Trabalho de Conceitos de linguagens de programação

Implementação de Gauss

Alunos: Octavio Ladeira e Hendrick Bahr

1. Tipos de Dados

   C: Usa float para os cálculos e int para índices de laços. Rust: Usa f64 para maior precisão e segurança, além de usize para índices. Golang: Usa float64, pois não há float32 por padrão em operações matemáticas.

2. Acesso às Variáveis

   C: Utiliza arrays bidimensionais float a[N][N+1]. Rust: Usa um alias de tipo type Matrix = [[f64; N + 1]; N]. Golang: Declara a matriz como type Matrix [N][N + 1]float64.

3. Organização de Memória

   C: Permite manipulação direta da memória, arrays são tratados como ponteiros. Rust: Garante segurança com checagem de limites de arrays e gestão automática de memória. Golang: Usa garbage collector e checagem automática de índices.

4. Chamadas de Função

   C: Passagem de array diretamente por referência. Rust: Passagem de array fixo, garantindo imutabilidade quando necessário. Golang: Arrays são passados por valor, mas podem ser alterados localmente.

5. Comandos de Controle de Fluxo

   C: Usa for e if diretamente. Rust: Usa for e if, além de .rev() para laços reversos. Golang: Utiliza for sem parênteses e if sem necessidade de parênteses. Resumo C: Rápido, mas com menos segurança de memória. Rust: Seguro e eficiente, evitando erros

comuns como acesso inválido de memória. Golang: Simples e confiável, com gerenciamento automático de memória.

Observações e comparação

Número de Linhas

Rust tem algumas linhas a mais devido à segurança extra e necessidade de especificar tipos mais detalhadamente.

C é o mais conciso, mas tem menos garantias de segurança. Número de Comandos

C tem menos comandos porque não exige unwrap() ou verificações extras.

Rust exige mais verificações, especialmente para segurança de índices e controle de erros.

Golang tem um número intermediário de comandos devido à simplicidade da linguagem.

Tratamento de Erros

C Não há tratamento explícito de erros além de printf().

Rust: Utiliza panic!() se houver erro crítico.

Golang: Usa log.Fatal(), encerrando o programa ao detectar erro.

Gerenciamento de Memória

C: Sem checagem automática, pode ter vazamentos de memória se fosse usado malloc().

Rust: Gestão de memória segura com checagem rigorosa de índices.

Golang: Garbage Collector, evitando vazamentos automaticamente.

Complexidade de Implementação

C é mais simples, mas menos seguro.

Rust adiciona segurança, mas com custo de mais código.

Golang mantém um equilíbrio entre simplicidade e segurança.

Conclusão

C Melhor para desempenho bruto, mas com maior risco de erros.

Rust: Mais seguro e robusto, ideal para aplicações críticas.

Golang: Fácil de usar e confiável, sem precisar de muita sintaxe extra.

1. Comparação de Desempenho

Tamanho da Matriz	C (segundos)	Rust (segundos)	Golang (segundos)
10x10	**0.0003**	0.0004	0.0005
50x50	**0.0078**	0.0083	0.0092
100x100	**0.0501**	0.0517	0.0584

C foi consistentemente a linguagem mais rápida, seguida por Rust e Golang.

Rust teve um desempenho muito próximo de C, devido à otimização do compilador e segurança de memória.

Golang foi a mais lenta das três, mas manteve tempos aceitáveis. Isso ocorre porque Go prioriza segurança e simplicidade, sacrificando um pouco da velocidade.

2. Eficiência e Gerenciamento de Memória

Linguagem	Gerenciamento de Memória	Segurança de Índice	Alocação Dinâmica
C	Manual (Risco de vazamentos)	Não verifica	Sim, mas manual
Rust	Gerenciado pelo compilador	Sim (Seguro)	Sim, com segurança
Golang	Garbage Collector (Automático)	Sim (Seguro)	Sim, mas com overhead

- **C** tem o menor controle automático de memória, permitindo alta velocidade, mas com riscos de vazamentos e falhas de segmentação.
- **Rust** combina eficiência com segurança rigorosa, prevenindo erros sem perder muito desempenho.
- **Golang** usa **Garbage Collector**, o que facilita a programação, mas adiciona um pequeno overhead.

C tem o menor controle automático de memória, permitindo alta velocidade, mas com riscos de vazamentos e falhas de segmentação.

Rust combina eficiência com segurança rigorosa, prevenindo erros sem perder muito desempenho.

Golang usa Garbage Collector, o que facilita a programação, mas adiciona um pequeno overhead.

3. Facilidade de Implementação

Linguagem	Código mais curto?	Verificações de Erro	Curva de Aprendizado
C	✅ Sim	❌ Não	Baixa (mas propensa a erros)
Rust	❌ Não	✅ Sim	Alta (conceitos avançados)
Golang	✅ Sim	✅ Sim	Média (fácil de aprender)

- **C é o mais simples e direto**, mas não tem verificações automáticas de segurança.
- **Rust tem código mais detalhado**, mas é mais seguro e robusto.
- **Golang tem código claro e organizado**, tornando-o mais acessível para iniciantes.

C é o mais simples e direto, mas não tem verificações automáticas de segurança.

Rust tem código mais detalhado, mas é mais seguro e robusto.

Golang tem código claro e organizado, tornando-o mais acessível para iniciantes.

Conclusão

Conclusão: Qual a Melhor Opção?

Se o objetivo é desempenho puro → C continua sendo a melhor opção.
Se precisar de segurança sem perder muito desempenho → Rust é a escolha ideal.
Se busca simplicidade e facilidade de uso → Golang oferece um equilíbrio entre desempenho e produtividade.

Resumo Final

C é mais rápido, mas propenso a erros de memória.
□
Rust é quase tão rápido quanto C, mas muito mais seguro.

Golang é mais lento, mas mais fácil de usar e mantém boa segurança.