Compilador e Interpretador LISP (Python) 🐍

Projeto Integrador III - UNIPAMPA

Autores: Filipe Rosa e Lucas Moreira

Este projeto consiste na implementação completa de um compilador para um subconjunto da linguagem de programação **LISP**. O sistema é composto por um Analisador Léxico, Analisador Sintático (Parser), Gerador de Código Intermediário e uma Máquina Virtual de Pilha (Stack VM) com suporte a um shell interativo (REPL).

🚀 Funcionalidades

- **Análise Léxica e Sintática:** Reconhecimento de tokens e validação gramatical baseada em S-expressions.
- **Geração de AST:** Criação e visualização da Árvore de Sintaxe Abstrata.
- **Compilação:** Tradução de LISP para um código intermediário (Assembly de Pilha).
- **Máquina Virtual:** Execução do código com suporte a:
  - Recursão.
  - Escopo de variáveis (parâmetros locais).
  - Manipulação de Listas (cons, car, cdr).
- **Modo Interativo (REPL):** Permite definir funções e testar expressões diretamente no terminal.
- **Logs Detalhados:** Gera arquivos de texto contendo os tokens, a AST e o código intermediário de cada execução.

🛠️ Pré-requisitos e Instalação

O projeto foi desenvolvido em **Python 3**. A única dependência externa é a biblioteca PLY (Python Lex-Yacc).

1. **Clone o repositório:**
   git clone
   https://github.com/FilipeTRosa/LISP_Compiler.git
   cd LISP_Compiler

2. **Instale a dependência:**
   pip install ply

💻 Como Usar

Você pode usar o compilador de duas formas: lendo um arquivo ou digitando no terminal.

1. Executando um Arquivo (.lisp)

Crie um arquivo chamado codigo_fonte.lisp na mesma pasta do projeto com o seu código. Ao rodar o programa, ele lerá o arquivo, executará as instruções e depois abrirá o modo interativo.

Comando:

python analizador.py

2. Modo Interativo (Shell / REPL)

Se o arquivo codigo_fonte.lisp estiver vazio ou não existir, ou após a execução do arquivo, o sistema entra no modo interativo. Você verá o prompt:

LISP >

Basta digitar sua expressão LISP e pressionar Enter. Digite sair para fechar.

📚 Manual da Linguagem

Abaixo estão os comandos suportados por este compilador.

Operadores Aritméticos e Lógicos

Comando	Descrição	Exemplo
+, -, *	Soma, Subtração, Multiplicação	(+ 10 5)
div	Divisão Inteira	(div 10 2)
mod	Resto da Divisão	(mod 10 3)
eq	Igualdade (Equal)	(eq a b)
neq	Diferença (Not Equal)	(neq a b)
gt, lt	Maior que (>), Menor que	(gt 10 5)

	(<)	
geq, leq	Maior/Igual (>=), Menor/Igual (<=)	(geq 5 5)
and, or, not	Operadores Lógicos	(and (eq a 1) (gt b 2))

Estruturas de Controle e Funções

Comando	Descrição	Sintaxe
if	Condicional Se/Senão	(if (condicao) (entao) (senao))
defun	Define uma nova função	(defun nome (params) corpo)
print	Imprime resultado no terminal	(print valor)

Manipulação de Listas

Nota: Como este é um compilador simplificado, as listas devem ser construídas explicitamente.

Comando	Descrição	Exemplo
cons	Constrói lista (adiciona item ao início)	(cons 1 (cons 2 nil)) -> [1, 2]
car	Retorna o primeiro item (Cabeça)	(car lista)
cdr	Retorna o resto da lista (Cauda)	(cdr lista)
nil	Representa lista vazia ou Falso	nil

🧪 Exemplos de Código

1. Soma Recursiva de uma Lista

Este exemplo demonstra definição de função, if, recursão e manipulação de listas (car/cdr).

```
(defun soma (lista)
  (if (eq lista nil)
      0
      (+ (car lista) (soma (cdr lista)))))

;; Para testar (constrói a lista [1, 2, 3]):
(print (soma (cons 1 (cons 2 (cons 3 nil)))))
```

2. Cálculo do Fatorial

```
(defun fatorial (n)
  (if (eq n 0)
      1
      (* n (fatorial (- n 1)))))

(print (fatorial 5))
```

3. Função de Dobro

```
(defun dobro (x)
  (* x 2))

(print (dobro 10))
```

📂 Arquivos de Saída (Logs)

Para fins de estudo e depuração, o compilador gera três arquivos a cada execução:

1. **saida_lexer.txt**: Lista de todos os tokens identificados no código fonte.
2. **saida_parser_ast.txt**: Representação textual e gráfica da Árvore de Sintaxe Abstrata (AST).
3. **saida_codigo_intermediario.txt**: O código "Assembly" gerado que foi executado pela Máquina Virtual.