**Developer**

Desenvolver um interpretador em tempo real para AdvPL e TLPP (TOTVS Language for Presentation and Processing) é um projeto bastante complexo, considerando que essas linguagens possuem suas próprias sintaxes e semânticas específicas, além de um ambiente de execução que gerencia estado, dados e interações com o sistema.

## **Proposta Geral do Interpretador**

Abaixo estão as etapas e a estrutura básica para criar um interpretador em Go, que permitirá a entrada de código fonte e a execução em tempo real:

1. **Ler Código Fonte**: Capturar a entrada do usuário.
2. **Tokenização**: Quebrar a entrada em tokens para reconhecimento da sintaxe.
3. **Análise Sintática**: Verificar se os tokens formam uma estrutura válida.
4. **Execução**: Interpretar e executar o código.
5. **Saída**: Exibir os resultados e possíveis erros.

### **Estrutura Básica do Código**

Para simplificar, aqui está uma implementação muito básica:

package main

import (

"fmt"

"os"

"strings"

)

// Função para tokenizar a entrada

func tokenize(input string) []string {

return strings.Fields(input)

}

// Função para interpretar e executar o código de AdvPL e TLPP

func interpret(tokens []string) {

if len(tokens) == 0 {

fmt.Println("Código vazio!")

return

}

// Exemplo de reconhecimento de um comando fictício

switch tokens[0] {

case "EXIBIR":

if len(tokens) > 1 {

fmt.Printf("Exibindo: %s\n", strings.Join(tokens[1:], " "))

} else {

fmt.Println("Nenhum valor para exibir.")

}

default:

fmt.Printf("Comando não reconhecido: %s\n", tokens[0])

}

}

func main() {

fmt.Println("Interpretador de AdvPL e TLPP - Digite seu código ou 'sair' para encerrar.")

for {

fmt.Print("> ")

var input string

\_, err := fmt.Scanln(&input)

if err != nil {

if err.Error() == "expected newline" {

// lida com múltiplas palavras

var extraInput string

fmt.Scanln(&extraInput)

input += " " + extraInput

}

}

if strings.ToLower(input) == "sair" {

break

}

tokens := tokenize(input)

interpret(tokens)

}

}

### **Como Funciona o Código:**

1. **Entrada do Usuário**: O usuário digita um comando.
2. **Tokenização**: A entrada é quebrada em tokens para facilitar o reconhecimento.
3. **Interpretação**: A função **interpret** avalia o comando (por exemplo, no exemplo acima, reconhece o comando **EXIBIR**).
4. **Execução e Resultado**: O comando é executado e o resultado é exibido.

### **Execução do Programa:**

Para executar o programa:

1. Salve o código acima em um arquivo chamado **main.go**.
2. No terminal, navegue até o diretório do projeto e execute:
3. **go run main.go**
4. Você poderá digitar comandos como:
5. **EXIBIR "Olá, Mundo!"**
6. Você também pode usar **sair** para terminar a execução.

### **Expansões e Melhorias**

Um interpretador completo exigiria:

* **Gramática Completa**: Para suportar a linguagem AdPL/TLPP totalmente, você precisaria definir toda a gramática.
* **Gerenciamento de Estado**: A capacidade de manter variáveis e estados entre as execuções.
* **Suporte a Funções e Modularidade**: Permitir definir e invocar funções.
* **Manipulação de Erros**: Melhor tratamento de erros com mensagens mais descritivas.
* **Interface Gráfica ou de Linha de Comando Aprimorada**: Para uma melhor interação do usuário.

Esses são apenas os primeiros passos na criação de um interpretador. Para um interpretador real, cada aspecto deve ser aprimorado e expandido para cobrir as características ricas das linguagens AdvPL e TLPP. Se precisar de algo mais específico ou se houver interesse em alguma funcionalidade adicional, fique à vontade para perguntar!