

...

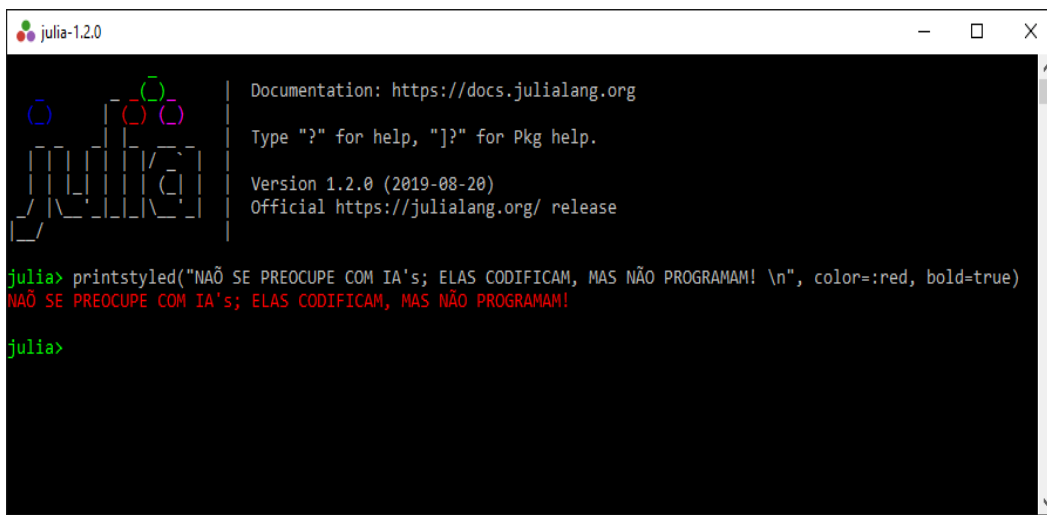
Todos os programadores sabem alguma coisa sobre a linguagem **C**; seu poder de processamento e sua grande variedade de recursos; é uma linguagem compilada e muito poderosa, desenvolvida por *Dennis Ritchie* no início dos anos 1970. Esta linguagem é sempre utilizada quando se deseja mais eficiência dos códigos e menos tempos de processamento em sistemas profissionais. Mas, sobre tempos de processamento, embora não seja páreo direto com **C**, existem outras alternativas com linguagens não totalmente compiladas; é o caso de **Julia**. Esta linguagem foi criada em 2012 por quatro pesquisadores: *Jeff Bezanson*, *Stefan Karpinski*, *Viral Shah* e *Alan Edelman* com o objetivo era unir a facilidade de Python, Matlab e R com a velocidade do C e Fortran em aplicações científicas, estatísticas, e Data Science & Machine Learning, usando **JIT (Just-In-Time)** via LLVM, o que gera código nativo otimizado para a máquina. A conversão do código-fonte para código de máquina é feita como compilação “sob demanda”: apenas quando o código é chamado, garantindo alto desempenho. Por outro lado, é importante frisar que fazer comparação de Julia com C pode ser uma empreitada um pouco ousada, já que ela não é uma linguagem totalmente compilada, e C é. Assim, Julia já parte em desvantagem sob esse aspecto. Entretanto, ela não fica muito atrás de uma linguagem totalmente compilada, como será demonstrado...

Um dos processamentos computacionais mais interessante, e que faz brilhar os olhos dos programadores iniciantes, é acompanhar um *loop* “cuspindo” valores calculados por algum algoritmo matemáticos; por exemplo, exibir números primos em uma determinada faixa de valores. Dependendo da faixa desejada e da quantidade de números, o tempo de processamento pode durar uma eternidade, mesmo empregando algoritmos eficientes. De qualquer forma, encontrar números primos é uma tarefa fundamental na computação e na matemática; com aplicações que vão desde a criptografia até à teoria dos números. A eficiência do algoritmo utilizado e a performance da linguagem de programação são cruciais para essa tarefa, especialmente quando se busca números primos grandes. É o caso relacionando à segurança de informações; por exemplo em algoritmos de criptografia como no método RSA. Neste cenário, foi comparado o desempenho de Julia e C na tarefa de encontrar os primeiros N números primos, com o objetivo de mostrar a performance de cada linguagem para problemas computacionais comuns.

Os resultados obtidos indicam que, para a tarefa de encontrar os primeiros 1000 números primos utilizando o algoritmo de teste de primalidade implementado, C foi ligeiramente mais rápido: (C em 0.219 s e Julia em 0.344 segundos); uma diferença relativamente pequena. A linguagem C, sendo uma linguagem compilada de médio nível, oferece um controle mais direto sobre o *hardware* e a memória, o que frequentemente resulta em um desempenho marginalmente superior para tarefas computacionais intensivas como a apresentada aqui. Julia, por outro lado, é uma linguagem compilada **Just-In-Time (JIT)** que busca combinar a facilidade de uso de linguagens de alto nível com a performance de linguagens de baixo nível. Embora não tenha superado C neste *benchmark* específico, sua performance é notavelmente próxima; especialmente considerando a facilidade de desenvolvimento e a expressividade da linguagem. Para muitos problemas científicos e de engenharia, a produtividade que Julia oferece pode compensar essa pequena diferença de tempo de execução.

É importante ressaltar que o desempenho pode variar dependendo do algoritmo, do *hardware*, do compilador (para C) e da versão da linguagem (para Julia), bem como de otimizações específicas que podem ser aplicadas em cada código. No entanto, este *benchmark* fornece uma boa indicação da capacidade de ambas as linguagens para lidar com problemas de computação numérica. Portanto, para a tarefa de encontrar os primeiros N números primos com o algoritmo utilizado, C demonstrou ser marginalmente mais rápido que Julia. Ambas as linguagens, no entanto, oferecem excelente desempenho para computação numérica, e a escolha entre elas pode depender de outros fatores, como a complexidade do projeto, a facilidade de desenvolvimento e o ecossistema de bibliotecas disponíveis. Julia continua a ser uma forte candidata para aplicações que exigem alta performance e alta produtividade. A **figura 1** mostra um exemplo de linha de código para ser executada no console interativo (REPL) de Julia, mas para com várias linhas de código o programa deve ser escrito em editores *VS Code* ou mesmo no *Bloco de Notas* e executado com o arquivo executável **julia.exe**, como foi feito neste exemplo de exibir os números primos.

```
D:\Livro11\Julia> C:\Users\Usuario\AppData\Local\Julia-1.2.0\bin\julia.exe NumerosPrimos.jl
```



```
julia-1.2.0

Documentation: https://docs.julialang.org
Type "?" for help, "]" for pkg help.
Version 1.2.0 (2019-08-20)
Official https://julialang.org/ release

julia> printstyled("NÃO SE PREOCUPE COM IA's; ELAS CODIFICAM, MAS NÃO PROGRAMAM! \n", color=:red, bold=true)
NÃO SE PREOCUPE COM IA's; ELAS CODIFICAM, MAS NÃO PROGRAMAM!

julia>
```

Figura 1 - Exemplo de código no console interativo (REPL) de Julia

```
#NumerosPrimos.jl

#Função para verificar se um número é primo

function ehPrimo(num::Int)
    if num < 2
        return false
    elseif num == 2
        return true
    elseif num % 2 == 0
        return false
    end
    limite = floor(int, sqrt(num))
    for i in 3:2:limite
        if num % i == 0
            return false
        end
    end
    return true
end

println("Digite o valor de n: ")
n = parse(int, readline())

count = 0
num = 2

inicio = time()

while count < n
    if ehPrimo(num)
        println(num)
        count += 1
    end
    num += 1
end

fim = time()
tempo = fim - inicio
println("\nTempo gasto em Julia: $(round(tempo, digits=6)) segundos")
#Fim do programa "NumerosPrimos.jl" -----
```

```
//NumerosPrimos.C

//Mostra os n primeiros Números Primos

#include <stdio.h>
#include <math.h>
#include <time.h>

// Função que verifica se um número é primo
int ehPrimo(int num) {
    if (num < 2) return 0;
    if (num == 2) return 1;
    if (num % 2 == 0) return 0;
    int limite = sqrt(num);
    for (int i = 3; i <= limite; i += 2) {
        if (num % i == 0) return 0;
    }
    return 1;
}

int main() {
    int n, count = 0, num = 2;
    clock_t inicio, fim;
    double tempo;

    printf("Digite o valor de n: ");
    scanf("%d", &n);

    inicio = clock();

    while (count < n) {
        if (ehPrimo(num)) {
            printf("%d\n", num);
            count++;
        }
        num++;
    }
    fim = clock();
    tempo = ((double)(fim - inicio)) / CLOCKS_PER_SEC;
    printf("\nTempo gasto em C: %.6f segundos\n", tempo);

    return 0;
} //Fim do programa "NumerosPrimos.C" -----
```

```
Prompt de Comando

D:\>cd Cantinho da Programação\Códigos\Julia

D:\Cantinho da Programação\Códigos\Julia>C:\Users\Usuario\AppData\Local\Julia-1.2.0\bin\julia.exe NumerosPrimos.jl
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
101
```

...

```
Prompt de Comando

7727
7741
7753
7757
7759
7789
7793
7817
7823
7829
7841
7853
7867
7873
7877
7879
7883
7901
7907
7919

Tempo gasto em Julia: 0.344 segundos

D:\Cantinho da Programação\Códigos\Julia>
```

Figura 1 - Saída do programa "NumerosPrimos" (em Julia)

```
D:\Cantinho da Programação\C\digos\C\NumerosPrimos.exe
Digite o valor de n: 1000
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
101
```

...
...

```
D:\Cantinho da Programação\C\digos\C\NumerosPrimos.exe
7699
7703
7717
7723
7727
7741
7753
7757
7759
7789
7793
7817
7823
7829
7841
7853
7867
7873
7877
7879
7883
7901
7907
7919

Tempo gasto em C: 0.219000 segundos

-----
Process exited after 3.91 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 2 - Saída do programa "NumerosPrimos" (em C)