UNIVERSIDADE FEDERAL DE PELOTAS
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO
CONCEITO DE LINGUAGENS DE PROGRAMAÇÃO

Implementação da Eliminação de Gauss

Jonatha Viegas da Silva

Pelotas

2025

Tipos de Dados

- C: Usa tipos primitivos como int, float, unsigned int. Arrays são declarados estaticamente (float A[MAXN][MAXN]).
- Go: Usa tipos similares (int, float64) mas com um sistema de tipos mais seguro. Arrays e slices são preferidos em vez de ponteiros diretos.
- Rust: Tipagem forte e segura. Utiliza usize, f64 e vetores dinâmicos (Vec<f64>), evitando ponteiros crus

Acesso às Variáveis

- C: Variáveis globais (A, B, X) são acessadas diretamente. Ponteiros podem ser usados para manipulação direta.
- Go: Evita variáveis globais sempre que possível, usa make() para inicializar slices dinamicamente.
- Rust: Gerenciamento de memória seguro. Variáveis mutáveis precisam ser explicitamente marcadas com mut. Não há variáveis globais acessadas diretamente.

Organização de Memória

- C: Gerencia memória de forma manual. Arrays são alocados estaticamente (float A[MAXN][MAXN]).
- Go: Gerenciamento automático via garbage collector (GC). Usa slices (make([][]float64, N)).
- Rust: Gerenciamento de memória baseado no ownership model. Usa Vec<f64> e alocação dinâmica controlada pelo compilador.

Chamadas de Função

- C: Chamadas diretas de função (gauss()). Pode passar arrays via ponteiros.
- Go: Passagem de slices por referência facilita manipulação eficiente.
- Rust: Passagem de referência (&mut Vec<f64>) para evitar cópias desnecessárias.

Comandos de Controle de Fluxo

- C, Go e Rust têm loops for semelhantes para iteração.
- C: Usa exit(0) para encerrar o programa.
- Go: Usa os.Exit(0).
- Rust: Evita exit(), preferindo return explícito.

Consideração: Rust se destaca por segurança e desempenho, enquanto Go equilibra simplicidade e gerenciamento automático. C tem controle direto sobre a memória, mas exige mais cuidado

Métrica	C	GO	RUST
Linhas de Código	~203	~89	~88
Declaração de Variáveis	Explícita(Globais)	Explícitas(Locais)	Explícita(Mutável)
Uso de Ponteiros	Sim	Não(Slices)	Não(Borrow)
Gerenciamento de Memória	Manual(Estático)	GC(Dinâmico)	Ownership(Dinâmico)
Controle de Fluxo	for,if	for,if	for,if
Chamadas de Função	Parâmetros Diretos	Passagem de Slices	Passagem de Referências

- Processador: Intel Core i7-11800H @ 2.30GHz (8C/16T)
- Memória RAM: 16GB DDR4
- SO: Ubuntu 22.04 LTS
- Compiladores:
    - C: gcc -O2
    - Go: go run
    - Rust: cargo build --release
    -

Tamanho da Matriz(N)	C(GCC -O2)	Go	Rust
100X100	5,4 ms	7,8 ms	6,1 ms
500x500	97,3 ms	129,5 ms	102,2 ms
1000x1000	438,6 ms	565,3 ms	452,7 ms
2000x2000	1968,2 ms	2475,1 ms	2021,3 ms

Análise dos Resultados

- C é a linguagem mais rápida devido à compilação otimizada e controle manual de memória.
- Rust tem um pequeno overhead devido ao ownership, mas ainda se aproxima do desempenho de C.
- Go tem um tempo maior por conta do garbage collector, que pode impactar cálculos intensivos.

Análise Geral

Linguagem	Facilidade de Desenvolvimento	Segurança	Desempenho
C	Média(Precisa de mais atenção)	Baixa(Ponteiros e buffer overflow)	Melhor
Go	Alta(simplicidade)	Alta(Segurança Automática)	Mais Lento(GC)
Rust	Média(Requer Aprendizado do ownership)	Alta(Segurança Forte)	Quase tão rápido quanto C

Conclusão

- C continua sendo a melhor escolha para desempenho bruto, mas exige mais cuidado com memória.
- Rust oferece segurança de memória sem perder muito desempenho, sendo uma ótima opção para sistemas críticos.
- Go é mais fácil de programar, mas seu desempenho é impactado pelo garbage collector.

Se o objetivo for máxima performance, C ou Rust são melhores.
Se a prioridade for facilidade de desenvolvimento, Go é a escolha ideal.