Relatório de Aula Prática – Conjectura de Collatz

Luiz Antônio Rangel  
*Universidade Norte do Paraná – Polo Gramado*

*Gramado, estado do Rio Grande do Sul, Brasil, outubro de 2023*  
  
Introdução

Nesse documento pretendo mostrar como implementar uma demonstração da Conjectura de Collatz ̶ a qual também me referi como “Problema de Siracusa” ̶ na linguagem de programação C.

Como o próprio nome diz, esse questionamento foi introduzido ao mundo pelo matemático alemão Lothar Collatz no ano de 1937, dois anos após ter finalizado seus estudos de doutorado em 1935. Mesmo tendo produzido soluções, questionamentos e fundamentos mais importantes que a supracitada conjectura, ela é um marco em sua carreira pois de fato intrigou diversos matemáticos que tentaram resolvê-la ao longo da história utilizando diversos métodos e, além disso, mostrou que em ramos como a Teoria de Números ainda haveriam “buracos” que precisariam ser preenchidos ou, ao menos, reconhecidos como axiomáticos e sem prova.

Esse problema tem, curiosamente, diversos sinônimos, como “O problema ”, o algoritmo de Hasse, Conjectura de Ulam (na Teoria de Números) e por aí vai. Cada matemático que contribuiu para o problema acabou por ter seu nome atribuído a ele, incluindo instituições como a Universidade de Siracusa, na cidade de Siracusa, no estado de Nova Iorque, nome esse que foi proposto por Helmut Hasse ao problema em uma de suas visitas à instituição nos anos 50.

Esse problema é algo icônico dentro da Matemática pois, mesmo parecendo trivial, ele não tem uma prova completa até os dias atuais, no máximo demonstrações feitas na “força bruta” ̶ pedindo perdão prévio pela tradução literal de “*bruteforce*” ̶ utilizando computadores modernos para testar se o algoritmo, em algum ponto, não “quebra” com um número maior.

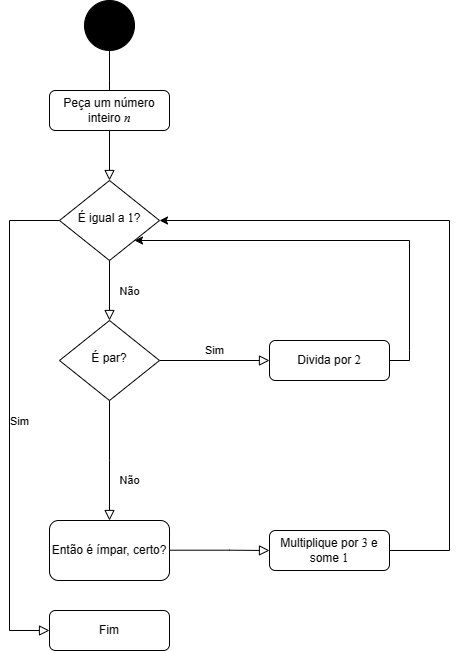
É chamado “conjectura” pois, mesmo sem uma prova rigorosa, ele é verdadeiro até onde sabemos por conta desses testes e de conhecimentos em álgebra ̶ como, por exemplo, o de que um número ímpar multiplicado a um número ímpar continua sendo ímpar e que, se pegarmos esse número ímpar e somarmos a 1, ele se torna par.

Método

A partir da cadeira de Algoritmos e Lógica de Programação no 2º semestre do curso de Engenharia da Computação de 2023, nós, enquanto alunos, passamos a ter acesso a conteúdos sobre a Linguagem de Programação C, essa que, como eu disse na Introdução, será utilizada para demonstrar a Conjectura de Collatz na prática.

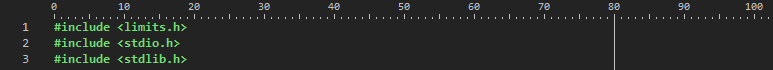
Por sorte, além das aulas serem intuitivas, eu já havia programado em C antes e, além disso, havia implementado o mesmo problema em 2021, a fim de mostra-lo à minha professora de matemática na época, durante as minhas férias de verão entre o 2º e 3º ano do Ensino Médio, na linguagem KornShell 93, que é similar a C em diversos aspectos, então reescrever o código acabou por ser uma tarefa trivial, mas, ainda sim, interessante.

O algoritmo é trivial: Temos um número , e, para esse número, oferecemos duas condições,

com o único diferencial sendo que iremos repetir esse algoritmo até que tenhamos 1 como retorno, valor esse que é o fim da sequência 4, 2 e 1 que o algoritmo nos retorna.  
“Passando” esse algoritmo para um diagrama, temos esse resultado:  


Então, vamos ao código em C.

Primeiramente, por uma formalidade da linguagem, devemos incluir os cabeçalhos limits.h ̶ que provê, basicamente, *macros* com indicações sobre quais os tamanhos mínimo e máximo que um determinado tipo pode ter, analogamente a uma tabela ̶ , stdio.h ̶ que significa, em tradução, “entrada/saída padrão”, e que, como o nome diz, nos provê funções para lidar com entrada e saída ̶ , e, por fim, stdlib.h ̶ o que se traduziria como “biblioteca padrão” e que provê funções diversas, nesse caso estaremos utilizando a função exit().

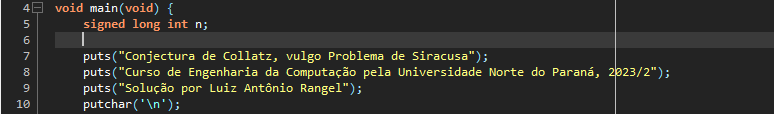


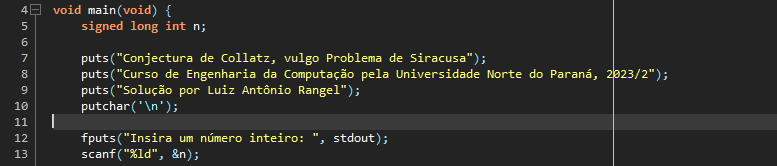
Em seguida, declaramos a função que conterá o algoritmo descrito. Ela terá o nome de “main” por conta do padrão da linguagem que requer que uma função com esse identificador esteja presente para que chame as outras pois, como o próprio nome diz, é a função principal.  
Utilizaremos o tipo “void” ̶ que poderíamos traduzir como “vazio”, “nulo” ̶ como uma espécie de *stopper* , uma rolha, fazendo uma breve analogia com instrumentos utilizados na Química e na antiga Alquimia, para dizer explicitamente ao compilador que nossa função não irá retornar nada ̶ pois já estaremos imprimindo na tela ̶ e que ela não receberá nenhum parâmetros ̶ pois, também, já estaremos pedindo a entrada do usuário no tempo de execução, não dentro do código ̶ , assim blindando o programa final de algum tipo de falha de segurança. Isso é algo difícil de acontecer na prática, mas se mostra uma boa prática ao mostrar noção do que você está fazendo.

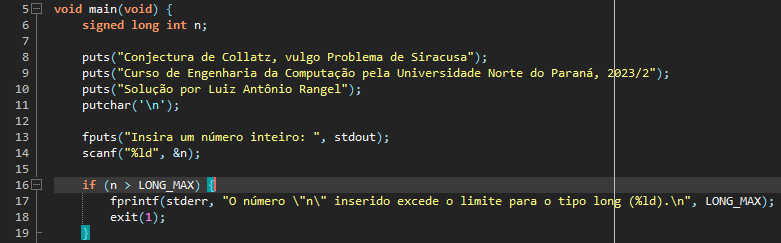


Em seguida, declaramos que iremos utilizar um número “” do tipo “inteiro longo com sinal” (signed long int), esse que, na arquitetura i86 de 64-bit, pode ir até , número, de fato, gigantesco e que funcionaria bem para demonstrar até onde a conjectura se aplica.

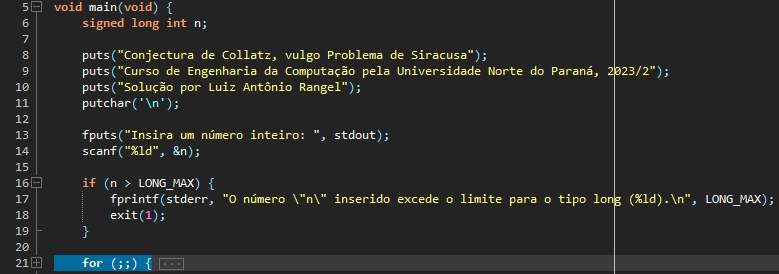


Por uma simples formalidade, iremos nos apresentar ao usuário antes de executar as próximas instruções do programa.  
  
  
Agora, nós pedimos ao usuário por um valor a ser calculado utilizando a função scanf(), que poderia ter seu identificador traduzido como ""escaneie" um valor com um formato específico".

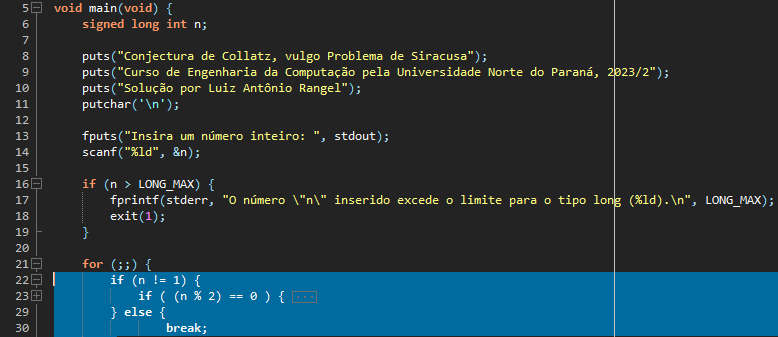
Essa função funciona tendo o formato em que o valor digitado deve ser formatado ao lado do endereço na memória da variável em que desejamos guardar o valor digitado.  
  
Puxando uma tangente, em minha humilde opinião, seria mais interessante se pudéssemos ter o valor digitado retornado pela própria função e salvo na variável por meio de uma associação padrão, e não por meio de um ponteiro.

Em seguida ̶ por mais que uma implementação mais, digamos, profissional ou “endurecida” (*hardened*) da função scanf() consiga detectar um problema do tipo ̶ nós iremos verificar se está fora dos limites estabelecidos para o seu tipo, ou seja, se é maior que o valor de LONG\_MAX. Se for, nós iremos imprimir uma mensagem descrevendo o problema ao usuário e sairemos do programa com como estado de erro. 

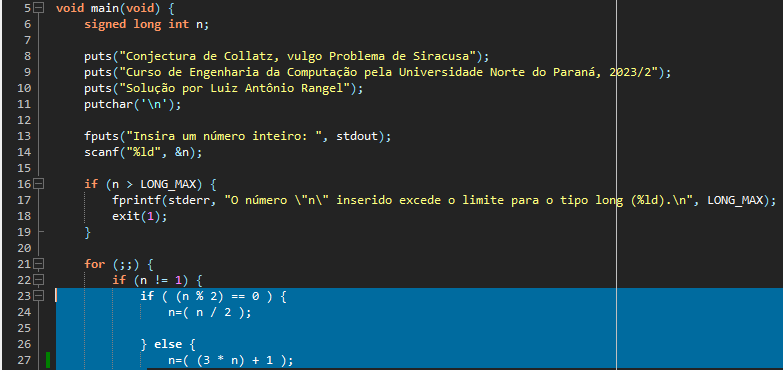
Se tudo estiver nos conformes, iremos à parte principal do algoritmo.  
Como expliquei anteriormente, enquanto não for igual a , nós iremos repetir o algoritmo com o último valor de . Isso pode ser feito de -formas (sem trocadilhos aqui) dependendo da criatividade ou do desafio que quisermos impor a nós mesmos enquanto programadores ou, ao menos, hobbystas. Nesse caso, trazendo uma bagagem do estilo de código clássico da AT&T e, também, da linguagem Google Go, decidi utilizar um laço para (for) infinito ̶ ou seja, sem cláusulas definidas que fariam ele parar em algum ponto.



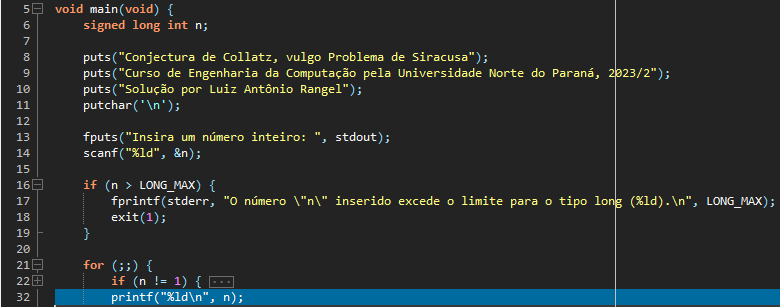
Logo em seguida, nós adicionamos um fluxo de controle de condição “e se” (if) dentro desse laço, a fim de podermos pará-lo em algum momento.  
Nesse caso, nós iremos apenas “quebrar” (break), ou seja, parar o laço quando for igual a .



Caso (ainda) não seja igual a , nós iremos abrir a condição antes descrita no começo desse texto que, sendo extremamente prolixo e descrevendo novamente “se for par, divida por , se for ímpar, multiplique por e some ”:



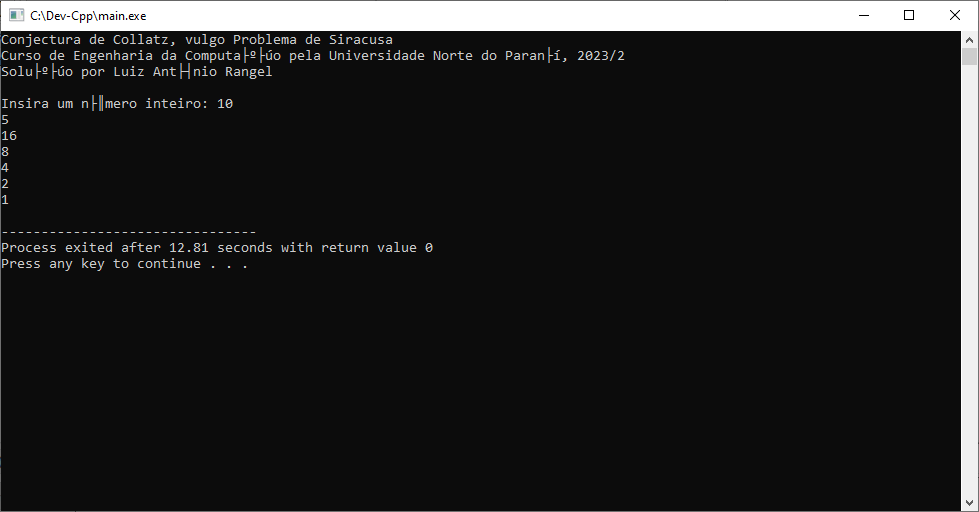
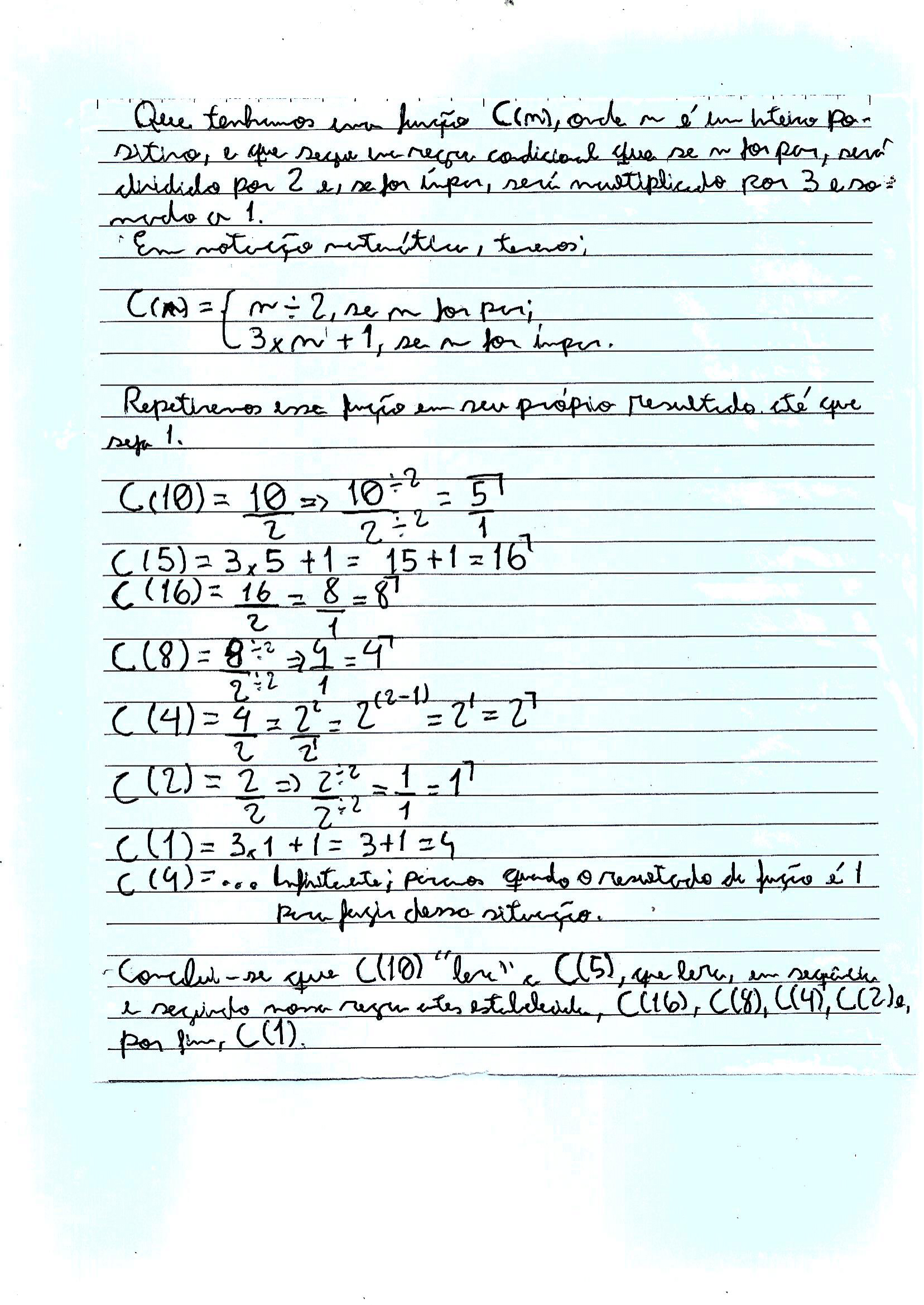
Por fim, que apenas imprimamos o valor ao nosso usuário independente das condicionais anteriormente apresentadas dentro desse laço:

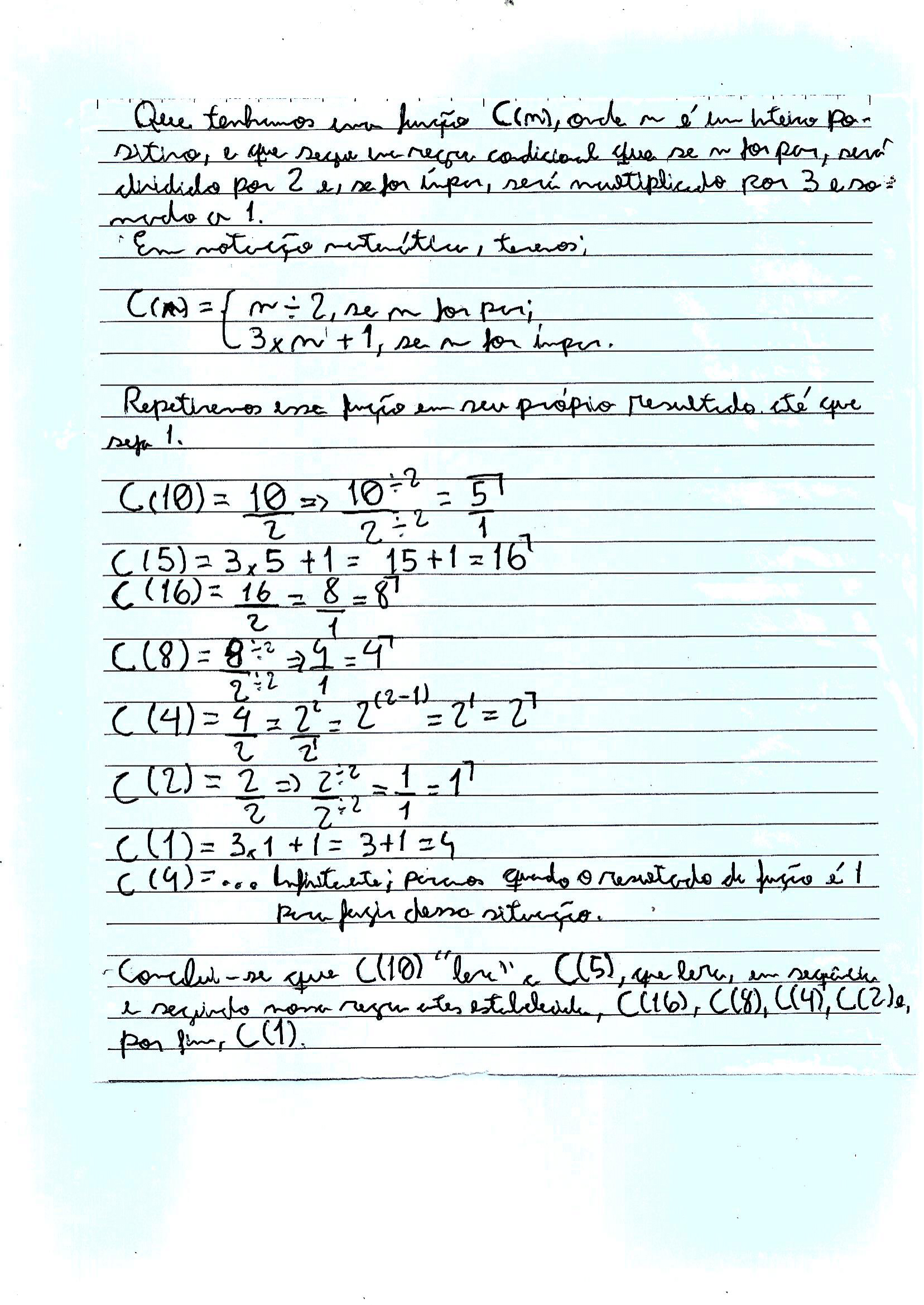


Resultados

Após salvar o código e compilar para o programa final, apesar dos erros de codificação dos caracteres ̶ pois, aparentemente, o Windows NT ainda não suporta UTF-8 ̶, o programa funciona como esperado.

Para fins de “prova-real”, executei a função no número 10 no programa e também calculei manualmente, e os resultados se mostraram precisamente iguais, indicando que o algoritmo foi implementado de forma correta.

  
  
Conclusão  
Esse exercício se mostrou interessante tanto ao demonstrar como aplicar diversas estruturas de decisão e de repetição e como representar essas estruturas em um fluxograma. Além disso, também podemos ver como computadores são utilizados para demonstrar e, digamos, “testar” limites de conjecturas, teoremas e teorias matemáticas, acelerando trabalhos de pesquisa na área.   
  
O código-fonte completo para esse programa está disponível na minha conta pessoal no GitHub: [takusuman/siracusa-2023-2: Minha solução para a Aula Prática I do curso de "Algoritmos e Lógica de Programação" da Universidade Norte do Paraná (github.com)](https://github.com/takusuman/siracusa-2023-2)

Referências

Lagarias, Jeffrey C.  The 3x + 1 Problem and its Generalizations. Murray Hill, Nova Jérsei: AT&T Bell Laboratories, The American Mathematical Monthly, 1985.

.  
Rangel, Luiz Antônio. “Uma Implementação Do Problema de Siracusa (Vulgo ‘Problema Do 3x + 1’) Em Korn Shell.” GitHub Gist, 3 de fevereiro, 2022. https://gist.github.com/takusuman/c72f1ae1e397bf613a09c2b0f08b0e06.

Schildt, Herbert. C - Completo e Total. Translated by Roberto Carlos Mayer. 3rd ed. Vol. 1. 1 vols. São Paulo, São Paulo: Pearson Makron Books, 1997.