Ex2

- Entrega 28 fev em 23:59
- Pontos 2
- Perguntas 1
- Disponível 28 fev em 12:30 - 28 fev em 23:59 11 horas e 29 minutos
- Limite de tempo Nenhum
- Tentativas permitidas Sem limite

Instruções

Para realizar esta tarefa, você:

- pode consultar a Bibliografia;

- não pode consultar colegas;

- deve observar o prazo máximo para responder, normalmente durante a aula.

- dê uma resposta objetiva, embora não haja limite mínimo ou máximo de conteúdo para sua resposta.

Este teste foi travado 28 fev em 23:59.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	5 minutos	0 de 2 *

* Algumas perguntas ainda não avaliadas

Pontuação desta tentativa: 0 de 2 *
* Algumas perguntas ainda não avaliadas
Enviado 28 fev em 19:44
Esta tentativa levou 5 minutos.

Pergunta 1
Não avaliado ainda / 2 pts

Considere os mecanismos de composição de tipos para as linguagens de programação. Escolha um, defina-o e mostre como o mesmo é implementado em um pequeno trecho de código de uma LP de sua preferência.

Sua resposta deve conter 4 componentes:

1) Mecanismo de composição escolhido;

2) Definição do mecanismo;

3) LP escolhida;

4) Trecho de código na LP escolhida.

Sua Resposta:

1) Recursividade em cauda

2)

A recursividade em cauda, também conhecida como recursão de cauda, é uma técnica de programação onde a última instrução de uma função recursiva é a própria chamada recursiva. Em outras palavras, a chamada recursiva é a última operação executada antes de retornar o resultado da função.

Em uma função recursiva comum, cada chamada recursiva resulta em uma nova instância da função sendo adicionada à pilha de chamadas. Essas chamadas permanecem na pilha até que a condição de parada seja alcançada e as chamadas começam a ser resolvidas uma a uma. Isso pode levar a um uso excessivo de memória, especialmente para funções recursivas com muitas chamadas aninhadas.

Por outro lado, na recursividade em cauda, como a chamada recursiva é a última operação a ser executada antes do retorno da função, o compilador pode otimizar o código para que não seja necessário armazenar as chamadas na pilha. Isso resulta em uma eficiência de memória significativa, pois não há necessidade de manter um registro das chamadas recursivas pendentes.

A recursão de cauda é especialmente útil em linguagens de programação que suportam otimização de chamadas de cauda, como algumas implementações de linguagens funcionais (por exemplo, Lisp, Scheme, Haskell) e em algumas linguagens imperativas modernas (por exemplo, Go). Essa técnica é frequentemente utilizada em algoritmos recursivos para melhorar o desempenho e reduzir o consumo de memória.

3) Linguagem de programação Go

4) Código executável e comentado em Go.

```
// O pacote main é o ponto de entrada para a execução do programa em Go.
package main
```

```
// Importa o pacote fmt, que é necessário para imprimir a sequência Fibonacci.
import (
    "fmt"
)

// A função fibonacciTailRecursive implementa a recursão de cauda para gerar a sequência Fibonacci.
// Recebe três parâmetros:
// - n: o número de termos a serem gerados na sequência Fibonacci.
// - a: o primeiro termo da sequência.
// - b: o segundo termo da sequência.
func fibonacciTailRecursive(n, a, b int) {
    // Verifica se n é igual a 0, indicando que não há mais termos para gerar na sequência.
    if n == 0 {
        return // Retorna, encerrando a recursão.
    }
    // Imprime o valor do primeiro termo (a) da sequência Fibonacci.
    fmt.Printf("%d ", a)
    // Chama recursivamente a função fibonacciTailRecursive com n-1, atualizando os termos a e b.
    fibonacciTailRecursive(n-1, b, a+b)
}

// A função main é onde a execução do programa começa.
func main() {
    // Imprime uma mensagem indicando que a sequência Fibonacci está sendo gerada usando recursão em cauda.
    fmt.Println("Sequência Fibonacci usando recursão em cauda:")
    // Chama a função fibonacciTailRecursive para gerar os primeiros 10 termos da sequência Fibonacci,
    // começando com os valores iniciais 0 e 1.
    fibonacciTailRecursive(10, 0, 1)
}

//Print do programa:

//Sequência Fibonacci usando recursão em cauda:

//0 1 1 2 3 5 8 13 21 34
```

RecursividadeCauda.go (Programa) (https://pucminas.instructure.com/users/98142/files/10743503?