1. Comparação de Código:

- Tipos de Dados
    - **C:**
        - Usa *double* para números de ponto flutuante.
        - Alocação dinâmica de memória com *malloc* e *free*.
    - **Go:**
        - Usa *float64* para números de ponto flutuante.
        - Slices *([ ][ ]float64*) para matrizes dinâmicas.
        - Gerenciamento de memória automático com garbage collector.
    - **Rust:**
        - Usa *f64* para números de ponto flutuante.
        - *Vec<Vec<f64>>* para matrizes dinâmicas.
        - Gerenciamento de memória seguro com ownership e borrowing.

- Acesso às Variáveis
    - **C:**
        - Acesso direto a ponteiros e arrays.
        - Requer liberação manual de memória.
    - **Go:**
        - Acesso direto a slices.
        - Gerenciamento automático de memória.
    - **Rust:**
        - Acesso seguro com verificações em tempo de compilação.
        - Uso de &mut para mutabilidade controlada.

- Organização de Memória
    - **C:**
        - Alocação manual de memória no heap.
        - Risco de vazamentos de memória e acesso inválido.
    - **Go:**
        - Alocação automática no heap com garbage collector.
        - Menos controle sobre o gerenciamento de memória.
    - **Rust:**
        - Alocação automática no heap com garantias de segurança.
        - Controle preciso sobre o ciclo de vida da memória.

- Chamadas de Função
    - **C**:
        - Funções simples com passagem de ponteiros.
    - **Go**:
        - Funções com passagem de slices.
    - **Rust**:

        - Funções com passagem de referências (& e &mut).

- Comandos de Controle de Fluxo
    - **C, Go e Rust**:
        - Estruturas semelhantes: for, if, else, return.
        - Rust usa iter() e rev() para loops mais expressivos.

2. Métricas de Código:

Métrica	**C**	**Go**	**Rust**
N.º Linhas	107	88	88
N.º Funções	3	3	3
Alocação de memória	Manual	Automática	Automática
Segurança de memória	Nenhuma	Garbage collector	Ownership/borrowing
Legibilidade	Média	Alta	Alta

3. Comparação de Desempenho:

Tamanho da Matriz	C(Segundos)	Go(Segundos)	Rust(Segundos)
50x50	0.000407	0.00009376	0.000249
500x500	0.176127	0.533572	0.45103
1000x1000	1.405924	4.341131	3.060692

Características da Máquina
CPU: Intel Xeon 2680v4 (14 núcleos e 28 threads)

Memória RAM: 48 GB DDR4

Sistema Operacional: Windows 11

4. Análise Geral

- C
    - Vantagens:
        - Desempenho bruto superior.
        - Controle total sobre a memória.
    - Desvantagens:
        - Gerenciamento manual de memória (propenso a erros).
        - Código mais verboso e menos seguro.
- Go
    - Vantagens:
        - Código simples e legível.
        - Gerenciamento automático de memória.
        - Boa performance para aplicações concorrentes.
    - Desvantagens:
        - Menos controle sobre o gerenciamento de memória.
        - Performance ligeiramente inferior a C e Rust.
- Rust
    - Vantagens:
        - Segurança de memória em tempo de compilação.
        - Desempenho próximo ao de C.
        - Código moderno e expressivo.
    - Desvantagens:
        - Curva de aprendizado mais íngreme.
        - Sintaxe mais complexa em comparação a Go.

5. Conclusão

- C é a melhor escolha para cenários onde o desempenho bruto é crítico e o desenvolvedor tem experiência em gerenciamento manual de memória.

- Go é ideal para aplicações que precisam de simplicidade e legibilidade, com desempenho aceitável.

- Rust oferece um equilíbrio entre desempenho e segurança, sendo uma ótima escolha para sistemas onde a segurança de memória é crucial.

A escolha da linguagem depende do contexto do projeto e das prioridades da equipe (desempenho, segurança, simplicidade, etc.). Para a maioria dos casos modernos, Rust e Go são escolhas mais seguras e produtivas do que C.