**Developer**

Os arquivos advPL foram lidos corretamente e contêm a definição de programas e comentários. Vamos agora estruturar a conversão do código AdvPL para Go.

### **Conversão do AdvPL para Go**

#### **1. Estrutura do Código**

Cada arquivo AdvPL segue uma estrutura similar:

* Inclui bibliotecas
* Define uma função que encapsula o seu comportamento
* Contém comentários sobre o autor, data e descrição do programa.

#### **2. Conversão das Funções**

Vamos converter cada função presente nos arquivos lidos. O código será estruturado para manter a lógica conforme necessário.

#### **2.1. Conversão de AMDA002 (exemplo)**

package main

import (

"fmt"

)

// AMDA002 simula a funcionalidade da função AMDA002

func AMDA002() {

// Implementação da lógica de cadastro de contas

fmt.Println("Cadastro de Contas - AMDA002")

}

// A função principal que chama AMDA002

func main() {

AMDA002()

}

#### **2.2. Conversão de AMDC001 (exemplo)**

package main

import (

"fmt"

)

// AMDC001 simula a funcionalidade da função AMDC001

func AMDC001(dataIni string, dataFim string) {

// Implementação da lógica de consulta de lançamentos

fmt.Printf("Consulta de lançamentos - Início: %s, Fim: %s\n", dataIni, dataFim)

}

// A função principal que chama AMDC001

func main() {

AMDC001("2023-01-01", "2023-12-31")

}

#### **2.3. Conversão de AMDA003 (exemplo)**

package main

import (

"fmt"

)

// AMDA003 simula a funcionalidade da função AMDA003

func AMDA003() {

// Implementação da lógica de tela de controle financeiro

fmt.Println("Tela centralizadora do controle financeiro - AMDA003")

}

// A função principal que chama AMDA003

func main() {

AMDA003()

}

#### **2.4. Conversão de AMDA001 (exemplo)**

package main

import (

"fmt"

)

// AMDA001 simula a funcionalidade da função AMDA001

func AMDA001() {

// Implementação da lógica de cadastro de agrupadores

fmt.Println("Cadastro de Agrupadores - AMDA001")

}

// A função principal que chama AMDA001

func main() {

AMDA001()

}

#### **2.5. Geração do Relatório AMDR001 em Excel**

A geração do relatório AMDR001 será realizada usando a biblioteca **github.com/xuri/excelize/v2**. Aqui está um exemplo de como fazer isso:

package main

import (

"fmt"

"log"

"github.com/xuri/excelize/v2"

)

// AMDR001 gera um relatório e salva como Excel

func AMDR001(dataIni, dataFim string) {

// Criar um novo arquivo Excel

f := excelize.NewFile()

// Adicionar uma nova folha

index := f.NewSheet("Relatório")

// Definindo os títulos

f.SetCellValue("Relatório", "A1", "Data Inicial")

f.SetCellValue("Relatório", "B1", dataIni)

f.SetCellValue("Relatório", "A2", "Data Final")

f.SetCellValue("Relatório", "B2", dataFim)

// Salvar o arquivo com um nome específico

if err := f.SaveAs("relatorio\_amdr001.xlsx"); err != nil {

log.Fatalf("Erro ao salvar o arquivo: %v", err)

}

fmt.Println("Relatório gerado com sucesso: relatorio\_amdr001.xlsx")

}

func main() {

AMDR001("2023-01-01", "2023-12-31")

}

### **Instruções para Execução**

1. **Instalação das Bibliotecas**: Execute o comando abaixo para instalar a biblioteca necessária para gerar o Excel:
2. **go get github.com/xuri/excelize/v2**
3. **Executar o Programa**: Salve cada parte do código acima em um arquivo **main.go** em um novo projeto. Depois, para compilar e executar, use:
4. **go run main.go**
5. **Verificar os Resultados**: O programa **AMDR001** irá gerar um arquivo chamado **relatorio\_amdr001.xlsx**.

### **Considerações Finais**

Se precisar de mais detalhes ou ajustes na lógica do AdvPL para Go, sinta-se à vontade para fornecer exemplos ou pedir esclarecimentos!