimização:** Código eficiente e compacto

Como Usar

Instalação Rápida (Ubuntu/Debian)

```
# 1. Instalar dependências
sudo apt-get update
sudo apt-get install flex bison gcc gcc-multilib nasm make

# 2. Clonar repositório
git clone https://github.com/antoniolma/RobotArmVM.git
cd RobotArmVM

# 3. Compilar
make
```

Uso Básico

```
# Executar exemplo
./robotarm examples/07_hello_robot.robot

# Ver todos os exemplos disponíveis
./robotarm

# Testar todos os 10 exemplos
make test
```

Conclusão

Uma **linguagem de programação completa** para controle de braços robóticos

Um **compilador funcional** usando ferramentas profissionais (Flex/Bison)

Uma **máquina virtual customizada** em Assembly x86 do zero

Um **ambiente de desenvolvimento** com feedback visual e testes

Impacto e Aplicações

- **Educacional** - Ensinar robótica de forma acessível para brasileiros
- **Industrial** - Prototipagem rápida de automações em fábricas
- **Acadêmico** - Base para estudos em compiladores e linguagens
- **Pesquisa** - Plataforma para experimentos em controle robótico

Diferenciais do Projeto

Sintaxe em **português** facilita aprendizado

- **VM própria** oferece controle total
- **Feedback visual** simplifica debug
- **Wrapper script** torna uso trivial
- **Turing-completa** permite qualquer algoritmo

Robot Arm VM

Repositório:
https://github.com/antoniolma/RobotArmVM

Autor:
Antonio Lucas Michelon de Almeida
Insper - Engenharia da Computação - 2025.2