Eliminação de Gauss

Leonardo Ferreira

2 de Julho de 2023

Resumo

Eliminação de Guass é um algoritmo que permite encontrar a solução para matrizes de qualquer tamanho. Este trabalho visa comparar a eleminação de Gauss nas linguagens de programação **Rust** e **Go** destacando suas características.

Comparativos

Tipo de dados

Na linguagem Rust por padrão as variáveis são imutáveis, sendo necessário adicionar o modificador 'mut' na declaração. Os tipos de dados disponíveis são:

- Booleano : permitindo os valroes 'true' e 'false'.
- Inteiros : inteiros com sinal 'i8', 'i16', 'i32', 'i64', 'i128' e 'isize'. E seus respectivos tamanhos sem sinal 'u8', 'u16', 'u32', 'u64', 'u128' e 'usize'.
- Caracteres : representa um caracter de 4 bytes.
- Ponto Flutuante : ponto flutuante de precisão simples 'f32' e precisão dupla 'f64'.
- Tuplas : coleção de valores de diferentes tipos.
- Arrys : coleção de valores de tipos iguais, com tamanho fixo definido em tempo de compilação.
- Ponteiros : são de dois tipos, '**brutos**' e '**inteligentes**'. Os ponteiros brutos permite que o programador manipule a memória diretamente. Ponteiros inteligentes, por outro lado, fornece mecanismos seguros para manipulação por meio do sistema **ownership**[1].

Já em Go, as variáveis são por padrão '**mutaveis**', permitindo os tipos:

- Booleano : permitindo os valroes 'true' e 'false'.
- Inteiros : inteiros com sinal 'int8', 'int16', 'int32', 'int64', seus respectivos tamanhos sem sinal 'uint8', 'uint16', 'uint32' e 'uint64'. E um ponteiro para inteiro .'uintptr'.

[1]Ownership: modelo de gerenciamento de memória, cada valor só pode ter um '**dono(owner)**' por vez, quando termina o escopo do 'dono' o valor é liberado.

- Ponto Flutuante : 'float32' e 'float64'.
- Caracteres : possui dois tipos, '**rune**' alias para 'inte '**byte**' representa um caracter de 4 bytes.
- Strings: o tipo string representa uma sequência de caracteres Unicode.
- Arrays: coleções fixas de valores do mesmo tipo, com um tamanho fixo definido em tempo de compilação.
- Estruturas: são tipos de dados personalizados que permitem que o programador defina um conjunto de campos com nomes e tipos específicos.
- Interfaces: são tipos que definem um conjunto de métodos que uma estrutura pode implementar.
- Ponteiros: são de dois tipos, '**padrão**' e '**inteligentes**'. Ponteiros **padrão** que permitem que o programador manipule a memória diretamente e os ponteiros **inteligentes** que oferecem mecanismos seguros de gerenciamento de memória.

Funções

Em Rust, as funções podem ser definidas com ou sem parâmetros e com ou sem valor de retorno. Como Rust é uma linguagem fortemente tipada, tanto os parâmetros como o(s) valor(es) de retorno dever possuir um tipo. Além disso, podem ser definidas usando tipos genericos, que permite uma função ser chamada passando argumentos de diferentes tipos.

Em Go, as funções também podem ser definidas com ou sem parâmetros e com ou sem valor de retorno. Diferente de Rust, Go possui funções variádicas, que permitem que uma função aceite um número variável de argumentos.

Ambas possuem funções anônimas (closures). Porém, as implementações em Rust e Go são diferentes. Em Rust, por ter o modelo ownership e '**emprestimo**[2]', uma closure pode pegar um valor de três maneiras, por emprestimo imutável, emprestimo mutável, e por se tornando 'dono', a função decidirá qual será usado com base na utilização do valor no corpo da função. Já em Go, as closures são definidas como funções que capturam variáveis definidas no escopo externo.

Métricas

Métrica	Rust	Go
Linhas de código	116	106
Número de imports	3	5
Tempo CPU	35ms	?
Memória	30Kb	?

Métricas referente a matrizes de tamanho 1000x1000.

[2]Emprestimo(borrowing): passagem por referência de uma variável para uma função ou outra variável. Não passa o 'owner'.

Conclusão

Concluimos que Rust e Go possuem diferenças no gerenciamento de memória, dando destaque a Rust. Mas mesmo asssim possuem muitas semelhanças principalmente na sintaxe.