A arte de escrever algoritmos

Metodologia PPCT: pensar, planejar, codificar e testar.

Renato Hidaka Torres

Sumário

1 Introdução **1**

1.1 O que é um programa? 2

1.2 O que é programar? 4

1.2.1 Módulo de interação 5

1.2.2 Módulo de processamento 8

1.2.3 Módulo de armazenamento 11

1.3 Resumo 12

1.4 Exercício 13

2 Fluxo de dados no computador **15**

2.1 Arquitetura de Von Neumann 16

2.2 Escrita e leitura na memória 20

2.3 Entrada e saída de dados 27

2.4 Resumo 30

2.5 Exercício 31

3 Operadores aritméticos **33**

3.1 Potenciação 35

3.2 Radiciação 37

3.3 Logaritmo 38

3.4 Resto da divisão 39
3.5 Exercício 42

4 Processamento aritmético 45
4.1 Representação da entrada de dados 46
4.2 Representação da saída de dados 48
4.3 Metodologia PPCT 51
4.4 Questão 1: Busca na internet 53
4.5 Questão 2: Pedágio 57
4.6 Questão 3: Tomadas 64
4.7 Questão 4: Corrida 68
4.8 Exercício 72

5 Operadores lógicos 83
5.1 Expressão lógica simples 84
5.2 Expressão lógica composta 86
5.3 Exercício 93

6 Controle de fluxo de seleção 97
6.1 Controle de fluxo de seleção SE-SENÃO 99
6.2 Controle de fluxo de seleção SE-ENTÃO 102
6.3 Controle de fluxo SE-SENÃOSE 105
6.4 Questão 1: Receita de Bolo 108
6.5 Questão 2: Detectando Colisões 115
6.6 Questão 3: Alarme Despertador 120
6.7 Exercício 126
6.8 Saída 151

7 Controle de fluxo de repetição 153

8 Vetores e Matrizes 155

9 Preparando o ambiente Python 157

10 Comandos básicos de Python 159

Capítulo 1

Introdução

1.1 O que é um programa?

Antes de mais nada, se você está lendo este livro, podemos definir programa como o que você está querendo aprender a construir.

Diariamente você acessa o seu e-mail, navega pela internet, utiliza aplicativos de celular, produz documentos de texto dentre outras tarefas similares. Para realizar suas tarefas diárias utilizando um computador você precisa utilizar programas.

Se você deseja navegar pela internet pode, por exemplo, utilizar programas como o Google Chrome ou o Mozilla Firefox. Para criar documentos de texto você pode utilizar programas como o Microsoft Word ou LibreOffice. Se você gosta de jogar, certamente, já instalou e passou noites utilizando programas de entretenimento. Não importa qual a sua necessidade, se você deseja resolvê-la utilizando um computador, então, precisará de um programa.

Um programa pode ser definido como a parte lógica do computador a qual é necessária para que você consiga realizar suas atividades. Todo programa, independentemente da sua finalidade, tem um fluxo comum de utilização e de componentes.

Conforme a figura 1.1, o fluxo comum de utilização do programa inicia com o usuário passando informações a partir do módulo de entrada. Estas informações são utilizadas para determinar quais processamentos devem ser realizados tal que gere um resultado e este seja entregue ao usuário pelo módulo de saída.

Dependendo das informações passadas pelo usuário, pode ser que o processamento necessite guardar ou recuperar informações do módulo de armazenamento permanente. Dentre as ações típicas que acionam o módulo de armazenamento temos: salvar, deletar, atualizar e pesquisar.

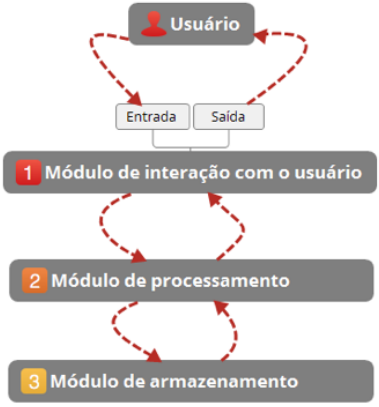


Figura 1.1: Fluxo de utilização de um programa.

Quando você está jogando e deseja salvar para continuar o jogo no outro dia, neste momento, o programa precisa acionar o módulo de armazenamento permanente. Obviamente, no outro dia, quando você for continuar o jogo, o programa também acionará o módulo de armazenamento permanente para recuperar seus dados salvos.

Em relação aos componentes, todo programa deve ter os módulos de interação com o usuário, de processamento e, na maioria dos casos, de armazenamento permanente (ver figura 1.2).

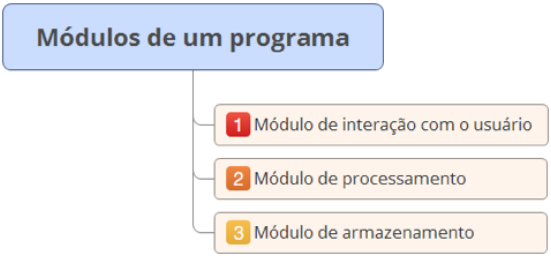


Figura 1.2: Componentes de um programa.

O componente de interação com o usuário se resume as interfaces gráficas que você utiliza para interagir com o programa. Este componente pode ser ramificado em módulo de entrada e módulo de saída.

O objetivo do módulo de entrada é receber, do usuário, as informações necessárias para realizar uma determinada tarefa. Já o objetivo do módulo de saída é apresentar, para o usuário, o resultado do processamento o qual atende a tarefa. Entende-se por tarefa uma determinada funcionalidade realizada por um programa. Dentre as funcionalidades mais comuns temos: salvar, pesquisar, deletar, atualizar e calcular.

O segundo componente que todo programa deve ter é o de processamento. O componente de processamento é formado por todas as funcionalidades que o programa pode realizar. Este componente é responsável por realizar as tarefas do programa levando em consideração as informações recebidas do módulo de entrada. Construir as funcionalidades do programa, na maioria das vezes, implica na parte mais complexa do desenvolvimento.

Por fim, o terceiro componente diz respeito ao de armazenamento permanente. Este componente é necessário quando o programa precisa salvar as informações no HD. Um programa que salva as informações no HD geralmente utiliza um sistema gerenciador de banco de dados (SGBD). Dentre os SGBDs mais conhecidos temos o Mysql, Postgre, Oracle e SQLServer.

1.2 O que é programar?

Programar é o que você precisa aprender para construir programas de computador.

Você já sabe que um programa pode ser visto como a junção de três componentes: interação com o usuário, processamento e armazenamento permanente. Desta forma, se você deseja

construir um programa de computador, terá que programar cada um desses componentes.

1.2.1 Módulo de interação

Para programar o módulo de interação com o usuário, primeiramente, deve ser definido em qual ou em quais plataformas o seu programa deverá funcionar. As plataformas de interação de um programa podem ser classificadas em três categorias: Desktop, Web e Mobile.

O desenvolvimento do módulo de interação para plataformas Desktop é fortemente dependente da linguagem de programação utilizada. Desenvolver programas para este tipo de plataforma está cada vez mais raro. Se você for parar para pensar, a maioria dos programas que você utiliza hoje ou é Web ou é Mobile.

Para desenvolver programas cujo módulo de interação seja Web, provavelmente, você terá que estudar HTML, CSS e JavaScript. Dominar esta trinca é fundamental para que você consiga desenvolver programas web com elevado grau de usabilidade e fluidez.

Usabilidade pode ser definido como o grau de facilidade que o usuário conseguirá utilizar o seu programa. Se um usuário, mesmo que inexperiente, consegue utilizar um programa com facilidade é porque este programa possui elevado grau de usabilidade.

Em relação a fluidez, podemos definir como a capacidade que a interface do seu programa terá em se adaptar a diferentes dispositivos. Se a interface de um programa não se desconfigura quando é aberta em um notebook, tablet, ou smartphone, então, é porque possui elevado grau de fluidez.

Se você já pretende dominar o desenvolvimento de interfaces web, posso lhe recomendar a leitura do livro HTML e

CSS3 de Lucas Mazza assim como o livro Dominando JavaScript com jQuery de Plínio Balduino.

Em relação a plataforma mobile, se você deseja desenvolver programas que operem nesta plataforma, provavelmente terá que estudar XML no caso dos dispositivos Android.

Vale ressaltar que os conceitos de usabilidade e fluidez também são válidos para o desenvolvimento em plataformas mobile. Se você deseja saber mais sobre este assunto lhe recomendo a leitura do livro Web Design Responsivo de Tárcio Zemel.

Antes de continuarmos quero deixar claro uma coisa. Considerando que um programa é formado pelos módulos de interação com usuário, processamento e armazenamento é normal que os estudos de computação se inicie pelos pré-requisitos para o desenvolvimento do módulo de processamento.

Entretanto, a evolução dos estudos desses pré-requisitos é lenta e, por este motivo, um tanto quanto frustrante. Em média um aluno precisa de um ano e meio a dois para dominar a lógica de programação e aprender uma linguagem de programação.

Neste processo de aprendizado os resultados são lentos e por não desenvolver as habilidades para construir interfaces com o usuário os programas desenvolvidos não são atrativos.

Resumindo, se você seguir a linha clássica de aprendizado você precisa ter paciência pois a curva de aprendizado para desenvolver o módulo de processamento de um programa é suave.

Por outro lado, aprender a desenvolver o módulo de interação com o usuário é mais simples. Em média um aluno precisa de no máximo um ano para já conseguir a produzir interfaces web de qualidade.

Neste momento você deve estar se perguntando, se é mais fácil aprender a desenvolver o módulo de interação com o usuá-

rio então por que não começar meus estudos por este módulo? Por que as faculdades ensinam primeiro os pré-requisitos para o desenvolvimento do módulo de processamento?

Realmente essas são ótimas perguntas. Inicialmente vou tentar justificar a segunda pergunta levando em consideração a visão das faculdades.

Começar os estudos de computação pelos pré-requisitos para o desenvolvimento do módulo de processamento é considerado o modelo clássico de ensino. A justificativa de iniciar os estudos por este módulo é que o processamento é núcleo de qualquer programa. O processamento de um programa determina quais as funcionalidades que o programa pode realizar.

Ainda na visão das faculdades, a justificativa de não iniciar os estudos pelos requisitos para o desenvolvimento do módulo de interação com o usuário é que não faz sentido desenvolver interfaces que não tenha funcionalidade. Para desenvolver as funcionalidades você precisa programar o módulo de processamento.

Bom, para você decidir por onde começar os seus estudos você deve se fazer a seguinte pergunta. O que é menos frustrante: aprender a construir programas com funcionalidades complexas, mas com interfaces pouco atrativas ou aprender a construir interfaces com elevado grau de usabilidade e fluidez, mas que não tenham funcionalidade alguma?

Se você optar pela primeira opção o seu estudo será o estudo clássico, você aprenderá a desenvolver programas de dentro para fora (ver figura 1.2). Caso a sua escolha seja a segunda, você aprenderá a desenvolver programas de cima para baixo (ver figura 1.2). Não foi mencionado, mas se optar iniciar seus estudos pelo módulo de armazenamento você aprenderá a desenvolver programas de baixo para cima (ver figura 1.2).

Na minha opinião, para suavizar a frustração dos alunos e acelerar o processo de aprendizagem, acredito que as faculda-

des poderiam abordar concomitantemente o ensino do módulo de processamento e de interação com o usuário. Esta visão não é particular, Paulo Silveira e Adriano Almeida também compartilham deste pensamento uma vez que publicaram o livro Lógica de Programação: Crie seus primeiros programas usando JavaScript e HTML.

Neste livro Paulo e Adriano tentam ensinar os conceitos básicos de lógica de programação e de desenvolvimento de interfaces web. Porém apesar de ter mencionado que as faculdades devem abordar concomitantemente o ensino do módulo de processamento e de interação com o usuário, não compartilho da ideia de que esses temas devem ser abordados na mesma disciplina.

Acredito que um aluno no primeiro ano de curso possa ter uma disciplina de lógica de programação e outra de desenvolvimento de interfaces. No caso do livro publicado por Paulo Silveira e Adriano Almeida, creio que seja uma ótima referência para a disciplina de desenvolvimento de interfaces e que este que você está lendo seja uma ótima referência para a disciplina de lógica de programação.

1.2.2 Módulo de processamento

O módulo de processamento é formado pelas funcionalidades do programa. Para desenvolver as funcionalidades é necessário que você tenha o conhecimento de duas coisas: de lógica de programação e de uma linguagem de programação.

Você precisa aprender uma linguagem de programação para poder, a partir de comandos, programar as funcionalidades do seu programa. As funcionalidades de um programa nada mais são do que uma série de passos bem definidos.

Para desenvolver as funcionalidades de um programa ou resolver um problema computacional você terá que passar pelas

etapas que chamo de PPCT.

1. Pensar.
2. Planejar.
3. Codificar.
4. Testar.

Antes de fazer qualquer coisa você precisar pensar, refletir em como resolver determinado problema. Em computação é normal que o desenvolvimento de uma determinada funcionalidade tenha mais de uma forma, neste contexto, além de pensar em como resolver o problema é interessante que você pense qual a melhor forma para resolve-lo.

O próximo passo consiste em planejar. Nesta etapa você deve detalhar passo a passo a solução do seu problema. Toda vez que você for planejar a solução de um problema computacional você deve particionar o seu problema em três fases.

Inicialmente deve descrever os dados de entrada. Os dados que o problema ou a funcionalidade precisa receber do usuário para realizar o processamento. Em seguida, na segunda fase você irá descrever como o processamento deve ser realizado. Lembre-se, para descreve esta etapa é fundamental que você tenha encontrado a solução na fase do Pensar.

Por fim, a terceira fase da etapa do planejamento consiste em descrever os dados de saída. Os dados de saída são as informações que o processamento gerou e deve ser visualizada pelo usuário.

Resumindo a etapa de planejamento tem como objetivo detalhar a solução encontrada na fase de pensamento. Para que isso ocorra você deve particionar o problema em entrada, processamento e saída. A construção do planejamento é realizada a partir da escrita de algoritmos, ou seja, se você quer planejar

a construção de uma determinada funcionalidade computacional então você deve escrever um algoritmo.

A terceira etapa do PPCT consiste na codificação do algoritmo. Neste momento você escolher uma linguagem de programação para transformar o seu planejamento em algo que o computador consiga entender. A partir da codificação você estará construindo o módulo de processamento do deu programa.

Neste momento você deve está se perguntando, dentre a várias linguagens de programação, qual a melhor? Costumo responder esta pergunta afirmando que a melhor linguagem de programação é aquela que você domina. Não importa se a linguagem é A, B ou C, se você conhece e domina uma determinada linguagem de programação então conseguirá construir excelentes programas com ela.

Também costumo falar que depois que você aprende uma linguagem de programação a prender a segunda, independentemente de qual seja, torna-se um processo mais fácil. É mais fácil porque durante o processo de aprendizado da primeira linguagem você aprendeu duas coisas. A linguagem propriamente dita e também aprendeu a aprender uma linguagem de programação. Este processo é similar para o aprendizado de idiomas.

Neste livro iremos lhe ensinar a codificar seus algoritmos utilizando a linguagem de programação Python. Python é uma linguagem interpretada criada pelo holandês Guido Van Rossum no final dos anos oitenta e início dos anos noventa. A justificativa para adotarmos esta linguagem é que sua sintaxe é simples e normalmente o número de linhas de código para codificar uma determinada funcionalidade é menor em Python do que em outras linguagens tradicionais como C, C++ ou Java.

Além disso, como divulgado por Philip Guo no site da

ACM, desde 2014 Python é a linguagem mais utilizada pelas universidades para ensinar lógica de programação.

Por fim, a última etapa do PPCT consiste em testar a codificação realizada. Neste momento você elaborar os chamados casos de teste e verificar se os resultados produzidos estão conforme os resultados esperados.

A etapa de teste é fundamental para verificar possíveis erros de lógica que por venturam venham a existir no seu código. Outra circunstância é que o teste nos proporciona a revisão do código o que na maioria das vezes contribui para a otimização do mesmo.

Ensinar lógica de programação para que você adquira as habilidades necessárias para construir o módulo de processamento dos programas é o principal objetivo deste livro. Para alcançar este objetivo iremos lhe ensinar como tirar o melhor proveito das etapas do PPCT (pensar, planejar, codificar e testar). Você irá perceber que a arte de programar requer muita dedicação e paciência, mas que o final é extremamente prazeroso e recompensador.

1.2.3 Módulo de armazenamento

O módulo de armazenamento permanente de um programa, normalmente, é construido a partir de um sistema gerenciador de banco de dados (SGBD). Dentre os SGBD's mais utilizados no mercado temos o Oracle, Microsoft SQL Server, PostgreSQL e MySQL.

O principal benefício da utilização de um SGBD é o ganho de performance e estruturação dos dados. Hoje, a maioria dos SGBD's armazenam os dados utilizando o paradigma de entidade e relacionamento.

Outro benefício de se utilizar um SGBD está na padronização da linguagem utilizada. Desde de a década de oitenta

os sistemas de gerenciamento de banco de dados utilizam de forma padronizada a linguagem Structured Query Language (SQL) para a manipulação dos dados. As tarefas mais comuns que realizamos em um banco de dados são: salvar, atualizar, deletar, pesquisar, pesquisar, pesquisar e pesquisar.

Apesar da padronização da linguagem SQL é comum que os SGBD's tenham suas particularidades na linguagem. Tais particularidades geram um problema conhecido como vendor lock-in. O vendor lock-in torna os clientes ou usuários dependentes da tecnologia utilizada.

Como cada SGBD tem diferenças, apesar de sutis, em sua SQL, acaba dificultando uma possível migração de um SGBD para outro. Essa dificuldade de migração é definida como vendor lock-in.

Se você deseja saber mais da história dos SGBD's e estudar a linguagem SQL, lhe recomendo a leitura do livro Sistema de Banco de Dados de Korth, Silberschatz e Sudarshan.

1.3 Resumo

Neste capítulo você teve uma noção introdutória do que é um programa de computador e o que é necessário para construí-lo. Aprendeu que um programa pode ser visto como a junção de três módulos. Módulo de entrada, de processamento e de armazenamento.

Você já sabe também que este livro tem como objetivo lhe ensinar os requisitos necessários para construir o módulo de processamento de um programa. Tais requisitos serão apresentados utilizando a metodologia PPCT (pensar, planejar, codificar e testar).

1.4 Exercício

1. Conforme explanado neste capítulo, qual as três partes em que um programa pode ser dividido?
2. Este livro tem o propósito de lhe ensinar os requisitos para que você seja capaz de construir o módulo de processamento de um programa. Qual a metodologia que será adotada neste livro?
3. Pesquise e defina o conceito de algoritmo.
4. Ao construir um algoritmo quais as três etapas de sua construção?
5. Descreva o fluxo de utilização de um programa apresentado na figura 1.1.
6. Quais os cinco programas que você mais utiliza e quais suas plataformas?
7. Em que módulo de um programa devemos desenvolver a sua interface gráfica?
8. Qual o módulo do programa faz uso de um sistema gerenciador de banco de dados?
9. Qual a finalidade de um SGBD?
10. Quais as tecnologias mais utilizadas para a construção de interfaces web?

Capítulo 2

Fluxo de dados no computador

2.1 Arquitetura de Von Neumann

John von Neumann foi um matemático, físico e cientista muito famoso que viveu na primeira metade do século XX. Nasceu em Budapeste na Hungria e posteriormente foi naturalizado americano. Foi professor da universidade de Princeton e realizou contribuições relevantes em teoria dos conjuntos, mecânica quântica, teoria dos jogos, estatística e na ciência da computação.

Na ciência da computação podemos destacar a participação de von Neumann na construção do ENIAC e na definição da arquitetura do computador digital. Sua definição de arquitetura de computador digital é tão consolidada que até hoje é a base dos computadores modernos e, por esse motivo, ficou conhecida como arquitetura de von Neumann. Se você deseja conhecer a vida deste importante cientista e suas contribuições para o desenvolvimento da computação recomendo a leitura do livro John von Neumann and the Origins of Modern Computing de William Aspray.

Na arquitetura de von Neumann, para que um dispositivo eletrônico seja considerado um computador, é necessário que o mesmo tenha capacidade de processamento lógico e aritmético, armazenamento temporário e permanente e dispositivos de entrada e saída. A figura 2.1 ilustra os componentes da arquitetura de von Neumann.

A função dos dispositivos de entrada e saída é permitir a interação entre usuário e máquina. A partir dos dispositivos de entrada o usuário envia as instruções que serão interpretadas para determinar um processamento e posteriormente gerar uma resposta a qual é recebida pelo usuário via os dispositivos de saída. Os principais dispositivos de entrada utilizados por um usuário são o teclado, o mouse e até mesmo a tela para os dispositivos providos da tecnologia touch screen.

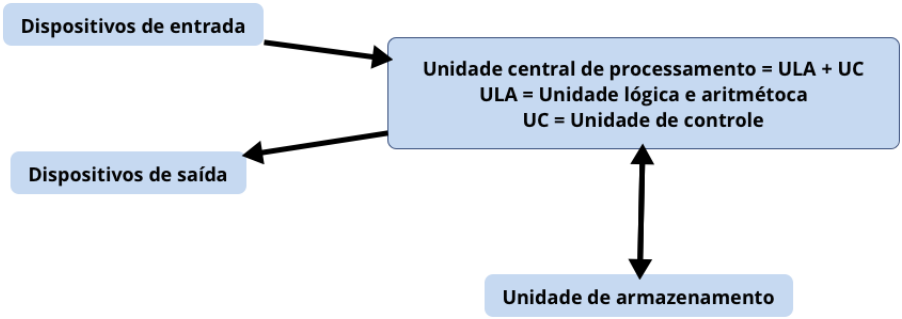


Figura 2.1: arquitetura de Von Neumann

Quanto aos dispositivos de armazenamento, von Neumann destacou que há a necessidade de dois tipos de memória, as de armazenamento permanente, classificadas tecnicamente como memórias não voláteis e as de armazenamento temporário que são classificadas como memórias voláteis.

Em um computador, normalmente, o disco rígido (HD) é a memória não volátil a qual tem a capacidade de armazenar dados mesmo na ausência de energia. Já as memórias voláteis, que só conseguem armazenar dados enquanto houver energia, são representadas pelas memórias RAM e cache.

Sabemos que a memória não volátil, quando comparada com a memória volátil, tem maior capacidade de armazenamento e em geral é mais barata. Sendo assim você poderia se perguntar: por que os computadores precisam das memórias voláteis? A resposta curta e grossa para essa pergunta é porque as memórias voláteis são mais rápidas e por esse motivo diminuem o overhead do processador.

Para você entender a resposta curta e grossa do parágrafo anterior vamos considerar a seguinte analogia. Imagine que você está preparando uma feijoada ou, se você for paraense, uma maniçoba para 100 pessoas. Certamente você vai precisar utilizar um caldeirão tanto para preparar quanto para guardar

a feijoada depois de pronta.

Considerando que as 100 pessoas estão sentadas em mesas espalhadas por um salão e extremamente famintas eu te pergunto: é mais fácil e rápido servir essas pessoas levando o caldeirão, extremamente pesado, de mesa em mesa ou pegar várias bandejas com pequenas tigelas e levar até as mesas? Provavelmente você preferiria servir as pessoas utilizando as tigelas.

Figura 2.2: analogia das tigelas

Nesta analogia o caldeirão representa a memória não volátil e as tigelas a memória volátil. Observe que o caldeirão nos permite armazenar uma grande quantidade de dados (feijoada), porém, é extremamente difícil e demorado ficar carregando todos os dados (caldeirão com a feijoada) para servir cada uma das 100 pessoas. Neste caso, é mais rápido e prático colocar pequenas porções da feijoada em tigelas para servir as 100 pessoas.

O tempo que você desperdiçaria carregando o caldeirão de um lado para o outro pode ser considerado o overhead. Logo,

utilizando as bandejas com pequenas tigelas iriamos gastar menos tempo e, consequentemente, o overhead seria menor. Observe a tabela 2.1 para entender o papel de cada elemento da analogia.

Tabela 2.1: Descrição dos elementos da analogia

Elemento da analogia	Elemento computacional	Descrição
Caldeirão	Memória não volátil	Consegue armazenar grande quantidade de informação e contribui para um alto overhead.
Tigela	Memória volátil	Consegue armazenar pequena quantidade de informação e, por este motivo, contribui para diminuir o overhead.
As 100 pessoas	Processos	São as tarefas que necessitam de processamento.
Servir as pessoas	Processamento	Tarefa realizada pelo processador.
Levar a feijoada até as pessoas	Overhead	Tempo desperdiçado realizando tarefas necessárias antes do processamento.

Por fim, o último componente que um computador deve ter, segundo a arquitetura de von Neumann, é a unidade central de processamento com capacidade de realizar processamento lógico e aritmético. Saber que a CPU realiza processamento lógico e aritmético é extremamente importante para quem está pensando em aprender uma linguagem de programação. Essa pequena informação implica que você terá apenas que aprender como programar expressões lógicas e aritméticas as quais serão processadas pelo computador.

Na verdade, quando você está estudando uma linguagem de programação, o que você está aprendendo são os comandos necessários para poder utilizar e controlar cada um dos componentes definidos na arquitetura de von Neumann.

Se você pretende construir um programa de computador, provavelmente precisará realizar interação com o usuário. Logo, você deverá aprender os comandos para utilizar e controlar os

dispositivos de entrada e saída de dados.

Como todo programa realiza algum tipo de processamento e estes ou são de natureza lógica ou aritmética, então, você, ao estudar a linguagem de programação, aprenderá os comandos necessários para programar tais processamentos.

O resultado de um processamento geralmente é armazenado, ainda que de forma temporária, para posterior utilização dentro do programa. Neste caso, você deve aprender quais os comandos da linguagem de programação são necessários para armazenar e recuperar as informações da memória volátil.

Muitos programas também precisam armazenar informações de forma permanente. Isso significa que você também precisa aprender como a linguagem de programação que você vai estudar gerencia os dados das memórias não voláteis. Normalmente armazenamos dados na memória não volátil a partir da manipulação de arquivos ou de um gerenciador de banco de dados.

Observe meu caro leitor que, se você pretende aprender uma linguagem de programação, é muito importante que você conheça e tenha em mente, ainda que de forma superficial, como os componentes da arquitetura de von Neumann funcionam e se interligam. Esse conhecimento lhe permitirá entender como um computador funciona e, quando estiver programando, terá a habilidade de associar suas linhas de código com os componentes do computador.

2.2 Escrita e leitura na memória

Para lhe demonstrar a importância de ter em mente a arquitetura de von Neumann enquanto estiver programando considere o seguinte comando: C = 10.

Certamente, pelo seu conhecimento matemático, você sabe

que o comando C = 10 está atribuindo o valor 10 à variável C. Pois bem, como mencionei, todo trecho de código está utilizando um ou mais componentes da arquitetura de von Neumann. Neste caso, atribuir um valor a uma variável está utilizando qual componente?

Se você respondeu memória volátil muito bem você acertou. Agora vamos entender qual o fluxo necessário para que este comando seja executado.

A figura 2.3 representa, de forma simplificada, como o processador se comunica com a memória RAM o os dispositivos de entrada e saída. O barramento vermelho representa o barramento de endereço, o barramento preto representa o barramento de dados e o barramento azul o barramento de controle.

Para simplificar irei omitir a utilização do barramento de controle, sua finalidade é preceder a utilização do barramento de endereço e de dados como uma forma de protocolo para preparar os dispositivos. Se você deseja saber mais sobre a utilização dos barramentos recomendo a leitura do livro Computer Organization and Architecture Designing for Performance de William Stallings ou Organização Estruturada de Computadores de Andrew Tanenbaum.

Toda vez que a CPU precisar realizar escrita ou leitura na memória RAM sempre utilizará primeiro o barramento de endereço (vermelho) e posteriormente o barramento de dado (preto). Em uma operação de leitura a CPU enviará, pelo barramento vermelho, o endereço do dado que precisa ser lido. O dado deste endereço será recuperado e enviado para a CPU pelo barramento preto.

No caso da figura 2.4 o processador está enviando o endereço C. A memória recebe este endereço, recupera o dado armazenado e envia este dado para o processador.

Já em uma operação de escrita, a CPU primeiro enviará, pelo barramento vermelho, o endereço aonde será armazenado

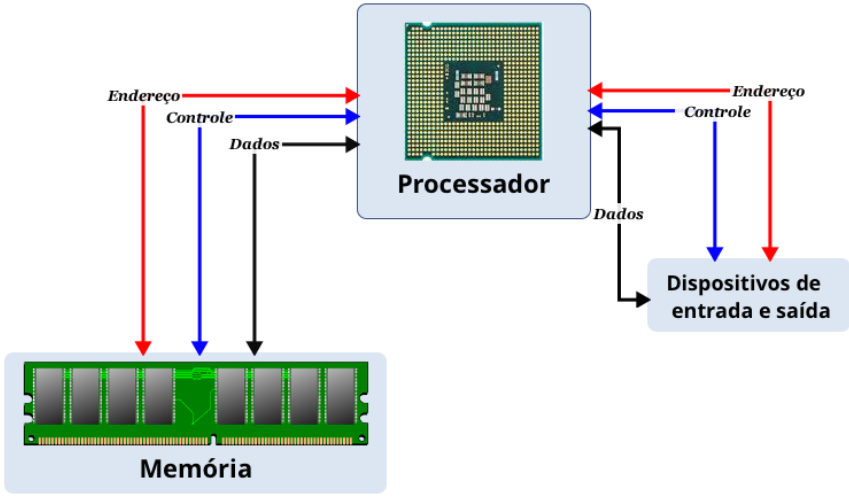


Figura 2.3: barramentos entre memória e processador

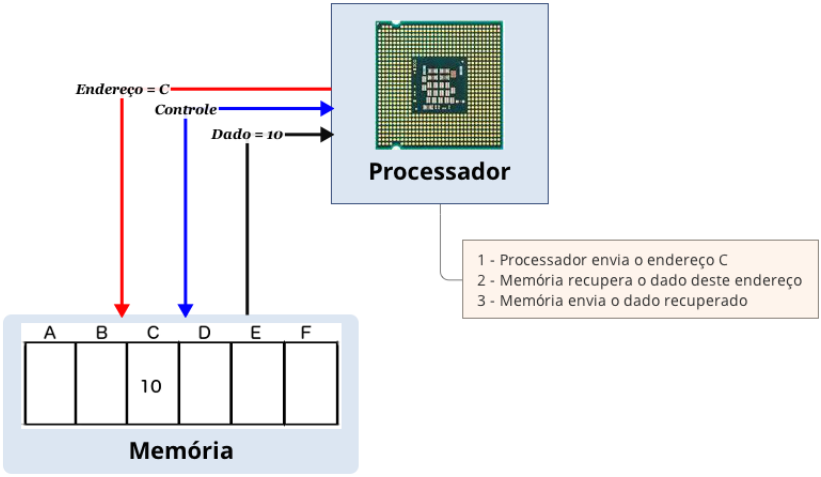


Figura 2.4: leitura de dados

o dado e, posteriormente, enviará, pelo barramento preto, o dado que será armazenado no endereço previamente reservado.

Na figura 2.5 o processador envia o endereço C e a memória reserva este endereço. Em seguida o processador envia o dado, a memória recebe e armazena no endereço previamente reservado.

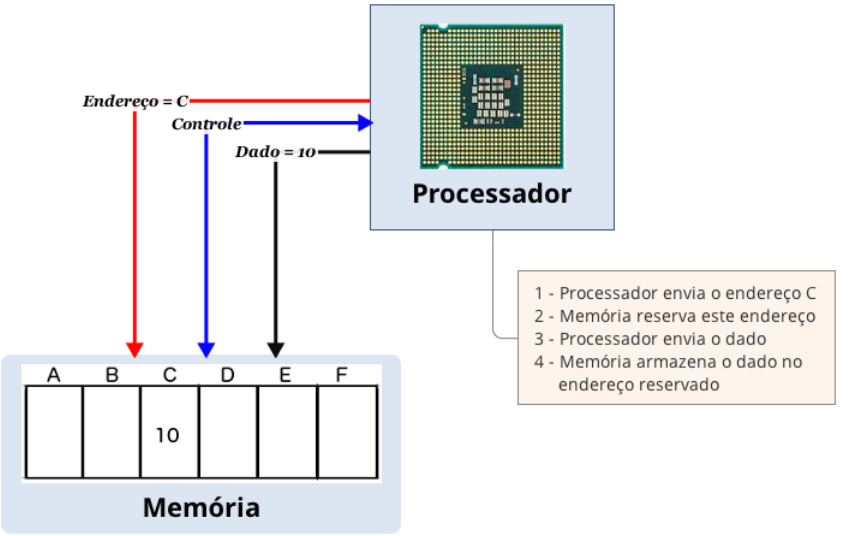


Figura 2.5: escrita de dados

Você pode está estranhando o C ser um endereço da memória. Certamente você já leu em livros de organização e arquitetura de computadores que a memória é endereçada por uma sequência binária e que em geral esta sequência é representada por números hexadecimais. Não se preocupe, tudo o que você estudou está certo. No caso do endereço C o que ocorreu foi que não detalhei que os endereços atribuídos pelo programador são, em tempo de execução, traduzidos para um endereço em sequência binária.

O que na verdade ocorre é que as linguagens de programação de alto nível nos permitem a utilização de endereços lógicos e em tempo de execução cada endereço lógico é asso-

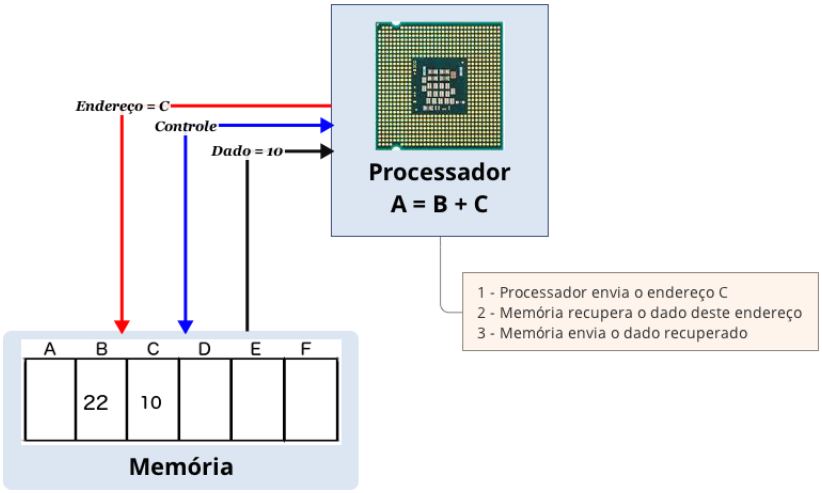


Figura 2.6: leitura do endereço C

ciado a um endereço físico. Explicar como esta tradução de endereços ocorre não é o objetivo deste livro. Se você deseja se aprofundar neste assunto recomendo a leitura do livro Computer Organization and Architecture Designing for Performance de William Stallings ou Organização Estruturada de Computadores de Andrew Tanenbaum.

Considerando o cenário apresentado na figura 2.6 vejamos o fluxo realizado para o processamento da instrução A = B + C.

Nesta operação o processador irá realizar duas leituras na memória e uma escrita, além, é claro, do processamento aritmético.

Você saberia responder quais os endereços serão utilizados para a leitura e escrita? Se para a leitura você respondeu que os endereços são B e C e para a escrita o endereço A, parabéns, você está no caminho certo.

Para realizar a soma de B + C e armazenar o resultado

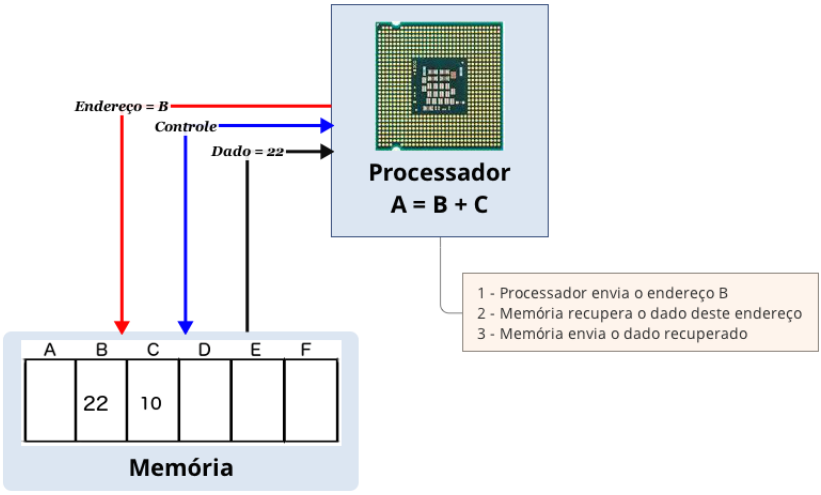


Figura 2.7: leitura do endereço B

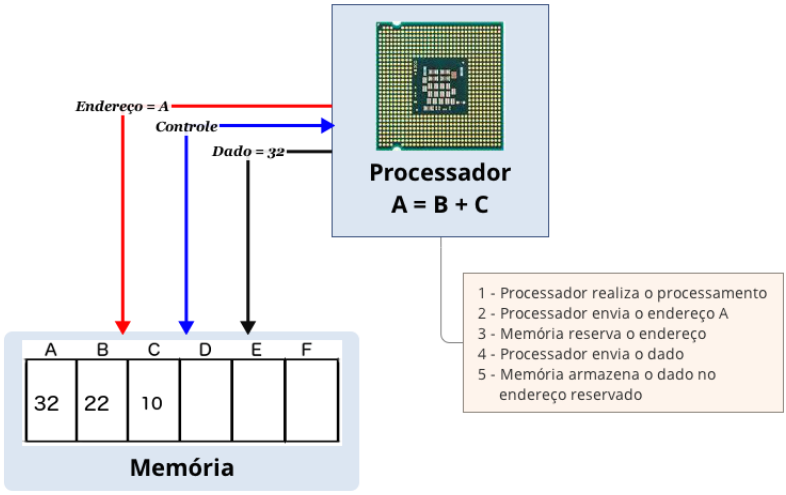


Figura 2.8: processamento e escrita no endereço A

em A, primeiro seria enviado, pelo barramento de endereços (vermelho), o endereço C e, seria retornado, pelo barramento de dados (preto), o valor 10 (ver figura 2.6).

Em seguida, seria enviado, pelo barramento de endereços (vermelho), o endereço B e, seria retornado, pelo barramento de dados (preto), o valor 22 (ver figura 2.7).

Após ler os dois valores é realizado o processamento aritmético e seu resultado precisa ser armazenado na memória volátil. Para isso, primeiramente a CPU envia, pelo barramento de endereços (vermelho), o endereço A e, posteriormente, envia, pelo barramento de dados (preto), o valor que deve ser armazenado no endereço previamente reservado (ver figura 2.8).

A tabela 2.2 descreve o fluxo necessário para realizar o processamento da instrução A = B + C.

Tabela 2.2: Fluxo de processmaento da intrução A = B + C

1- Enviar para a memória o endereço C
2- Recuperar na memória o valor do endereço C
3- Retornar para o processador o valor do endereço C
4- Enviar para a memória o endereço B
5- Recuperar na memória o valor do endereço B
6- Retornar para o processador o valor do endereço B
7- Realizar o processamento aritmético
8- Enviar para a memória o endereço A
9- Reservar o endereço A
10- Enviar para a memória o valor que deve ser armazenado no endereço reservado

A partir da descrição deste fluxo cabe uma observação muito importante para a leitura e interpretação de código fonte que você venha a estudar ou produzir. Você deve ler as linhas do código fonte da direita para a esquerda pois esse é o fluxo

que o computador utiliza para interpretar e processar as instruções. Observe o fluxo da instrução A = B + C. Primeiramente foi recuperado o valor de C seguido do valor de B e posteriormente armazenado o resultado do processamento na variável A. O fluxo parte da direita para a esquerda.

2.3 Entrada e saída de dados

Outro fluxo importante que você deve ter em mente é o de entrada e saída de dados provenientes dos dispositivos de entrada e saída do computador. Toda vez que falarmos de dispositivos de entrada e saída quer dizer que existirá interação com o usuário. Significa que o usuário ou enviará um dado para o computador ou receberá um dado produzido pelo computador.

E figura 2.9 ilustra a leitura de dados do dispositivo de entrada. Ler um dado do dispositivo de entrada significa que o usuário irá enviar um dado, na maioria das vezes pelo teclado, o processador irá recebe-lo e realizar uma escrita na memória. Ou seja, uma leitura do dispositivo de entrada implica em uma operação de escrita na memória.

Na ilustração da figura 2.9 o usuário digitou o número dez que foi lido pelo processador e armazenado no endereço C da memória. A tabela 2.3 detalha o fluxo desta operação.

Tabela 2.3: Leitura no dispositivo de entrada

1- Usuário digita o valor 10 no teclado
2- Processador lê esta informação
3- Processador envia o endereço C para a memória
4- Memória reserva o endereço C
5- Processador envia o valor 10 para a memória
6- Memória recebe o valor 10 e armazena no endereço reservado

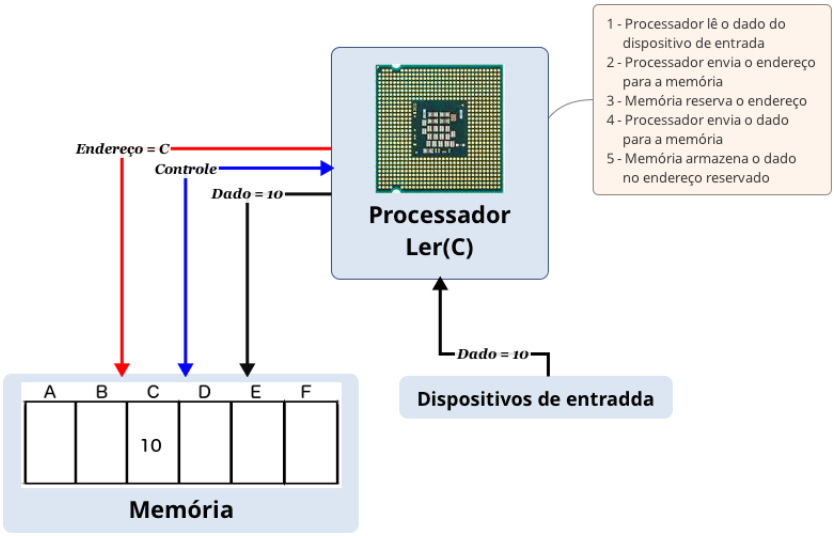


Figura 2.9: entrada de dados

Em relação ao dispositivo de saída, normalmente, é utilizado o monitor. Para o usuário receber uma informação no dispositivo de saída significa que o processador realizou uma escrita de dados neste dispositivo. Entretanto, para escrever esta informação é necessário recuperá-la da memória, ou seja, uma escrita no dispositivo de saída implica em uma leitura na memória.

A figura 2.10 ilustra a escrita de dados no dispositivo de saída e a tabela 2.4 descreve este fluxo.

Escrever programas de computador é uma tarefa que requer dedicação, paciência e conhecimento da arquitetura na qual o programa vai funcionar. Escrever um programa de computador e não saber como suas linhas de código serão interpretadas e processadas é o mesmo que um alfaiate realizar o corte de um tecido sem saber as medidas da pessoa. A chance de o alfaiate produzir uma roupa que fique sob medida será tão pequena

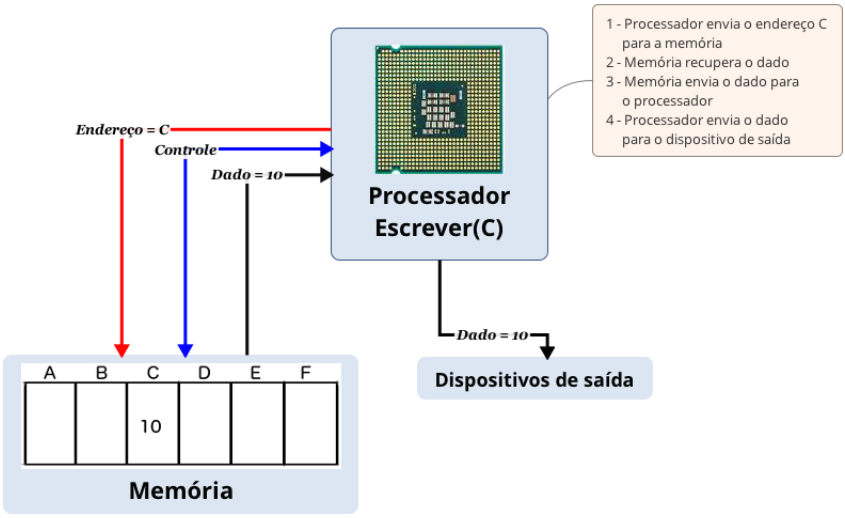


Figura 2.10: entrada de dados

Tabela 2.4: Escrita no dispositivo de saída

1- Processador envia o endereço C para a memória
2- Memória recebe o endereço e recupera o valor armazenado
3- Memória envia o valor recuperado
4- Processador recebe o número 10 e realiza a escrita no dispositivo de saída

quanto a chance de você conseguir escrever um programa de computador sem conhecer os componentes computacionais.

Programar exige dedicação e, neste caso, dedicação é sinônimo de repetição. Se você se interessa por instrumentos musicais sabe que quando estamos aprendendo a tocar um instrumento é fundamental que toquemos inúmeras vezes a mesma música. A cada repetição percebemos novos detalhes e aprendemos algo novo. Na arte de programar não é diferente, é

fundamental que você reescreva pequenos programas e resolva inúmeras vezes o mesmo exercício. A cada repetição você exercitará o seu raciocínio lógico e se familiarizará com a sintaxe da linguagem.

Programar também exige paciência. O início do processo de aprendizado de uma linguagem de programação requer muita paciência uma vez que passamos muitas horas estudando e os programas que conseguimos escrever são extremamente limitados.

Em geral os primeiros programas não possuem interface gráfica e suas funcionalidades são básicas. Normalmente, em um curso introdutório de programação aprendemos a escrever programas para calcular a média; verificar se uma pessoa é maior de idade; converte moedas e temperatura; verificar se um número é primo; e outros programas com grau de chatice igual aos já mencionados.

Mas não se preocupe, neste livro iremos fazer diferente. Os exercícios serão mais animadores, aguarde!

2.4 Resumo

Neste capítulo você aprendeu que é fundamental conhecer o fluxo de dados do computador para poder construir e entender como um programa funciona. Para isso lhe apresentamos a arquitetura de Von Neumann, os conceitos de memória volátil e não volátil.

Você aprendeu, a partir da analogia das tigelas, a importância de se ter a memória volátil na arquitetura computacional. Também lhe apresentamos fluxo de leitura e escrita na memória e o fluxo de escrita e leitura nos dispositivos de entrada e saída.

Aprendeu que uma leitura no dispositivo de entrada im-

plica em uma operação de escrita na memória assim como uma operação de escrita no dispositivo de saída requer em uma operação de leitura na memória.

Tenha em mente que se você dominar os fluxos apresentados neste capítulo você está a um passo de se tornar um ótimo projetista de algoritmo e consequentemente um ótimo programador.

2.5 Exercício

1. Informe se a operação x = 12 requer uma instrução de leitura ou escrita na memória e descreva passo a passo o fluxo realizado.

2. Informe quantas operações de leitura e escrita na memória são necessárias para realizar esta operação x = y*2+z. Descreva passo a passo o fluxo desta operação.

3. Considerando o comando Ler(x), informe qual dispositivo está sendo utilizado. O de entrada ou o de saída? Descreva passo a passo o fluxo desta operação.

4. Considerando o comando Escrever(y), informe qual dispositivo está sendo utilizado. O de entrada ou o de saída? Descreva passo a passo o fluxo desta operação.

5. Descreva passo a passo o fluxo do seguinte código:
   Ler(x)
   Ler(y)
   z = x + y
   Escrever(z)

Capítulo 3

Operadores aritméticos

No capítulo 2 você aprendeu que, segundo à arquitetura de Von Neumann, o processador possui uma unidade lógica e aritmética (ULA). A ULA implica que um computador só pode realizar dois tipos de processamento. Ou o processamento é lógico ou é aritmético.

Essa peculiaridade também reflete na possibilidade da construção de algoritmos. Sabendo que um algoritmo é um projeto computacional que deve ser codificado em uma linguagem de programação. Então, ao escrever um algoritmo, fica claro que devemos utilizar operadores lógicos e aritméticos para construir o processamento da solução.

No próximo capítulo iremos trabalhar com a metodologia PPCT para resolver problemas que necessitem apenas de processamento aritmético.

Sendo assim, o presente capítulo tem o objetivo de lhe apresentar os principais operadores aritméticos que você precisará para resolver a maioria dos exercícios deste livro.

Neste livro, usaremos as mesmas representações utilizadas na matemática. A tabela 3.1 ilustra os operadores que iremos utilizar com mais frequência na construção dos nossos algoritmos.

Assim como na matemática, a tabela 3.1 já está na ordem de precedência em que os operadores devem ser processados.

Antes de utilizarmos esses operadores para a resolução de problemas computacionais vamos fazer uma rápida revisão.

Irei adotar as definições do livro Fundamentos de matemática elementar de Gelson Iezzi, Osvaldo Dolce e Carlos Murakami. Se você não se recordar das definições e exemplos que irá ler, recomende que você leia o volume dois da coleção Fundamentos de matemática elementar.

Tabela 3.1: Operadores aritméticos

Operador	Exemplo	Descrição
( )		Parênteses
**	2**3=8	Potenciação
$\sqrt{}$	$\sqrt{36} = 6$	Radiciação
log_n	$log_2 8 = 3$	Logaritmo
*	$4 * 2 = 8$	Multiplicção
/	$5/2 = 2.5$	Divisão
div	$5 div 2 = 2$	Divisão inteira
mod	$5 mod 2 = 1$	Resto da divisão inteira
+	$5 + 2 = 7$	Adição
−	$5 - 2 = 3$	Subtração
||	$\|-5\| = 5$	Valor absoluto

3.1 Potenciação

Seja a um número real e n um número natural, a potência de base a e expoente n é o número a^n tal que: $\begin{cases} a^0 = 1, \\ a^n = a^{n-1} * a, \quad \forall n \geq 1 \end{cases}$

A potenciação pode ser definida como a multiplicação de um elemento por ele mesmo n vezes, onde n é a potência. Por exemplo, considerando 2**3 temos que:
$2 * *3 = 2^3 = 2 * 2 * 2 = 8$

Já recordou o que é a potenciação? Se ainda não, vejamos mais alguns exemplos:

- $2 * *6 = 2^6 = 2 * 2 * 2 * 2 * 2 * 2 = 64$
- $4 * *2 = 4^2 = 4 * 4 = 16$
- $3 * *3 = 3^3 = 3 * 3 * 3 = 27$
- $10 * *4 = 10^4 = 10 * 10 * 10 * 10 = 10000$

Você também deve recordar a potenciação para expoentes inteiros negativos. Vejamos!

Seja a um número real não nulo e n um número natural, define-se a potência a^{-n} pela relação $a^{-n} = \frac{1}{a^n}$.

Para recodar:

- $2^{-1} = \frac{1}{2^1} = \frac{1}{2}$
- $2^{-3} = \frac{1}{2^3} = \frac{1}{8}$
- $-2^{-3} = \frac{1}{-2^3} = \frac{1}{-8} = -\frac{1}{8}$
- $(\frac{-2}{3})^{-2} = \frac{1}{(\frac{-2}{3})^2} = \frac{1}{\frac{4}{9}} = \frac{9}{4}$

Propriedades

Eis algumas propriedades úteis para possíveis simplificações na construção dos algoritmos.

Se $a \in \mathbb{R}$, $b \in \mathbb{R}$, $m \in \mathbb{N}$ e $n \in \mathbb{N}$, então valem as seguintes propriedades:

- P1: $a^m * a^n = a^{m+n}$ se existir uma multiplicação de potências de mesma base, então pode-se manter a base e somar o expoentes.
- P2: $\frac{a^m}{a^n} = a^{m-n}, a \neq 0$ se existir a divisão de potências de mesma base tal que não seja nula e o expoente do dividendo seja maior ou igual ao do divisor, então pode-se manter a base e subtrair os expoentes.
- P3: $(a * b)^n = a^n * b^n$
- P4: $(\frac{a}{b})^n = \frac{a^n}{b^n}, b \neq 0$
- P5: $(a^m)^n = a^{m*n}$

A demonstração dessas propriedades pode ser estudada no volume 2 do livro Fundamentos de matemática elementar de Gelson Iezzi, Osvaldo Dolce e Carlos Murakami.

3.2 Radiciação

A radiciação pode ser definida como a operação inversa da potenciação. Sendo x um número real e n um número natural, temos que: $\sqrt[n]{x} = y$ implca em $x = y^n$.

Vejamos alguns exemplos:

- $\sqrt[2]{36} = 6$ pois $36 = 6^2$
- $\sqrt[3]{8} = 2$ pois $8 = 2^3$
- $\sqrt[4]{81} = 3$ pois $81 = 3^4$
- $\sqrt[3]{-27} = -3$ pois $-27 = -3^3$

Propriedades

Eis algumas propriedades úteis para possíveis simplificações na construção dos algoritmos.

Se $a \in \mathbb{R}_+$, $b \in \mathbb{R}_+$, $m \in \mathbb{Z}$, $n \in \mathbb{N}^*$ e $p \in \mathbb{N}^*$, temos:

- P1: $\sqrt[n]{a^m} = \sqrt[n*p]{a^{m*p}}$
- p2: $\sqrt[n]{a * b} = \sqrt[n]{a} * \sqrt[n]{b}$
- P3: $\sqrt[n]{\frac{a}{b}} = \frac{\sqrt[n]{a}}{\sqrt[n]{b}}, b \neq 0$
- P4: $(\sqrt[n]{a})^m = \sqrt[n]{a^m}$
- P5: $\sqrt[p]{\sqrt[n]{a}} = \sqrt[p*n]{a}$

A demonstração dessas propriedades pode ser estudada no volume 2 do livro Fundamentos de matemática elementar de Gelson Iezzi, Osvaldo Dolce e Carlos Murakami.

A partir da radiciação também podemos definir a potenciação com expoente racional.

Sendo $a \in \mathbb{R}_+^*$ e $\frac{p}{q} \in \mathbb{Q}$ ($p \in \mathbb{Z}$ e $q \in \mathbb{N}^*$), temos que $a^{\frac{p}{q}} = \sqrt[q]{a^p}$.

Vejamos alguns exemplos:

- $3^{\frac{1}{2}} = \sqrt[2]{3}$
- $7^{\frac{-2}{3}} = \sqrt[3]{7^{-2}} = \sqrt[3]{\frac{1}{7^2}} = \sqrt[3]{\frac{1}{49}}$
- $(\frac{2}{3})^{\frac{-1}{3}} = \sqrt[3]{(\frac{2}{3})^{-1}} = \sqrt[3]{\frac{3}{2}}$

3.3 Logaritmo

Você ainda lembra do logaritmo? Preste atenção que logaritmo é diferente de algoritmo!

Sendo a e b número reais positivos, com $a \neq 1$, chama-se logaritmo de b na base a o expoente que deve se dar à base a de modo que a potência obtida seja igual a b.

Em símbolos: Se $a, b \in \mathbb{R}, 0 < a \neq 1$ e $b > 0$, então $log_a b = x \Leftrightarrow b = a^x$.

Vejamos alguns exemplos:

- $log_2 8 = 3$ pois $8 = 2^3$
- $log_3 \frac{1}{9} = -2$ pois $\frac{1}{9} = 3^{-2}$
- $log_7 1 = 0$ pois $1 = 7^0$
- $log_4 8 = \frac{3}{2}$ pois $8 = 4^{\frac{3}{2}} = (2^2)^{\frac{3}{2}} = 2^3$

Propriedades

Eis algumas propriedades úteis para possíveis simplificações na construção dos algoritmos.

- Logaritmo do produto: Sendo $0 < a \neq 1$, $b > 0$ e $c > 0$, então $log_a(b * c) = log_a b + log_a c$
- Logaritmo do quociente: Sendo $0 < a \neq 1$, $b > 0$ e $c > 0$, então $log_a(\frac{b}{c}) = log_a b - log_a c$
- Logaritmo da potência: Sendo $0 < a \neq 1$, $b > 0$ e $x \in \mathbb{R}$, então $log_a b^x = x * log_a b$

3.4 Resto da divisão

Dentre os operadores apresentados na tabela 3.1, outro que talvez você tenha que revisar seja o *mod*. Este operador tem como objetivo encontrar o resto da divisão inteira.

Considerando a figura xx temos que:

- Divisão: $58/5 = 11.6$
- Divisão inteira: $58\ div\ 5 = 11$
- Resto da divisão inteira: $58\ mod\ 5 = 3$

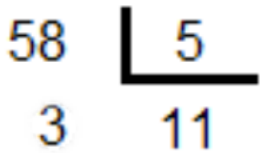


Figura 3.1: dividendo, divisor, quociente e resto.

O operador *mod* é muito utilizado na construção de algoritmos. Uma de suas principais características é o movimento

circular. Um exemplo clássico do movimento circular do operador *mod* é o relógio de parede.

Figura 3.2: relógio de parede

Quando falamos que são 16:00, que horas são no relógio de parede? Claro que você sabe que são quatro horas, mas você sabe por quê?

Veja só:

- $13\ mod\ 12 = 1$
- $14\ mod\ 12 = 2$
- $15\ mod\ 12 = 3$
- $16\ mod\ 12 = 4$
- $17\ mod\ 12 = 5$
- $18\ mod\ 12 = 6$
- $19\ mod\ 12 = 7$
- $20\ mod\ 12 = 8$
- $21\ mod\ 12 = 9$
- $22\ mod\ 12 = 10$

- $23\ mod\ 12 = 11$
- $24\ mod\ 12 = 0$
- .....

No caso do número 12, a operação *mod* 12 só poderá resultar em um dos seguintes valores: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$. Devido a esta característica é que o operador mod possui movimento circular.

Generalizando, sendo x um número inteiro positivo, o operação *mod* x só poderá resultar em um dos seguintes valores: $\{0, ..., x - 1\}$.

Observe o movimento circular para $x = 3$. Neste caso os possíveis valores da operação *mod* 3 são $\{0, 1, 2\}$.

- $0\ mod\ 3 = 0$
- $1\ mod\ 3 = 1$
- $2\ mod\ 3 = 2$
- $3\ mod\ 3 = 0$
- $4\ mod\ 3 = 1$
- $5\ mod\ 3 = 2$
- $6\ mod\ 3 = 0$
- $7\ mod\ 3 = 1$
- $8\ mod\ 3 = 2$
- .....

Outro caso clássico da utilização do operador *mod* é para verificar se um determinado número é par ou ímpar. Sabendo que um número par é aquele que quando dividido por dois gera resto zero, então, temos que:

- $0\ mod\ 2 = 0$
- $1\ mod\ 2 = 1$
- $2\ mod\ 2 = 0$
- $3\ mod\ 2 = 1$
- $4\ mod\ 2 = 0$
- $5\ mod\ 2 = 1$
- $6\ mod\ 2 = 0$
- .....

Observe que, pela definição, a operação *mod* 2 só pode resultar nos valores $\{0, 1\}$.

3.5 Exercício

Estes exercícios têm como objetivo realizar um aquecimento mental. Todas as questões foram retiradas do volume 2 do livro Fundamentos de matemática elementar de Gelson Iezzi, Osvaldo Dolce e Carlos Murakami.

1. Calcular:
   a)$(-3)^2$ b)-3^2 c)-2^3 d)$-(-2)^3$
   e)$(-3)^3$ f)$(-2)^1$ g)3^4 h)1^7
   i)$(\frac{2}{3})^3$ j)$(\frac{1}{3})^4$ k)0^7 l)$(\frac{2}{3})^0$

2. Calcular:
   a)3^{-1} b)$(-2)^{-1}$ c)$-(-3)^{-1}$ d)2^{-2}

   e)$(-3)^{-2}$ f)-5^{-2} g)$(\frac{1}{3})^{-2}$ h)$(\frac{2}{3})^{-1}$

   i)$(-\frac{2}{3})^{-3}$ j)$-(\frac{2}{5})^{-2}$ k)$\frac{1}{2^{-3}}$ l)$\frac{1}{(-3)^{-3}}$

3. Simplificar os radicais:
   a)$\sqrt[3]{64}$ b)$\sqrt{576}$ c)$\sqrt{12}$ d)$\sqrt[3]{2^7}$

   e)$\sqrt[3]{72}$ f)$\sqrt{144}$ g)$\sqrt{18}$ h)$\sqrt[3]{729}$

4. Calcular:
   a)$\sqrt{3} * \sqrt{12}$ b)$\frac{\sqrt[3]{24}}{\sqrt[3]{2}}$

   c)$\frac{\sqrt{\frac{3}{2}}}{\sqrt{\frac{1}{2}}}$ d)$(\sqrt{12} - 2 * \sqrt{27} + 3 * \sqrt{75}) * \sqrt{3}$

5. Expressar na forma de potência de expoente racional os seguintes radicais:
   a)$\sqrt{5}$ b)$\sqrt[3]{4}$ c)$\sqrt[4]{27}$ d)$\sqrt{\sqrt{2}}$

   e)$\sqrt[4]{\sqrt[3]{5}}$ f)$(\sqrt[3]{2^2})^2$ g)$\frac{1}{\sqrt{2}}$ h)$\frac{1}{\sqrt[3]{9}}$

6. Calcular:
   a)$log_2 \frac{1}{8}$ b)$log_8 4$ c)$log_{\frac{1}{4}} 32$ d)$log_4 16$

   e)$log_3 \frac{1}{9}$ f)$log_{81} 3$ g)$log \frac{1}{2} 8$ h)$log_2 \sqrt{2}$

Capítulo 4

Processamento aritmético

Neste capítulo você irá aprender a pensar, planejar e escrever algoritmos para resolver problemas que necessitem apenas de processamento aritmético.

Certamente, algoritmos que utilizam somente processamento aritmético não conseguem resolver todo e qualquer tipo de problema. Entretanto, você vai notar que existem problemas bem interessantes que podem ser resolvidos apenas aritmeticamente.

Iremos lhe apresentar problemas de vários níveis de dificuldade durante o capítulo e na lista de exercício. Você vai perceber que a parte mais difícil da resolução do problema é a parte do pensar. Depois de conseguir elaborar o raciocínio lógico para a resolução do problema, a escrita do algoritmo torna-se trivial.

Não se esqueça! Você sempre deve ter em mente que todo problema deve ser dividido em três partes: entrada, processamento e saída. Portanto, quando você for ler o enunciado das questões sempre procure identificar quais são as entradas necessárias, o que deve ser processado e qual ou quais as saídas que devem ser exibidas.

4.1 Representação da entrada de dados

A entrada de dados representa as informações que o usuário do programa precisa fornecer a partir do dispositivo de entrada para que o processamento possa ser realizado.

Imagine que você deve escrever um programa para somar dois números fornecidos pelo usuário. Neste caso, quantas informações o usuário precisa fornecer?

Caso você não tenha chegado a resposta lhe recomendo sempre realizar o seguinte exercício: ao ler o enunciado da

questão sempre tente desenhar a tela com campos de texto para que o usuário forneça as informações. A partir do desenho acredito que seja fácil identificar quantas entradas o programa precisa para realizar o processamento.

No caso do enunciado: Escrever um programa para somar dois números fornecidos pelo usuário. Uma possível tela seria como a ilustrada na figura **??**. Olhando para o desenho fica

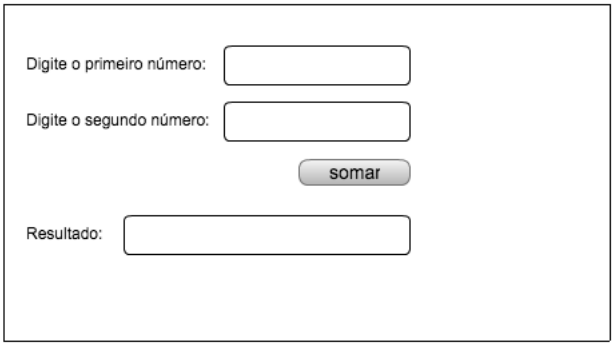


Figura 4.1: Pojetando a tela para descobrir as entradas.

fácil perceber que o usuário deve digitar dois números e que o programa que você deve construir terá que ler essas duas informações.

Em algoritmo iremos representar a leitura das informações do dispositivo de entrada a partir do comando "Ler". Como mencionado no capítulo 2, a leitura no dispositivo de entrada implica na escrita de memória. Por este motivo, para cada informação lida, é necessário informar em qual endereço esta informação será armazenada.

No caso do nosso enunciado a representação para a leitura de dois números seria, por exemplo, Ler(x,y). Com o comando Ler(x,y) estamos descrevendo que o programa deve ler duas informações do dispositivo de entrada e que estas informações serão armazenadas na memória nos endereços x e y

respectivamente. A sintaxe da instrução ler será Ler(end) ou Ler(end1,...,end2).

Quando for necessário ler apenas uma informação do dispositivo de entrada então basta informar, entre parênteses, o endereço da memória em que esta informação será armazenada.

Se houver a necessidade de ler mais de uma informação então você pode, em um único comando ler, informar os endereços da memória em que as informações serão armazenadas. Os endereços devem ser separados por vírgula. Os três pontinhos da sintaxe informam que você pode informar quantos endereços forem necessários.

É importante ressaltar que os endereços sempre devem ser diferentes. Caso você utilize um endereço mais de uma vez, então, o valor previamente armazenado neste endereço será substituído pelo novo.

Você também pode ler mais de uma informação de entrada utilizando varias vezes o comando "Ler". Por exemplo, ao invés de ler dois números como o comando Ler(x,y), poderíamos escrever:

- Ler(x)
- Ler(y)

4.2 Representação da saída de dados

A representação de saída de dados tem sintaxe similar ao da entrada de dados. Iremos utilizar o comando "Escrever" para informar que estaremos exibindo uma informação no dispositivo de saída de dados do computador.

A sintaxe da instrução escrever será Escrever(end) ou Escrever(texto) ou, de formar similar ao comando "Ler", podemos

utilizar as vírgulas para escrever mais de uma informação no mesmo comando.

No caso do enunciado: escreve um programa para realizar a soma de dois números fornecidos pelo usuário. O que precisamos mostrar no dispositivo de saída é o resultado desta soma.

Vejamos uma versão do algoritmo que resolve este problema:

```
INÍCIO
    LER(X,Y)
    Z = X + Y
    ESCREVER(Z)
FIM
```

Figura 4.2: Algoritmo para somar dois números.

Neste caso o que está sendo escrito no dispositivo de saída é a informação que está armazenada no endereço Z. Como mencionado no capítulo 2, a escrita no dispositivo de saída do computador implica na leitura de memória. Logo, uma possível leitura do comando ESCREVER(Z) seria: recupere da memória informação armazenada no endereço Z e envie para o dispositivo de saída.

Neste momento cabe uma observação importante. Toda vez que você ver uma letra ou uma palavra dentro dos parênteses do comando "Escrever", significa que esta letra ou palavra é um endereço da memória. Logo, o que será exibido no dispositivo de saída não é esta letra ou palavras, mas sim a informação que está armazenada neste endereço da memória.

Por outro lado, se você ver, dentro dos parênteses do comando "Escrever", uma letra, palavra ou frase que esteja entre aspas, significa que esta informação será exibida no dispositivo de saída. Ou seja, se tiver entre aspas não é endereço de

memória!

Para exemplificar, observe a segunda versão do algoritmo que resolve o problema da soma de dois números:

```
INÍCIO
    LER(X,Y)
    Z = X + Y
    ESCREVER("Soma = ", Z)
FIM
```

Figura 4.3: Algoritmo para somar dois números.

Neste exemplo o comando escrever está exibindo no dispositivo de saída o texto "Soma ="e o valor armazenado no endereço Z da memória.

Vamos praticar a leitura deste algoritmo:

- **LER(X,Y):** Leia dois números e armazene nos endereços de memória X e Y.
- **Z = X+Y:** Recupere os valores dos endereços X e Y, realize a soma e armazene o resultado no endereço Z.
- **ESCREVER("Soma = ", Z):** Escreva o texto "Soma = Z" e escreva o valor recuperado do endereço Z da memória.

Veja que a partir deste simples algoritmo podemos perceber claramente que a solução foi dividida nas etapas de entrada, processamento e saída.

Outra observação importante é que todo algoritmo deve iniciar com a palavra INÍCIO e terminar com a palavra FIM. Esses dois comandos são responsáveis pela definição do escopo do algoritmo, ou seja, onde ele começa e onde ele termina.

4.3 Metodologia PPCT

Para resolver um problema computacional iremos utilizar o que chamo de metodologia PPCT (pensar, planejar, codificar e testar).

Pensar

A primeira etapa, que diz respeito ao pensar, tem como objetivo ler o enunciado do problema, interpretar e conseguir raciocinar para chegar a fórmula matemática que resolva o problema.

Em resumo, dado o enunciado de um problema, você sempre tem que identificar quais os dados que serão fornecidos pelo dispositivo de entrada e pensar em como utilizá-los para realizar o processamento adequado para conseguir produzir o resultado que deve ser escrito no dispositivo de saída.

Durante a resolução das questões você vai perceber que a etapa do pensar é a mais complexa de todas.

Planejar

A etapa do planejamento diz respeito a escrita do algoritmo. Ao escrever o algoritmo você deve ter em mente que o mesmo deve ser estruturado em entrada, processamento e saída.

Como você leu anteriormente, a entrada e saída são realizadas respectivamente pelos comandos “Ler” e “Escrever”. Já o processamento será construindo a partir do raciocínio desenvolvido na etapa do pensar.

Você vai perceber que se a etapa do pensar for realizada com sucesso, a etapa de planejamento torna-se trivial. Ou seja, escrever um algoritmo não é difícil, difícil é desenvolver o raciocínio lógico para encontrar a solução do problema.

Codificar

Neste livro iremos lhe ensinar como realizar a codificação dos algoritmos utilizando a linguagem de programação Python.

Os conceitos da linguagem irão ser apresentados conforme a necessidade. Neste capítulo, como será abordado somente problemas de natureza aritmética, será necessário apenas o domínio dos comandos de entrada, saída e da representação dos operadores aritméticos da linguagem.

Você deve ler o capítulo 8 para aprender a preparar o ambiente de desenvolvimento e o capítulo 9 para tirar suas dúvidas em relação aos principais comandos da linguagem.

O capítulo 9 tem o objetivo de lhe apresentar a regra de utilização dos comandos independentemente de contexto. Você deve aprender a etapa de codificação no decorrer dos capítulos de uma forma contextualizada e quando necessário realize a consulta do capítulo 9.

Vale ressaltar que o principal objetivo deste livro é lhe ensinar a desenvolver o raciocínio lógico e, por este motivo, não será abordada todas as especificidades da linguagem Python.

No futuro, quando você terminar de ler este livro e quiser aprimorar suas habilidades nesta linguagem, recomendo a leitura do livro Python para desenvolvedores de Luiz Eduardo Borges.

Testar

O teste consiste em uma das principais etapas da metodologia PPCT. É nesta etapa que você tem a oportunidade de encontrar possíveis erros de lógica existentes no seu programa.

Para testar o seu programa você precisa elaborar um conjunto de entrada e verificar se a saída produzida está conforme a esperada.

No caso das nossas questões você pode utilizar os exemplos de entrada e verificar se os resultados produzidos são os esperados.

Além dos exemplos você deve elaborar outros casos de teste. Elaborar casos de teste é uma prática importante para entender o fluxo do seu código e aperfeiçoar a sua leitura do código fonte.

Outra prática que deve ser colocada em prática durante a etapa de teste é a depuração de código.

Depurar um código consiste em verificar, em tempo de execução, o seu comportamento.

A depuração de código lhe permite verificar para cada caso de teste os valores calculados e armazenados na memória.

Você de ler o capítulo 8 para aprender a utilizar a depuração de código do ambiente da linguagem Python.

Calme! Não vá ler o capítulo8 agora, quando for o momento certo irei lhe informar.

4.4 Questão 1: Busca na internet

Esta questão foi adaptada da Olimpíada Brasileira de Informática de 2012.

João fez uma pesquisa em seu site de busca predileto, e encontrou a resposta que estava procurando no terceiro link listado. Além disso, ele viu, pelo site, que t pessoas já haviam clicado neste link antes. João havia lido anteriormente, também na Internet, que o número de pessoas que clicam no segundo link listado é o dobro de número de pessoas que clicam no terceiro link listado. Nessa leitura, ele também descobriu que o número de pessoas que clicam no segundo link é a metade do número de pessoas que clicam no primeiro link. João está intrigado para saber quantas pessoas clicaram no primeiro

link da busca, e, como você é amigo dele, quer sua ajuda nesta tarefa.

Entrada

Cada caso de teste possui apenas um número, t, que representa o número de pessoas que clicaram no terceiro link da busca.

Saída

Para cada caso de teste imprima apenas uma linha, contendo apenas um inteiro, indicando quantas pessoas clicaram no primeiro link, nessa busca.

Exemplo

Entrada	Saída
2	8
163	652
67	268
213	852

Pensando

Ao interpretar o enunciado do problema você pode extrair as seguintes informações:

- Que t pessoas clicaram no terceiro link.
- Que o dobro de pessoas clicou no segundo link.
- E que o número de pessoas que clicaram no segundo link é a metade do número de pessoas que clicaram no primeiro link.

A partir dessas informações você pode montar um esquema conforme a figura **??** e chegar as seguintes conclusões:

- O dobro de t é $2t$. Logo, $2t$ representa o número de pessoas que clicaram no segundo link.
- Se $2t$ é a metade do número de pessoas que clicaram no primeiro link, então, $4t$ é o número de pessoas que clicaram no primeiro link.

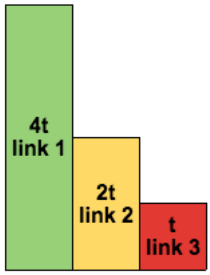


Figura 4.4: Proporção de cliques em cada link

Realizada esta análise é fácil perceber que o número de pessoas que clicaram no primeiro link é quatro vezes maios que o número de pessoas que clicaram no terceiro link.

Pronto! O processamento do programa já foi determinado, agora é planejar e escrever o algoritmo.

Planejando

Antes de escrever o algoritmo não esqueça que ele deve ser estruturado em três partes: entrada, processamento e saída.

A entrada, como dito no enunciado, é de apenas um número que informa o número de pessoas que clicaram no terceiro link.

O processamento, como analisado na etapa do pensar, é a simples multiplicação de t por 4.

A saída consiste em exibir o resultado do processamento a ser realizado.

Diante dessas observações veja o algoritmo resultante:

```
INÍCIO
    LER(t)
    link1 = 4*t
    ESCREVA(link1)
FIM
```

Como dito anteriormente, depois que você consegue pensar a solução do problema e sabe que deve estruturar o algoritmo nas etapas de entrada, processamento e saída a escrita do algoritmo torna-se trivial.

Vamos praticar a leitura deste algoritmo:

- **LER(t):** Leia o número de pessoas que clicaram no terceiro link e armazene no endereço "t".
- **link1 = 4*t:** Recupere o valor do endereço "t" e calcule o número de pessoas que clicaram no primeiro link. Armazene o resultado no endereço link1.
- **ESCREVER(link1):** Recupere da memória o valor armazenado no endereço link1 e escreva no dispositivo de saída.

Codificando

Para codificar este algoritmo em uma linguagem de programação você precisa saber quais são os comandos de entrada e

saída. No caso da linguagem Python os comandos são respectivamente input() e print(). Veja como utilizá-lo.

```
t = input()
link1 = 4*t
print(link1)
```

Como você pode perceber a única diferença entre o código e o algoritmo está no comando de entrada. Enquanto no algoritmo você deve informar o endereço que irá armazenar o valor lido do dispositivo de entrada dentro dos parênteses do comando "Ler", na linguagem Python este endereço fica fora e separado pelo operador de atribuição igual.

Você deve ler o capítulo XX para aprender como preparar o ambiente da linguagem de programação Python.

Testando

Você pode utilizar os casos de teste apresentados no enunciado da questão para testar o seu programa. Também é recomendado que você produza outros casos de teste e verifique se o resultado será conforme o esperado.

4.5 Questão 2: Pedágio

Esta questão foi adaptada da Olimpíada Brasileira de Informática de 2010.

A invenção do carro tornou muito mais rápido e mais barato realizar viagens de longa distância. Realizar uma viagem rodoviária tem dois tipos de custos: cada quilômetro percorrido na rodovia tem um custo associado (não só devido ao

consumo de combustível, mas também devido ao desgaste das peças do carro, pneus, etc.), mas também é necessário passar por vários pedágios localizados ao longo da rodovia.

Os pedágios são igualmente espaçados ao logo da rodovia; o começo da estrada não possui um pedágio, mas o seu final pode estar logo após um pedágio (por exemplo, se a distância entre dois pedágios consecutivos for de 37 km e a estrada tiver 111 km, o motorista deve pagar um pedágio aos 37 km, aos 74 km e aos 111 km, logo antes de terminar a sua viagem).

Dadas as características da rodovia e os custos com gasolina e com pedágios, você foi contratado para construir um programa que calcule o custo total da viagem.

Entrada

A entrada consiste de duas linhas. A primeira linha da entrada contém dois inteiros L e D, indicando o comprimento da estrada e a distância entre pedágios, respectivamente. A segunda linha contém dois inteiros K e P, indicando o custo por quilômetro percorrido e o valor de cada pedágio. O primeiro pedágio está localizado no quilômetro D da estrada (ou seja, a distância do início da estrada para o primeiro pedágio é D quilômetros).

Saída

Seu programa deve imprimir uma única linha contendo um único inteiro, indicando o custo total da viagem.

Exemplo

Entrada	Saída
111 37 1 10	141
100 30 3 14	342
9999 10000 9000 8000	89991000
5734 1433 9678 8527	55527760

Pensando

Para chegar a solução deste problema, o custo total da viagem, você tem que calcular dois custos. O custo por quilômetro percorridos e o custo com os pedágios.

Calcular o custo por quilômetro percorridos é relativamente simples. Dentre as informações recebidas pelo dispositivo de entrada o programa deve armazenar respectivamente nos endereços K e L o custo por para percorrer um quilômetro e o comprimento da estrada.

A partir destas informações o custo por quilômetro percorrido é calculado pelo produto de K e L. Por exemplo, considerando que $K = 5$ e $L = 100$, então o custo por quilômetro seria igual a $L * K = 500$.

A segunda parte da expressão que informa o custo total da viagem diz respeito ao custo com os pedágios. Para calcular o custo com os pedágios você irá precisar de três informações recebidas pelo dispositivo de entrada e que serão armazenadas respectivamente nos endereços L, D e P. Em L será armazenada o comprimento da estrada, em D a distância entre os pedágios e em P o valor que deve ser pago em cada pedágio.

Sendo assim, o custo com os pedágios será o produto entre a quantidade de pedágios e o valor de cada pedágio. Porém, observe que você ainda não possui a quantidade de pedágios existentes na estrada. Este valor precisa ser calculado a partir das informações armazenadas nos endereços L e D.

A figura **??** ilustra um esquema da distribuição dos pedágios ao longo da estrada.

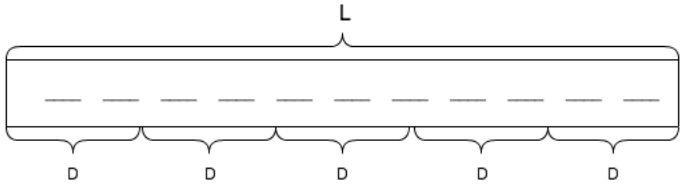


Figura 4.5: Pedágios ao longo da estrada

Observe que ao longo comprimento da estrada (L), a cada (D) quilômetros existirá um pedágio. Logo, para descobrir o número de pedágios basta calcular a razão de L por D.

Todavia observe o seguinte. Considerando $L = 90$ e $D = 7$ a razão de L por D seria 12.857. Porém, é claro que não pode existir essa quantidade de pedágios.

Como o número de pedágios é uma contagem inteira, ou seja, ou existem 12 ou existem 13 pedágios. Então, a razão de L por D deve retornar somente a parte inteira do quociente.

Sendo assim, para realizar esta operação você deve utilizar a operação *div* conforme mencionado no capítulo 3.

Realizada esta análise temos que:

- Custo por quilômetro: $Cq = L * k$.
- Custo com pedágios: $Cp = (LdivD) * P$.
- Custo total: $Ct = Cq + Cp$.

Planejando

Antes de escrever o algoritmo não esqueça que ele deve ser estruturado em três partes: entrada, processamento e saída.

A entrada, como dito no enunciado, se dá em duas partes. Primeiro devem ser lidos dois números e armazenados respectivamente nos endereços L e D. O endereço L irá armazenar o comprimento da estrada e D a distância entre os pedágios.

Em seguida devem ser lidos mais dois números e armazenados respectivamente nos endereços K eP. O endereço K irá armazenar o custo para percorrer um quilômetro e P o custo de um pedágio.

O processamento, como analisado na etapa do pensar, será dividido em três etapas. Cálculo do custo por quilômetro, cálculo do custo com os pedágios e cálculo do custo total.

A saída consiste em exibir o resultado do processamento a ser realizado.

Diante dessas observações veja o algoritmo resultante:

```
INÍCIO
    LER(L,D)
    LER(K,P)
    Cq = L*K
    Cp = (L div D)*P
    Ct = Cq+Cp
    ESCREVER(Ct)
FIM
```

Vamos praticar a leitura deste algoritmo:

- **LER(L,D):** Leia do dispositivo de entrada o comprimento da estrada e a distância entre os pedágios e armazene respectivamente nos endereços de memória L e D.

- **LER(K,P):** Leia do dispositivo de entrada o custo para percorrer um quilômetro e o custo de um pedágio e armazene respectivamente nos endereços de memória K e P.

- **Cq = L*K:** Recupere os valores dos endereços L e K, realize a multiplicação e armazene o resultado no endereço de memória Cq.

- **Cp = (L div D) * P:** Recupere os valores dos endereços L, D e P, realize a divisão inteira de L por D e em seguida multiplique por P. Armazene o resultado deste cálculo no endereço Cp.

- **Ct = Cq + Cp:** Recupere os valores dos endereços Cq e Cp, realize a adição entre esses valores e armazene o resultado no endereço de memória Ct.

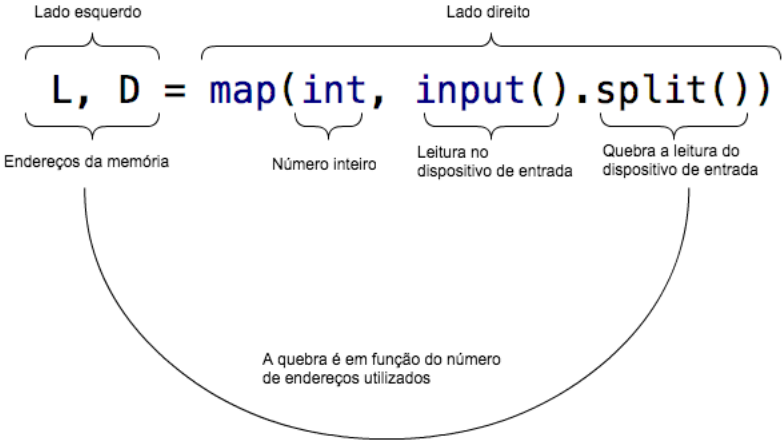
- **ESCREVER(Ct):** Recupere da memória o valor armazenado no endereço Ct e escreva no dispositivo de saída.

Codificando

Para codificar este algoritmo na linguagem Python você precisará aprender como ler mais de uma entrada por linha. Ou seja, como traduzir para o Python o comando $Ler(L, D)$ por exemplo.

Em Python a tradução de $Ler(L, D)$ é conforme a apresentada na figura **??**.

No lado esquerdo do operador de atribuição você deve informar, separado por vírgula, os endereços de memória que irão armazenar os dados lidos do dispositivo de entrada. No caso deste exemplo, como foi informado somente dois endere-



ços (L e D), significa que serão lidos dois dados do dispositivo de entrada e serão armazenados nos respectivos endereços.

No lado direito do operador de atribuição você deve utilizar a função $map(arg1, arg2)$. Esta função recebe dois argumentos ($arg1$ e $arg2$). No primeiro argumento ($arg1$) você deve informar o tipo dos dados que serão lidos do dispositivo de entrada e mapeados para os endereços de memória. Neste exemplo $arg1$ é int, ou seja, está sendo informado que os valores lidos do dispositivo de entrada serão números inteiros.

Já no segundo argumento você deve utilizar o comando $input().split()$. O comando $intput()$ informa a leitura no dispositivo de entrada, já o comando $split()$ quer dizer que a leitura no dispositivo de entrada será quebrada conforme o número de endereços informados no lado esquerdo do operador de atribuição.

Sabendo ler mais de um dado por linha no dispositivo de entrada veja a codificação do algoritmo elaborado na etapa de planejamento.

Vale ressaltar que em Python o operador // é o operador que nos dá a parte inteira da divisão.

```
L, D = map(int, input().split())
K, P = map(int, input().split())
Cq = L * K
Cp = (L//D) * P
Ct = Cq + Cp
print(Ct)
```

Testando

Você pode utilizar os casos de teste apresentados no enunciado da questão para testar o seu programa. Também é recomendado que você produza outros casos de teste e verifique se o resultado será conforme o esperado.

4.6 Questão 3: Tomadas

Esta questão foi adaptada da Olimpíada Brasileira de Informática de 2013.

A Olimpíada Internacional de Informática (IOI, no original em inglês) é a mais prestigiada competição de programação para alunos de ensino médio; seus aproximadamente 300 competidores se reúnem em um país diferente todo ano para os dois dias de prova da competição. Naturalmente, os competidores usam o tempo livre para acessar a Internet, programar e jogar em seus notebooks, mas eles se depararam com um problema: o saguão do hotel só tem uma tomada.

Felizmente, os quatro competidores da equipe brasileira da IOI trouxeram cada um uma régua de tomadas, permitindo assim ligar vários notebooks em uma tomada só; eles também podem ligar uma régua em outra para aumentar ainda mais o número de tomadas disponíveis. No entanto, como as réguas têm muitas tomadas, eles pediram para você escrever um programa que, dado o número de tomadas em cada régua, determina quantas tomadas podem ser disponibilizadas

no saguão do hotel.

Entrada

A entrada consiste de uma linha com quatro inteiros positivos T1, T2, T3, T4, indicando o número de tomadas de cada uma das quatro réguas.

Saída

Seu programa deve imprimir uma única linha contendo um único número inteiro, indicando o número máximo de notebooks que podem ser conectados num mesmo instante.

Exemplo

Entrada	Saída
2 4 3 2	8
6 6 6 6	21
1000 1001 2000 2001	5999
99 101 200 122	519

Pensando

Para resolver este problema você deve perceber que a primeira ($T1$) régua será ligada na tomada, a segunda régua ($T2$) será ligada em uma das entradas da régua $T1$, a terceira régua será ligada em uma das entradas da régua $T2$ e a quarta régua será ligada em uma das entradas da régua $T3$.

No exemplo da figura **??** a régua $T1$ e $T2$ possuem quatro entradas e as réguas $T3$ e $T4$ três entradas.

Observe na figura que sempre uma das entradas da régua $T1$ será ocupada para ligar a régua $T2$. Da mesma forma, sempre uma das entradas da régua $T2$ será ocupada para ligar a régua $T3$. E, uma das entradas da régua $T3$ será ocupada para ligar a régua $T4$.

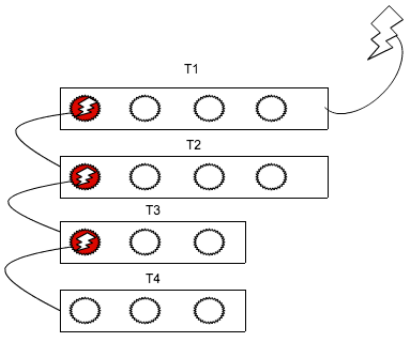


Figura 4.6: Réguas de tomadas interligadas

Observe também que a régua $T4$ sempre ficará com todas as suas entradas livres.

Realizada esta analise, considerando $T1$, $T2$, $T3$ e $T4$ o número de entradas das réguas um, dois, três e quatro respectivamente temos que o número de computadores que podem ser ligados é dado pela fórmula: $Nt = T1-1+T2-2+T3-1+T4$.

A subtração em uma unidade nas réguas $T1$, $T2$ e $T3$ representam justamente uma das entradas sendo ocupadas para ligar a régua subsequente.

Planejando

Antes de escrever o algoritmo não esqueça que ele deve ser estruturado em três partes: entrada, processamento e saída.

A entrada, como dito no enunciado, se dá em uma linha contendo quatro números inteiros que representam respectivamente o número de entradas nas réguas $T1$, $T2$, $T3$ e $T4$.

O processamento, como analisado na etapa do pensar, será $Nt = T1 - 1 + T2 - 2 + T3 - 1 + T4$.

A saída consiste em exibir o resultado do processamento a ser realizado.

Diante dessas observações veja o algoritmo resultante:

```
INÍCIO
    LER(T1,T2, T3, T4)
    Nt = T1-1 + T2-1 + T3-1 + T4
    ESCREVER(Nt)
FIM
```

Vamos praticar a leitura deste algoritmo:

- **LER(T1,T2,T3,T4):** Leia do dispositivo de entrada o número de entradas das réguas 1, 2, 3 e 4 e armazene respectivamente nos endereços de memória T1, T2, T3 e T4.

- **Nt = T1-1 + T2-1 + T3-1 + T4:** Recupere os valores armazenados nos endereços T1, T2, T3 e T4, realize o cálculo e armazene o resultado no endereço Nt.

- **ESCREVER(Nt):** Recupere da memória o valor armazenado no endereço Nt e escreva no dispositivo de saída.

Codificando

Assim como na questão anterior, para codificar este algoritmo você precisa apenas saber como ler mais de uma informação por linha do dispositivo de entrada.

Para fazer isso você deve lembrar que precisa utilizar o comando map(). Caso não lembre, você pode revisar a questão anterior ou tirar sua dúvida lendo o capítulo 9 que trata dos principais comandos da linguagem Python.

Vejamos a codificação deste algoritmo:

```
T1, T2, T3, T4 = map(int, input().split())
Nt = T1-1 + T2-1 + T3-1 + T4
print(Nt)
```

Testando

Você pode utilizar os casos de teste apresentados no enunciado da questão para testar o seu programa. Também é recomendado que você produza outros casos de teste e verifique se o resultado será conforme o esperado.

4.7 Questão 4: Corrida

Esta questão foi adaptada Olimpíada Brasileira de Informática 2012.

Leonardo é um corredor profissional que participa de diversos campeonatos de atletismo pelo mundo. O tamanho das pistas ao redor do mundo não é padronizado. Por isso, Leonardo, que treina em um clube que possui uma pista circular, resolveu fixar seu treinamento em C metros, ao invés de um número fixo de voltas na pista. Após cada treinamento, Leonardo deve tomar meio litro de água antes de fazer qualquer esforço, e por isso quer deixar sua garrafa de água exatamente no ponto da pista onde ele termina o seu treinamento.

Sabendo o comprimento da pista de corrida que Leonardo pretende treinar, ele resolveu pedir sua ajuda para calcular o local do ponto de término do treinamento. O ponto de término é o local da pista onde ele termina o percurso de C metros considerando que ele parte do ponto de partida e se movimenta sempre na mesma direção. O ponto de partida sempre é igual a zero. Por exemplo, se a pista tem 12 metros e Leonardo fixou seu treinamento em 22 metros, o ponto de término é 10. Sua tarefa é: dado o número C de metros que Leonardo

pretende correr e o comprimento N em metros da pista circular, determinar o ponto de término de seu treinamento.

Entrada

A entrada consiste de dois inteiros C e N que indicam, respectivamente, o número de metros que Leonardo pretende correr e o comprimento da pista.

Saída

Seu programa deve imprimir apenas uma linha, contendo apenas um inteiro, indicando o ponto de término do treinamento de Leonardo.

Exemplo

Entrada	Saída
7000 100	0
918 76	6
45000 152	8
10001 123	38

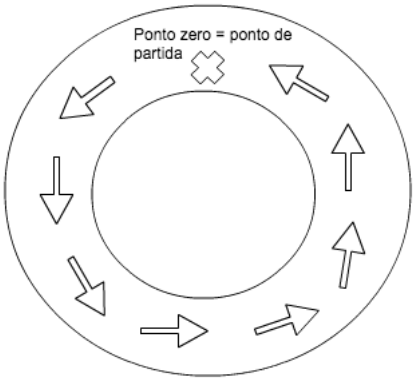
Pensando

Como informado no enunciado da questão, Leonardo irá correr em uma posta circular e quer saber onde deixar sua garrafa tal que coincida com o fim do seu treinamento.

Considerando que a pista tem N metros e é circular, então, Leonardo só poderá deixar sua garrafa em uma das seguintes possibilidades: $\{0, 1, 2, 3, ...N - 1\}$.

Você está lembrado qual o operador possui essa característica? Se não lembrar recomendo a leitura do capítulo 3.

Segundo a definição do resto da divisão, sendo N um número inteiro positivo, o operação $mod\ N$ só poderá resultar em um dos seguintes valores: $\{0, ...N - 1\}$.



Por exemplo, se Leonardo pretende correr 200 metros em uma pista de 30 metros, então deverá deixar sua garrafa a 20 metros do ponto zero pois:

$$\begin{array}{r|l} 200 & 30 \\ \hline 20 & 6 \end{array}$$

Para completar o seu treinamento de 200 metros Leonardo deve dar 6 voltas na pista e correr mais 20 metros.

Você deve lembrar que para encontrar o resto da divisão é necessário utilizar o operador *mod*. Logo, o cálculo do ponto de término do treinamento de Leonardo é dado por $P = C mod N$.

Planejando

Antes de escrever o algoritmo não esqueça que ele deve ser estruturado em três partes: entrada, processamento e saída.

A entrada, como dito no enunciado, se dá em uma linha contendo dois números inteiros que representam, respectiva-

mente, o número de metros que Leonardo pretende correr e o comprimento da pista.

O processamento, como analisado na etapa do pensar, será $P = CmodN$.

A saída consiste em exibir o resultado do processamento a ser realizado.

Diante dessas observações veja o algoritmo resultante:

```
INÍCIO
    LER(C, N)
    P = C mod N
    ESCREVER(P)
FIM
```

Vamos praticar a leitura deste algoritmo:

- **LER(C,N):** Leia do dispositivo de entrada o número de metros e o comprimento da pista. Armazene esses valores nos endereços de memória C e N respectivamente.
- **P = C mod N:** Recupere os valores dos endereços C e N. Calcule o resto da divisão de C por N e armazene o resultado no endereço de memória P.
- **ESCREVER(P):** Recupere o valor de endereço de memória P e escreva no dispositivo de saída.

Codificando

Para codificar o algoritmo elaborado na etapa de planejamento, além de ter que utilizar a função map() para ler mais de um valor na mesma linha do dispositivo de entrada, você também precisa saber como se dá a representação do operador mod na linguagem de programação Python.

Em Python o resto da divisão é calculado com o operador %. Ou seja, $200\%30 = 20$.

Vejamos a codificação do algoritmo:

```
C, N = map(int, input().split())
P = C%N
print(P)
```

Testando

Você pode utilizar os casos de teste apresentados no enunciado da questão para testar o seu programa. Também é recomendado que você produza outros casos de teste e verifique se o resultado será conforme o esperado.

4.8 Exercício

Neste capítulo as questões estão em ordem de dificuldade crescente. Recomendo que você resolva cada questão utilizando a metodologia PPCT conforme apresentada neste capítulo.

Questão 1: Pneu

Esta questão foi adaptada da Olimpíada Brasileira de Informática de 2010.

Calibrar os pneus do carro deve ser uma tarefa cotidiana de todos os motoristas. Para isto, os postos de gasolina possuem uma bomba de ar. A maioria das bombas atuais são eletrônicas, permitindo que o motorista indique a pressão desejada num teclado. Ao ser ligada ao pneu, a bomba primeiro lê a pressão atual e calcula a diferença de pressão entre a desejada

e a lida. Com esta diferença ela esvazia ou enche o pneu para chegar na pressão correta.

Sua ajuda foi requisitada para desenvolver o programa da próxima bomba da SBC - Sistemas de Bombas Computadorizadas.

Você deve escrever um programa que, dada a pressão desejada digitada pelo motorista e a pressão do pneu lida pela bomba, indica a diferença entre a pressão desejada e a pressão lida.

Entrada

Cada caso de teste possui dois inteiros N e M que representam respectivamente a pressão desejada pelo motorista e a pressão lida pela bomba.

Saída

Seu programa deve imprimir uma única linha, contendo a diferença entre a pressão desejada e a pressão lida.

Exemplo

Entrada	Saída
30 18	12
27 27	0
27 30	-3

Questão 2: Dadinhos

Esta questão foi adaptada do Warmup do 8^o Contest Noturno.

Um Dadinho é um doce muito popular no Brasil, por ser gostoso e barato. É tão barato que Rodovalho, sempre que toma café com seus amigos, compra vários Dadinhos para serem comidos enquanto o café é bebido.

Há N pessoas no grupo de amigos de Rodovalho, e ele quer comprar D dadinhos para cada pessoa. Considerando que cada dadinho custa C centavos, quanto Rodovalho irá gastar?

Entrada

Cada caso contém uma única linha contendo três valores, N, C e D.

Saída

Para cada caso de teste, imprima uma única linha contendo um único inteiro, indicando quanto Rodovalho irá gastar, em centavos.

Exemplo

Entrada	Saída
3 2 15	90
1 1 1	1

Questão 3: Ano novo

Esta questão foi adaptada do Warmup do 7^o Contest Noturno.

Hoje é 31 de dezembro, véspera de ano novo! Várias pessoas costumam comemorar esta data com fogos de artifício, champanhe, uvas, etc.

Além disso, também é comum fazer uma contagem regressiva, em segundos, para a meia-noite. Assim, se a contagem começar, por exemplo, às 23:59:50, uma contagem de 10 segundos é feita: a sequência (10, 9, 8, ..., 2, 1, 0) é falada, um número por segundo. Ao término da sequência, será exatamente meia-noite, e o novo ano iniciará.

Você foi contratado para escrever um programa que dado o horário de início da contagem regressiva, determine por quantos segundos ela deve durar.

Entrada

Cada caso de teste é descrito em uma linha contendo três inteiros H, M, S que representam o horário do início da contagem. O horário de entrada sempre estará entre (00:00:00 a 23: 59:59).

Saída

Para cada caso de teste, imprima uma única linha, contendo a quantidade de segundos que a contagem deve durar.

Exemplo

Entrada	Saída
23 59 50	10
23 59 00	60
00 00 00	86400

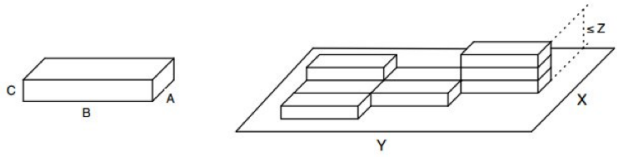
Questão 4: Transporte

Esta questão foi adaptada da Olimpíada Brasileira de Informática de 2011.

A Betalândia é um país que apenas recentemente se abriu para o comércio exterior e está preparando agora sua primeira grande exportação. A Sociedade Betalandesa de Comércio (SBC) ficou encarregada de conduzir a exportação e determinou que, seguindo os padrões internacionais, a carga será transportada em contêineres, que são, por sua vez, colocados em grandes navios para o transporte internacional.

Todos os contêineres betalandeses são idênticos, medindo A metros de largura, B metros de comprimento e C metros de altura. Um navio porta-contêineres pode ser visto como um retângulo horizontal de X metros de largura e Y metros de comprimento, sobre o qual os contêineres são colocados.

Nenhuma parte de contêiner pode ficar para fora do navio. Além disso, para possibilitar a travessia de pontes, a altura máxima da carga no navio não pode ultrapassar Z metros.



Devido a limitações do guindaste utilizado, os contêineres só podem ser carregados alinhados com o navio. Ou seja, os contêineres só podem ser colocados sobre o navio de tal forma que a largura e o comprimento do contêiner estejam paralelos à largura e ao comprimento do navio, respectivamente.

A SBC está com problemas para saber qual a quantidade máxima de contêineres que podem ser colocados no navio e pede sua ajuda. Sua tarefa, neste problema, é determinar quantos contêineres podem ser carregados no navio respeitando as restrições acima.

Entrada

A primeira linha contém três inteiros A, B e C que representam as dimensões dos contêineres, enquanto a segunda linha contém outros três inteiros X, Y e Z que representam as dimensões do navio.

Saída

Seu programa deve imprimir apenas uma linha contendo um inteiro que indica a quantidade máxima de contêineres que o navio consegue transportar.

Exemplo

Entrada	Saída
1 1 1 1 1 1	1
1 2 5 9 6 11	54
1 2 12 6 9 10	0

Questão 5: Bits trocados

Esta questão foi adaptada da Olimpíada Brasileira de Informática de 2000.

As Ilhas Weblands formam um reino independente nos mares do Pacífico. Como é um reino recente, a sociedade é muito influenciada pela informática. A moeda oficial é o Bit; existem notas de B$ 50,00, B$10,00, B$5,00 e B$1,00. Você foi contratado para ajudar na programação dos caixas automáticos de um grande banco das Ilhas Weblands.

Os caixas eletrônicos das Ilhas Weblands operam com todos os tipos de notas disponíveis, mantendo um estoque de cédulas para cada valor (B$ 50,00, B$10,00, B$5,00 e B$1,00). Os clientes do banco utilizam os caixas eletrônicos para efetuar retiradas de um certo número inteiro de Bits.

Sua tarefa é escrever um programa que, dado o valor de Bits desejado pelo cliente, determine o número de cada uma das notas necessário para totalizar esse valor, de modo a minimizar a quantidade de cédulas entregues. Por exemplo, se o cliente deseja retirar B$50,00, basta entregar uma única nota de cinquenta Bits. Se o cliente deseja retirar B$72,00, é necessário entregar uma nota de B$50,00, duas de B$10,00 e duas de B$1,00.

Entrada

Cada conjunto de teste é composto por uma única linha, que contém um número inteiro positivo V, que indica o valor solicitado pelo cliente.

Saída

Na saída o seu programa deve imprimir quatro inteiros I, J, K e L que representam o resultado encontrado pelo seu programa: I indica o número de cédulas de B$50,00, J indica o número de cédulas de B$10,00, K indica o número de cédulas de B$5,00 e L indica o número de cédulas de B$1,00.

Exemplo

Entrada	Saída
1	0 0 0 1
72	1 2 0 2

Questão 6: Distância de Manhattan

Esta questão foi adaptada da Olimpíada Brasileira de Informática de 2013.

Maria é uma moradora de Nlogópolis, uma cidade na Nlogônia que tem uma característica muito interessante: todas as ruas da cidade ou são orientadas no sentido norte-sul ou são orientadas no sentido Leste-Oeste. Isso significa que, dadas duas ruas, ou elas são paralelas ou elas são perpendiculares entre si.

Todas as ruas da cidade são de mão dupla e é possível seguir em qualquer direção em um cruzamento. Agora Maria está atrasada para uma reunião e precisa de sua ajuda. Dadas as coordenadas iniciais de Maria e da reunião, determine o número mínimo de cruzamentos que Maria deve atravessar para chegar ao seu destino. Esse número inclui o cruzamento onde ocorrerá a reunião, mas não inclui a posição inicial de Maria.

Entrada

A única linha da entrada contém quatro inteiros, Xm, Ym, Xr, Yr, indicando as coordenadas de Maria (Xm, Ym) e da reunião (Xr, Yr). O ponto de partida de Maria nunca será igual ao local da reunião, ou seja, pelo menos uma das coordenadas será diferente.

Saída

Seu programa deve imprimir uma única linha contendo um único inteiro: o número mínimo de cruzamentos que Maria precisa atravessar para chegar até o local da reunião.

Exemplo

Entrada	Saída
0 0 5 6	11
52 75 120 75	68

Questão 7: Tapetes

Esta questão foi adaptada da Olímpiada Brasileira de Informática de 2014.

Nlogonia é conhecida por sua indústria de tradicionais tapetes quadrados, que são produzidos apenas com dimensões inteiras, para todos os números inteiros positivos. Quer dizer, os tapetes são de dimensão 1×1, 2×2, 3×3, e assim por diante. João Tapetão, grande empresário do setor, está planejando o próximo carregamento para exportação, que deve ser de exatamente N tapetes. Os tapetes são sempre enrolados e colocados em um tubo, um após o outro.

Por exemplo, para um carregamento de $N = 4$ tapetes de dimensões 2×2, 4×4, 6×6 e 3×3, será necessário um tubo de comprimento $2 + 4 + 6 + 3 = 15$. A questão é que o

preço do tapete é proporcional à sua área, de modo que quanto maior a soma das áreas dos tapetes, maior o lucro do Tapetão.

No exemplo anterior, a soma das áreas é $22 + 42 + 62 + 32 = 65$. Só que daria para lucrar mais, com o mesmo tubo de comprimento 15, se o carregamento fosse com quatro tapetes de dimensões 1×1, 4×4, 7×7 e 3×3, cuja soma das áreas dá 75. Será que daria para lucrar ainda mais?

O navio chegou e Tapetão precisa embarcar o carregamento. Há apenas um tubo de comprimento L e o carregamento deve conter exatamente N tapetes. Tapetão quer saber qual é a maior soma possível das áreas dos N tapetes que poderá ser transportada? Para isso Tapetão lhe contratou para construir um programa para resolver este problema.

Entrada

A primeira e única linha da entrada contém dois inteiros, L e N, o comprimento do tubo e a quantidade de tapetes que deve transportada, respectivamente.

Saída

Seu programa deve produzir uma única linha, contendo apenas um inteiro, a maior soma possível das áreas dos tapetes.

Exemplo

Entrada	Saída
2 2	2
10 5	40
1000000 9	999984000072

Questão 8: Chocolate

Esta questão foi adaptada da Olimpíada Brasileira de Informática 2012.

Por lei, na Nlogônia todas as barras de chocolate são quadradas. Ana tem uma barra quadrada de chocolate de lado L, que ela quer compartilhar com alguns colegas da obi. Mas ela é uma boa cidadã e cumpre a lei. Então, ela divide a barra em quatro pedaços quadrados, de lado L=L/2. Depois, ela repete esse procedimento com cada pedaço gerado, sucessivamente, enquanto o lado for maior do que, ou igual a 2cm. Você deve escrever um programa que, dado o lado L da barra inicial, em centímetros, determina quantos pedaços haverá ao final do processo.

Entrada

A entrada consiste de uma linha, com um único inteiro, L, o número de centímetros do lado do quadrado.

Saída

Seu programa deve imprimir uma única linha, contendo um único inteiro, igual ao número total de pedaços obtidos pela Ana.

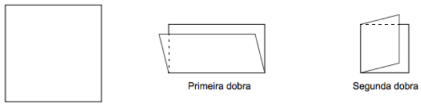
Exemplo

Entrada	Saída
4	16
9	64
256	65536

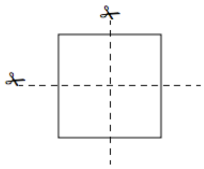
Questão 9: Dobra

Zezinho tem aulas de Iniciação Artística em sua escola, e recentemente aprendeu a fazer dobraduras em papel. Ele ficou fascinado com as inúmeras possibilidades de se dobrar uma simples folha de papel. Como Zezinho gosta muito de matemática, resolveu inventar um quebra-cabeça envolvendo dobraduras. Zezinho definiu uma operação de dobradura D que

consiste em dobrar duas vezes uma folha de papel quadrada de forma a conseguir um quadrado com 1/4 do tamanho original, conforme ilustrado na figura.



Depois de repetir N vezes esta operação de dobradura D sobre o papel, Zezinho cortou o quadrado resultante com um corte vertical e um corte horizontal, conforme a figura abaixo.



Zezinho lançou então um desafio aos seus colegas: quem adivinha quantos pedaços de papel foram produzidos?

Entrada

Cada conjunto de teste é composto de uma única linha, contendo um número inteiro N que indica o número de vezes que a operação de dobradura D foi aplicada.

Saída

Seu programa deve calcular e imprimir número de pedaços de papel obtidos depois de cortar a dobradura.

Exemplo

Entrada	**Saída**
1	9
0	4
10	1050625

Capítulo 5

Operadores lógicos

No capítulo 2 você aprendeu que o processador possui uma unidade lógica e aritmética (ULA). A ULA implica que um computador só pode realizar dois tipos de processamento. Ou o processamento é lógico ou é aritmético.

Essa peculiaridade também reflete na possibilidade da construção de algoritmos. Sabendo que um algoritmo é um projeto computacional que deve ser codificado em uma linguagem de programação. Então, ao escrever um algoritmo, fica claro que devemos utilizar operadores lógicos e aritméticos para construir o processamento da solução.

No capítulo anterior utilizamos a metodologia PPCT para resolver problemas que necessitavam apenas de processamento aritmético. No próximo capítulo iremos utilizar a mesma metodologia para resolver problemas que, além de processamento aritmético, também necessite de processamento lógico.

Sendo assim, o presente capítulo tem o objetivo de lhe apresentar os operadores lógicos e relacionais para que você saiba como construir expressões lógicas que devem ser utilizadas nos mecanismos de controle fluxo de um programa.

5.1 Expressão lógica simples

Uma expressão lógica é aquela cujo resultado só pode ser verdadeiro ou falso.

As expressões lógicas podem ser classificadas como expressões lógicas simples ou compostas. Uma expressão lógica simples possui a seguinte sintaxe: $EXP_A\ OR\ EXP_A$.

EXP_A representa uma expressão aritmética. Em seu caso mais simples, uma expressão aritmética (EXP_A) é formada por um valor constante ou por uma variável.

Um valor constante é um valor imutável, por exemplo, $1, 2, 3, 4, 5$, etc. Já uma variável é um valor mutável que nor-

malmente é representado por uma letra.

Levando em consideração a arquitetura de computador, mais especificamente a memória RAM, uma variável se equivale a um endereço de memória.

Segundo a ilustração da figura xx, podemos considerar os endereços A, B e C como variáveis e $32, 22$, e 10 os valores armazenados em cada uma.

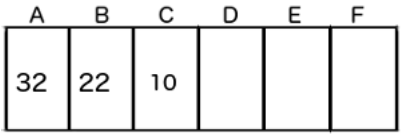


Figura 5.1: Variáveis.

OR representa os operadores relacionais são representados pelo conjunto: $\{>, <, ==, \neq, \geq, \leq\}$

Vejamos alguns exemplos de expressões lógicas simples no formato $EXP_A\ OR\ EXP_A$:

$A > 2 \qquad B == A \qquad A\ mod\ 2 \neq 0 \qquad (AV1 + AV2)/2 \geq 7 \qquad K**2 < 2**K$

Observe que, conforme a definição, todas as expressões exemplificadas só podem resultar em verdadeiro ou falso.

Vale ressaltar que em uma expressão lógica os operadores aritméticos têm maior ordem de precedência em relação aos operadores relacionais, ou seja, sempre resolva primeiro as expressões aritméticas para depois resolver a expressão lógica.

No caso da expressão lógica $(AV1 + AV2)/2 \geq 7$, considerando $AV1 = 10$ e $AV2 = 5$ temos que:

- $(AV1 + AV2)/2 \geq 7$
- $(10 + 5)/2 \geq 7$
- $(15)/2 \geq 7$

- $7.5 \geq 7$
- $Verdadeiro$

Vejamos mais um exemplo! Considerando $k = 3$ e a expressão lógica $k * *2 < 2 * *k$ temos que:

- $k * *2 < 2 * *k$
- $3 * *2 < 2 * *3$
- $9 < 8$
- $Falso$

5.2 Expressão lógica composta

Uma expressão lógica composta pode ser definida como a reunião de duas ou mais expressões lógicas simples unidas por operadores lógicos. Considerando EXP_{Ls} uma expressão lógica simples e OL um operador lógico, uma expressão lógica composta possui a seguinte sintaxe: $EXP_{Ls}\ OL\ EXP_{Ls}$.

Conforme a definição de expressão lógica simples: $EXP_{Ls} = EXP_A\ OR\ EXP_A$.

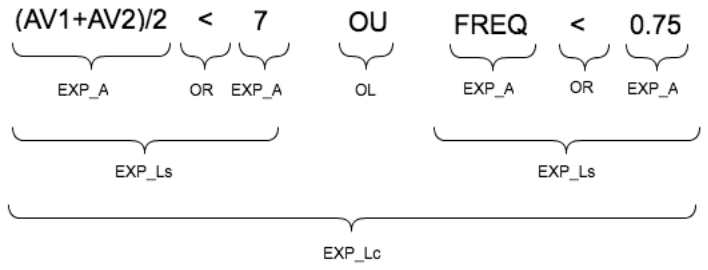
Podemos reescrever a sintaxe de expressão lógica composta de: $EXP_{Ls}\ OL\ EXP_{Ls}$

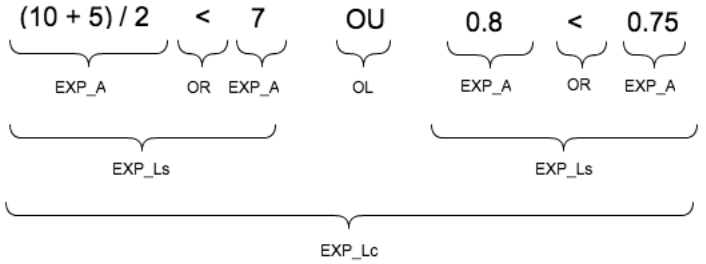
Para: $EXP_A\ OR\ EXP_A\ OL\ EXP_A\ OR\ EXP_A$.

Vamos traduzir:

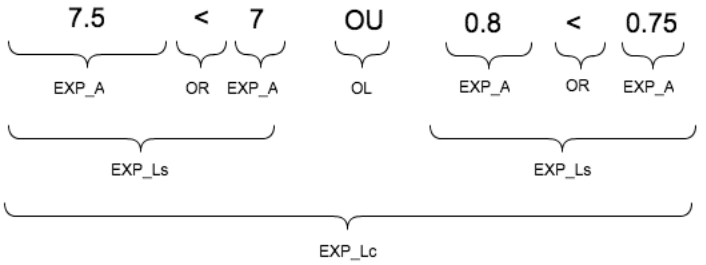
Considerando $AV1 = 10$, $AV2 = 5$ e $FREQ = 0.8$, vamos resolver esta expressão lógica composta.

Primeiro devemos substituir as variáveis pelos valores mencionados:

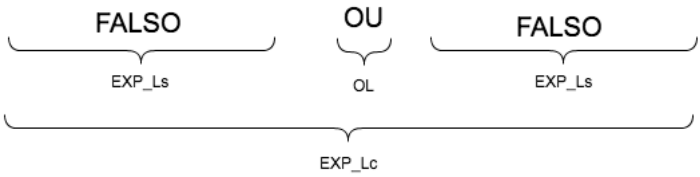




Realizada a substituição das variáveis pelos seus respectivos valores, agora, devemos resolver todas as expressões aritméticas (EXP_A).



Quando todas as expressões aritméticas (EXP_A) estiverem em seus valores absolutos você já pode resolver as expressões lógicas simples (EXP_Ls).



Após resolver todas as expressões lógicas simples você deve resolver a expressão lógica composta.

Observe que neste exemplo a expressão lógica composta utilizou o operador lógico "OU" (OL= OU) e que o resultado da expressão foi falso.

Você deve está se perguntando! Porque o resultado foi falso? Quando é que dá verdadeiro? Quais são os outros operadores lógicos além do "OU"?

Vamos responder as perguntas de trás para frente. Neste livro iremos utilizar dois operadores lógicos: "OU" e "E".

Quando o operador lógico "OU" é utilizado, leve em consideração a seguinte regra para saber se expressão lógica é verdadeira ou falsa:

Tabela 5.1: Tabela verdade do perador lógico OU

EXP_{Ls}	OU	EXP_{Ls}	$=$	EXP_{Lc}
$Verdadeiro$	OU	$Verdadeiro$	$=$	$Verdadeiro$
$Verdadeiro$	OU	$Falso$	$=$	$Verdadeiro$
$Falso$	OU	$Verdadeiro$	$=$	$Verdadeiro$
$Falso$	OU	$Falso$	$=$	$Falso$

Observe que, com o operador lógico "OU", a expressão lógica só será falsa quando as duas expressões lógicas simples também forem falsas. Em qualquer outro caso a expressão lógica composta será verdadeira.

Já para o operador lógico "E", como pode ser observado na tabela xx, a expressão lógica só será verdadeira quando as duas expressões lógicas forem verdadeiras. Em qualquer outro caso a expressão lógica composta será falsa.

Tabela 5.2: Tabela verdade do perador lógico E

EXP_{Ls}	E	EXP_{Ls}	$=$	EXP_{Lc}
$Verdadeiro$	E	$Verdadeiro$	$=$	$Verdadeiro$
$Verdadeiro$	E	$Falso$	$=$	$Falso$
$Falso$	E	$Verdadeiro$	$=$	$Falso$
$Falso$	E	$Falso$	$=$	$Falso$

Em certos casos também existe a necessidade de construir expressões lógicas compostas com a reunião de mais de duas expressões lógicas simples. Por exemplo, considerando os valores das variáveis $a = 10$, $b = 15$, $c = 5$ e $d = 20$, vamos descobrir o valor lógico da seguinte expressão:

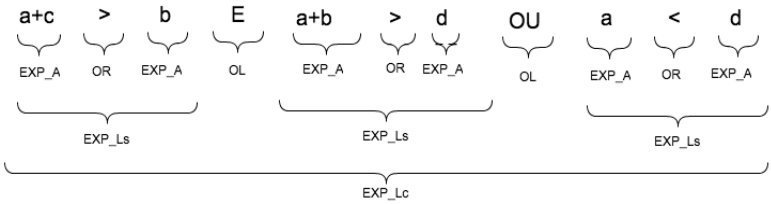
- $a + c > b \; E \; a + b > d \; OU \; a < d$

Observe que a expressão é uma expressão lógica composta que reúne três expressões lógicas simples. Qual o resultado desta expressão? Quando uma expressão lógica composta usa tanto o operador lógico OU e E qual você deve resolver primeiro?

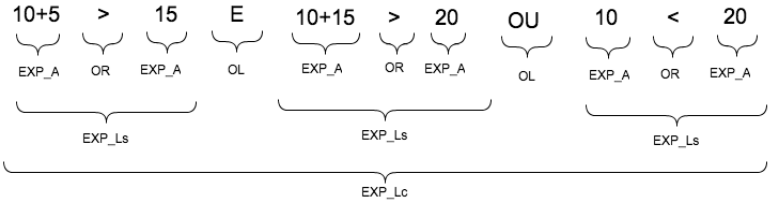
Saber a ordem de precedência dos operadores lógicos E e OU é muito importante para não produzir resultados equivocados. Então preste atenção! Sempre resolva primeiro o operador lógico E e depois resolva o operador lógico OU.

O operador lógico E possui maior precedência em relação ao operador lógico OU.

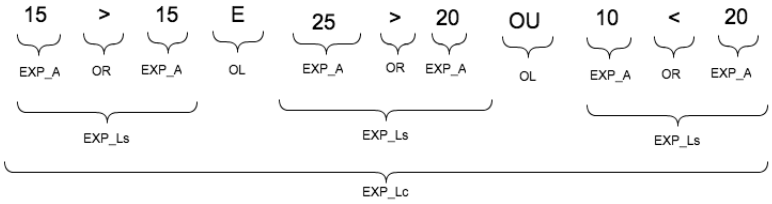
Agora vamos resolver a expressão e descobrir o seu valor lógico:



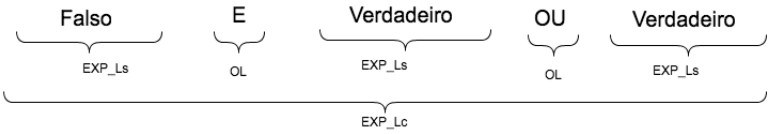
Primeiro você deve substituir as variáveis pelos seus respectivos valores.



Realizada a substituição, o segundo passo consiste em resolver todas as expressões aritméticas (EXP_A).



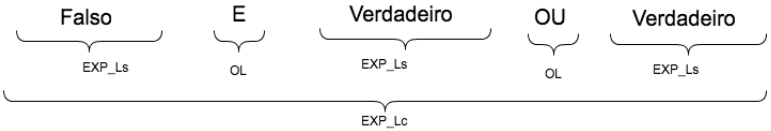
No terceiro passo você deve resolver todas as expressões lógicas simples (EXP_{Ls}).

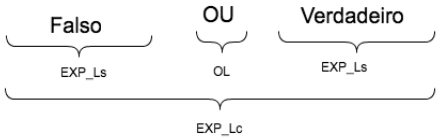


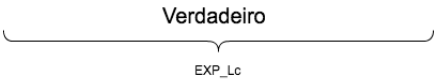
No quarto passo você deve resolver a expressão lógica composta. Porém, observe que temos o operador lógico OU e E.

Neste caso, você já sabe que sempre deve resolver primeiro o operador lógico E e depois o operador lógico OU.

Para este exemplo temos que Falso E Verdadeiro = Falso, logo:







Vamos escrever mais uma expressão lógica!

Considerando a variáveis a, b e c, escreva uma expressão lógica para que seja verdadeira toda vez que o segundo maior valor for o da variável “a”.

Vamos atribuir valores as variáveis tal satisfaçam o enunciado e descobrir o padrão:

- a = 2, b = 1 e c = 3
- a = 15, b = 20 e c = 10

A partir desses exemplos observe que para a variável "a" ter o segundo maior valor, ou a variável "b" ou a variável "c" tem que ter o maior valor.

Trabalhando com hipótese de que a variável "c" tem o maior valor, então, para que a expressão lógica seja verdade quando a variável "a" tiver o segundo maior valor é: $c > a \ E \ a > b$.

Se testarmos essa expressão com os valores a=2, b=1 e c=3, veremos que a expressão é verdadeira. Por outro lado, para os valores a=15, b=20 e c= 10 a expressão resulta no valor lógico falso.

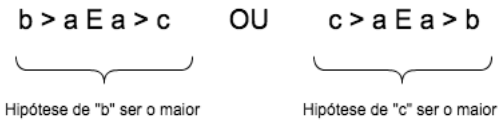
Isso quer dizer que a expressão $c > a \ E \ a > b$ está incompleta, pois no exemplo a=15, b=20 e c=10 o valor de da variável "a" é o segundo maior.

Para completar a expressão também temos que assumir a hipótese da variável "b" ser a maior. Neste caso, a expressão lógica que resulta em verdade é: $b > a \ E \ a > c$.

Realizando os testes para essa expressão, veremos que os valores a=15, b=20 e c=10 resultarão em verdade. Já para os valores a=2, b=1 e c=3 a expressão resulta no valor lógico falso.

Sendo assim, sabendo que para a variável "a" possuir o segundo maior valor, ou a variável "b" ou a variável "c" tem que ter o maior valor. A expressão lógica completa que resolve este problema é: $b > a \ E \ a > c \ OU \ c > a \ E \ a > b$.

Perceba, mais uma vez, que a parte mais difícil é pensar em como resolver o problema. Depois da fase do pensar, construir a expressão lógica não é uma tarefa tão complicada.



Elabore vários casos de teste e veja que esta expressão lógica só será verdadeira quando o valor da variável "a" for o segundo maior.

5.3 Exercício

1. Considerando as varáveis a=10, b=5, c=2, d=7, e=18 resolva passo a passo as expressões lógicas.

   (a) $c * *b > b * *c$

   (b) $e < a \ OU \ c + 1 == 10 \ mod \ 7$

   (c) $e \ div \ b > 10 \ mod \ 7 \ OU \ b * c > d - e \ E \ log_c \ 32 == c * *b$

   (d) $a - b == a + b - a \ E \ e \ div \ d > c$

   (e) $a > b \ OU \ c < d \ E \ e == a + d \ OU \ b == c \ E \ a + b + c < d + e$

2. Escreva uma expressão lógica para que seja verdadeira quando o valor da variável "a" for par.

3. Escreva uma expressão lógica para que seja verdadeira quando o valor da variável "a" for um número natural.

4. Escreva uma expressão lógica para que seja verdadeira quando o valor da variável "a" for negativo.

5. Considerando os valores das variáveis "a", "b" e "c", escreva uma expressão lógica para que seja verdadeira quando as medidas de "a", "b" e "c" formar um triângulo.

6. Considerando "x" e "y" os valores de um ponto $P_{x,y}$ no plano cartesiano, escreva uma expressão lógica que seja verdadeira quando o ponto estiver no primeiro quadrante.

7. Considerando "x" e "y" os valores de um ponto $P_{x,y}$ no plano cartesiano, escreva uma expressão lógica que seja verdadeira quando o ponto estiver no segundo quadrante.

8. Considerando "x" e "y" os valores de um ponto $P_{x,y}$ no plano cartesiano, escreva uma expressão lógica que seja verdadeira quando o ponto estiver no terceiro quadrante.

9. Considerando "x" e "y" os valores de um ponto $P_{x,y}$ no plano cartesiano, escreva uma expressão lógica que seja verdadeira quando o ponto estiver no quarto quadrante.

10. Considerando "x" e "y" os valores de um ponto $P_{x,y}$ no plano cartesiano, escreva uma expressão lógica que seja verdadeira quando o ponto estiver na origem.

11. Considerando "x" e "y" os valores de um ponto $P_{x,y}$ no plano cartesiano, escreva uma expressão lógica que seja verdadeira quando o ponto estivar somente sobre o eixo X.

12. Considerando "x" e "y" os valores de um ponto $P_{x,y}$ no plano cartesiano, escreva uma expressão lógica que seja verdadeira quando o ponto estivar somente sobre o eixo Y.

13. Considerando "x1" e "y1" os valores do ponto $P_{x1,y1}$ e "x2" e "y2" os valores do ponto $P_{x2,y2}$, escreva uma expressão lógica para que seja verdadeira quando a reta formada por esses pontos for crescente.

14. Considerando "x1" e "y1" os valores do ponto $P_{x1,y1}$ e "x2" e "y2" os valores do ponto $P_{x2,y2}$, escreva uma expressão

lógica para que seja verdadeira quando a reta formada por esses pontos for decrescente.

15. Considerando "x1" e "y1" os valores do ponto $P_{x1,y1}$ e "x2" e "y2" os valores do ponto $P_{x2,y2}$, escreva uma expressão lógica para que seja verdadeira quando a reta formada por esses pontos estiver na horizontal.

16. Considerando "x1" e "y1" os valores do ponto $P_{x1,y1}$ e "x2" e "y2" os valores do ponto $P_{x2,y2}$, escreva uma expressão lógica para que seja verdadeira quando a reta formada por esses pontos estiver na vertical.

Capítulo 6

Controle de fluxo de seleção

Até o momento, todos os algoritmos que construímos podem ser categorizados como algoritmos de fluxo linear. Um algoritmo de fluxo linear é aquele que, para qualquer dado de entrada, executa todas alinhas do código somente uma única vez.

```
1  INÍCIO
2      LER(A, B)
3      C = A + B
4      ESCREVER(C)
5  FIM
```

Figura 6.1: Algoritmo linear.

A Figura **??** ilustra um algoritmo linear para realizar a soma de dois números lidos a partir do dispositivo de entrada. Observe que, independentemente dos valores lidos e armazenados nos endereços "A" e "B", sempre o algoritmo irá processar todas as linhas.

Neste capítulo, iremos estudar as classes de problemas que necessitem de algoritmos de fluxo não linear. Os algoritmos de fluxo não linear são aqueles que, dependendo do dado de entrada, algumas linhas do código não são executadas ou algumas linhas são executadas diversas vezes.

Os algoritmos que possuem o comportamento que, dependendo do dado de entrada, algumas linhas do código não são executadas, são construídos utilizando o conceito de controle de fluxo de seleção.

Estudar controle de fluxo de seleção e saber como utilizá-lo para resolver problemas computacionais é o objetivo deste capítulo.

6.1 Controle de fluxo de seleção SE-SENÃO

O controle de fluxo de seleção SE-SENÃO possui o comportamento conhecido como causa-consequência-alternativa. Se a expressão for verdadeira (causa), então, realize este processamento (consequência), senão, realize este outro processamento (alternativa).

Em nosso cotidiano é comum nos depararmos com situações de causa-consequência-alternativa. Veja se você se identifica com alguns dos exemplos:

- Se não chover então vou jogar tênis senão vou jogar basquete.
- Se tirar boas botas então vou passar de ano senão vou reprovar.
- Se eu ganhar a aposta então vou viajar senão vou trabalhar.
- Se eu for promovido vou comprar um carro senão vou comprar uma moto.

Podemos dizer que o controle de fluxo SE-SENÃO é uma estrutura de tomada de decisão bifurcatória, ou seja, para uma determinada condição, definimos uma atividade específica a ser realizada se condição for verdadeira e outra atividade específica que só será realizada se a condição for falsa.

A Figura **??** exemplifica o fluxo desta estrutura. Neste exemplo, a condição que irá determinar qual a atividade a ser realizada é a chuva. Se estiver chovendo (causa), então, vou jogar basquete (consequência), senão, vou jogar tênis (alternativa).

Perceba que a bifurcação existe pois não há a possibilidade de jogar basquete e tênis ao mesmo tempo. Definir o fluxo a ser seguido em função de uma condição é o objetivo do controle de fluxo.

Veja também que, independentemente de estar chovendo ou não, após realizar uma das atividades esportivas o próximo passo será tomar banho. Esta atividade que reúne a bifurcação determina o fim das atividades dependentes da condição.

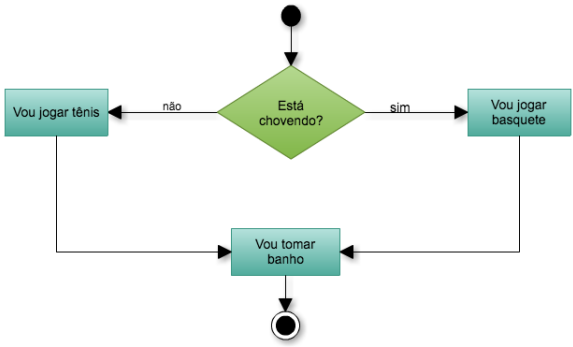


Figura 6.2: Fluxograma se-senão.

Agora que você já conhece o comportamento do controle de fluxo SE-SENÃO, vejamos como representa-lo em algoritmos.

```
1  INÍCIO
2      LER(A)
3      SE A mod 2 == 0 ENTÃO
4          ESCREVER("par")
5      SENÃO
6          ESCREVER("ímpar")
7      FIM-SE
8  FIM
```

Figura 6.3: Algoritmo não linear.

Considerando a definição de que com o controle de fluxo de seleção, dependendo dos dados de entrada, algumas linhas não

serão executadas, você saberia me informar, para A=10, quais as linhas do algoritmo da Figura **??** não seriam executadas?

Sendo A=5, você saberia informar qual a linha deste algoritmo não seria executada?

Para A=10 as linhas 5 e 6 não seriam executadas. Para A=5 somente a linha 4 não seria executada.

Antes de analisarmos o motivo desta resposta e entendermos a representação da estrutura SE-SENÃO, vamos ver o comportamento deste algoritmo em fluxograma.

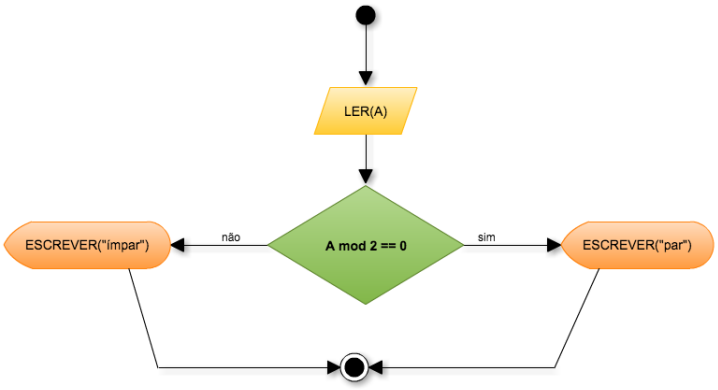


Figura 6.4: Fluxograma não linear.

Observando a Figura **??**, é fácil perceber que o objetivo do controle de fluxo de seleção SE-SENÃO é criar uma bifurcação no nosso algoritmo. A partir da bifurcação, para decidir qual caminho seguir, devemos construir uma expressão lógica.

No caso da representação algorítmica, se o valor da expressão for verdadeiro, serão processadas todas as linhas entre as instruções ENTÃO e SENÃO. No exemplo apresentado, toda vez que a expressão lógica A mod 2 == 0 for verdadeira, será executada a linha ESCREVER(“par”).

De forma complementar, na estrutura SE-SENÃO, as linhas entre as instruções SENÃO e FIM-SE só serão processadas quando a expressão lógica for falsa. No exemplo apresentado, a instrução ESCREVER("ímpar") só será executada quando A mod 2 == 0 for falsa.

Em resumo, devemos utilizar o controle de fluxo de seleção SE-SENÃO, toda vez que precisarmos definir um conjunto de processamento específico em função da verdade de uma expressão lógica e outro conjunto de processamento em função da falsidade desta mesma expressão.

As linhas de código entre as instruções ENTÃO e SENÃO, por serem dependentes da verdade da expressão lógica, são chamadas de bloco de código verdade. Para demonstrar a relação de dependência, recomendasse a utilização da tabulação.

De forma análoga, as linhas de código entre as instruções SENÃO e FIM-SE, por serem dependentes da falsidade da expressão lógica, são chamadas de bloco de código falsidade. Para demonstrar a relação de dependência, também recomendasse a utilização da tabulação.

6.2 Controle de fluxo de seleção SE-ENTÃO

Diferentemente do controle de fluxo SE-SENÃO que determina processamentos específicos em função da verdade e falsidade da expressão lógica, a estrutura SE-ENTÃO prevê somente o bloco de código verdade, ou seja, determina processamentos específicos apenas em função da verdade da expressão lógica.

O controle de fluxo SE-ENTÃO possui o comportamento conhecido como causa-consequência. Se a expressão for verdadeira (causa), então, realize este processamento (consequência).

Se você parar para pensar, o nosso cotidiano também é cercado de atividades com essa característica:

- Se tirar boas notas então passo de ano.
- Se não chover então vou jogar tênis.
- Se resolver toda as questões então vou tirar dez.
- Se não estudar então vou reprovar.
- Se me esforçar então vou conseguir.

Podemos dizer que o controle de fluxo SE-ENTÃO é uma estrutura de tomada de decisão de desvio de fluxo, ou seja, para uma determinada condição, definimos uma atividade específica a ser realizada se, e somente se, condição for verdadeira.

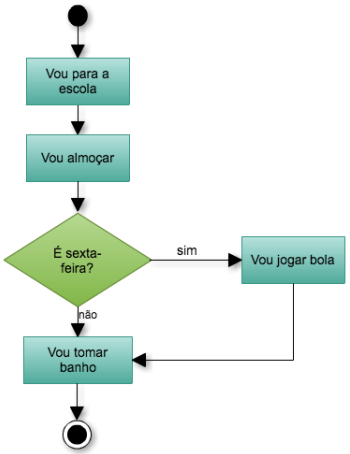


Figura 6.5: Fluxograma não linear.

A Figura **??** exemplifica o fluxo desta estrutura. Neste exemplo, o desvio de fluxo só irá acontecer se for sexta-feira. Toda vez que for sexta-feira (causa) a atividade jogar bola

será realizada (consequência). Podemos dizer que jogar bola é a atividade subordinada a verdade da expressão "É sexta-feira?".

Veja que a única atividade que não é linear, ou seja, que está subordinada a verdade da expressão, é a de jogar bola. Podemos concluir que, dependendo do dia da semana, poderei ou não jogar bola. Este comportamento é o que caracteriza o desvio de fluxo.

Agora que você já sabe o comportamento do controle de fluxo SE-ENTÃO, vejamos como representa-lo em algoritmos.

```
1  INÍCIO
2      LER(A)
3      SE A > 300 ENTÃO
4          A = A-A*0.1
5      FIM-SE
6      ESCREVER("valor a ser pago: ",A)
7  FIM
```

Figura 6.6: Fluxograma não linear.

Neste exemplo, estamos lendo um valor e, caso ele seja maior que 300, estamos descontando 10% deste valor.

Na estrutura SE-ENTÃO as linhas subordinadas à verdade da expressão lógica devem ficar entre as instruções ENTÃO e FIM-SE. Para demonstrar a subordinação, recomenda-se que as linhas entre as instruções ENTÃO e FIM-SE sejam tabuladas.

A figura **??** demonstra o fluxo deste algoritmo e podemos perceber claramente o desvio realizado em função da verdade da expressão.

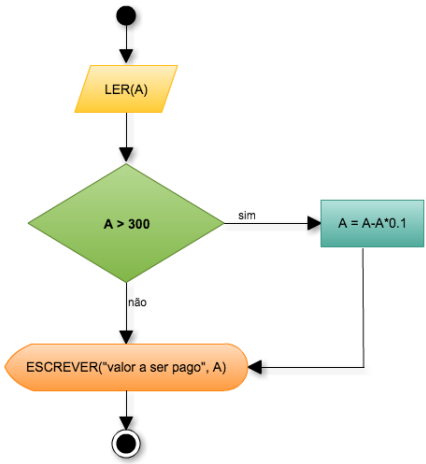


Figura 6.7: Fluxograma não linear.

6.3 Controle de fluxo SE-SENÃOSE

Existem situações em que precisamos utilizar várias expressões lógicas tal que a próxima expressão lógica só seja processada se a antecessora for falsa. Para resolver problemas que necessitem deste procedimento podemos utilizar o controle de fluxo SE-SENÃOSE.

Para exemplificar, vejamos o caso clássico de transformação de nota na escala de 0 a 10 para nota conceitual de A a D.

Perceba que a estrutura encadeada SENÃOSE nos permite terminar com a instrução SENÃO. Neste caso, quando o SENÃO está presente, ele só será executado se nenhuma das expressões lógicas antecessoras forem verdadeiras.

Na maioria das vezes utilizamos a estrutura encadeada SENÃOSE para determinar, em função de expressões lógicas, o valor a ser atribuído a uma determinada variável. No nosso exemplo, estamos utilizando a estrutura encadeada para determinar o valor da variável C.

```
1  INÍCIO
2      LER(A)
3      SE A > 8 e A <= 10 ENTÃO
4          C = "A"
5      SENÃOSE A > 6 e A <= 8 ENTÃO
6          C = "B"
7      SENÃOSE A > 5 e A <= 6 ENTÃO
8          C = "C"
9      SENÃO
10         C = "D"
11     FIM-SE
12     ESCREVER(C)
13 FIM
```

Figura 6.8: Fluxograma não linear.

Assim como nas outras estruturas de seleção, nesta, também se recomenda a tabulação para demonstrar a subordinação.

Vejamos o fluxograma deste algoritmo:

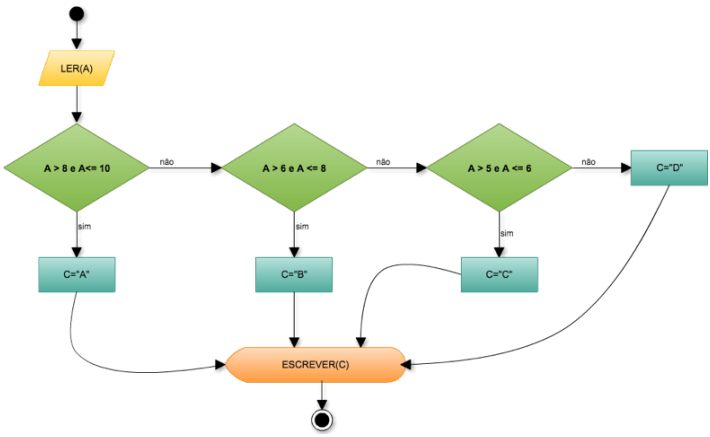


Figura 6.9: Fluxograma não linear.

Na Figura X?? o comportamento SE-SENÃOSE é observado quando percebemos que a aresta "não" de uma condição é incidente em outra condição. Não custa lembrar que a aresta é representada por uma seta, uma condição por um losango e

que incidente significa, neste contexto, apontar para.

Agora que já sabemos o comportamento dos controles de fluxo de seleção, você deve estar ansioso para verificar sua aplicação na resolução de problemas computacionais. Mas, antes de começarmos vamos resumir:

- Controle de fluxo de seleção possibilitam a construção de algoritmos que, dependendo do dado de entrada, algumas linhas não são processadas.
- Vimos que podemos construir controle de fluxo de seleção a partir das estruturas SE-SENÃO, SE-ENTÃO e SE-SENÃOSE.
- A estrutura SE-SENÃO possui o comportamento bifurcatório, ou seja, dependo do resultado de uma expressão lógica, um processamento será realizado se a expressão for verdadeira e outro poderá ser realizado se a expressão for falsa.
- A estrutura SE-ENTÃO possui o comportamento de desvio de fluxo, ou seja, dependendo do resultado de uma expressão lógica, um processamento será realizado se, e somente se, a expressão for verdadeira.
- A estrutura SE-SENÃOSE possui o comportamento de múltiplas condições subordinadas, ou seja, a próxima condição só é verificada se sua predecessora for falsa.
- Em todas as estruturas de seleção é recomendado o uso da tabulação para representar a subordinação.
- Na estrutura SE-SENÃO, o processamento que será realizado quando a expressão for verdadeira deve ficar entre as instruções ENTÃO e SENÃO. Já o processamento que será realizado quando a expressão for falsa deve ficar entre as instruções SENÃO e FIM-SE.

- Na estrutura SE-ENTÃO, o processamento que será realizado quando a expressão for verdadeira deve ficar entre as instruções ENTÃO e FIM-SE.

- Na estrutura SE-SENÃOSE, com exceção do último bloco de processamento, todos os demais que serão executados quando a expressão for verdadeira devem ficar entre as instruções ENTÃO e SENÃOSE

- Na estrutura SE-SENÃOSE, o último bloco de processamento que será executado quando a instrução for verdadeira deve ficar entre as instruções ENTÃO e FIM-SE.

- Na estrutura SE-SENÃOSE, se existir, a estrutura SE-NÃO deve ser a última da cadeia de condições.

- Na estrutura SE-SENÃOSE, se a instrução SENÃO estiver presente, significa que o seu bloco de código subordinado só será executado se todas as condições predecessoras forem falsas.

6.4 Questão 1: Receita de Bolo

Esta questão foi adaptada da Olimpíada Brasileira de Informática.

Conceição deseja fazer bolos para seus amigos, usando uma receita que indica que devem ser usadas 2 xícaras de farinha de trigo, 3 ovos e 5 colheres de sopa de leite. Em casa ela tem "A" xícaras de farinha de trigo, "B" ovos e "C" colheres de sopa de leite. Conceição não tem muita prática com a cozinha e, portanto, ele só se arriscará a fazer medidas exatas da receita de bolo (por exemplo, se ele tiver material suficiente para fazer mais do que 2 e menos do que 3 bolos, ele fará somente 2 bolos). Sabendo disto, ajude Conceição escrevendo um algoritmo que

determine qual a quantidade máxima de bolos que ela consegue fazer.

Entrada

A entrada é dada em uma única linha, que contém três números inteiros A, B e C, indicando respectivamente o número de xícaras de farinha de trigo, o número de ovos e o número de colheres de sopa de leite que Conceição tem em casa.

Saída

Seu programa deve imprimir uma única linha, contendo um único inteiro, a quantidade máxima de bolos que Conceição consegue fazer.

Exemplo

A	B	C	Valor esperado
10	15	20	4
100	100	4	0
9	9	9	1

Pensando

Sabendo que para fazer um bolo a Conceição utiliza 2 xícaras de farinha de trigo, 3 ovos e 5 colheres de sopa de leite, você deve, inicialmente, dividir a quantidade de cada ingrediente

que a Conceição possui pelas medidas indicadas na receita. Figura **??** ilustra este raciocínio.

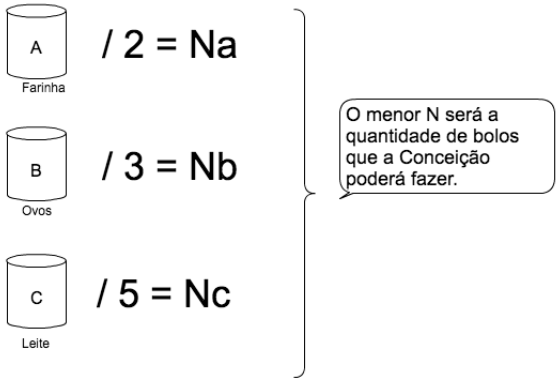


Figura 6.10: Ingredientes do bolo

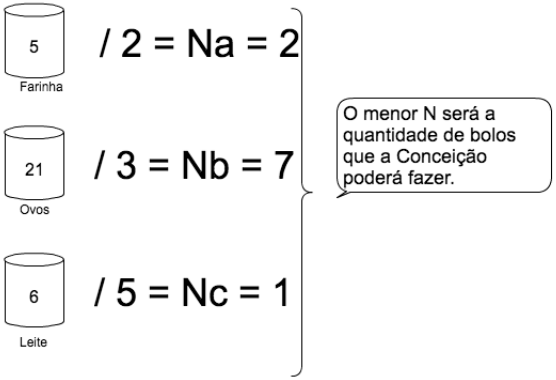


Figura 6.11: Ingredientes do bolo

A partir deste esquema você deve perceber que a quantidade de bolos que conceição conseguirá fazer será dependente da menor proporção de ingredientes que ela possui. Por exemplo, se Conceição tiver 5 xícaras de farinha de trigo (A=5);

21 ovos (B=21) e 6 colheres de leite (C=6) conseguirá fazer apenas um bolo.

Apesar dos outros ingredientes permitirem que Conceição faça mais de um bolo, o leite, neste exemplo, é o delimitador.

Em resumo, dado os valores de A, B e C, devemos encontrar a proporção dos ingredientes e a menor proporção será igual a quantidade de bolos que Conceição conseguirá fazer.

Planejando

Agora que você já sabe como resolver o problema, devemos planejar o algoritmo. Ao planejar o algoritmo lembre-se que este deve ser estruturado em entrada, processamento e saída.

A entrada que conceição deve informar são respectivamente a quantidade de xícaras de farinha de trigo, de ovos e de colheres de leite que ela possui.

Na etapa de processamento iremos dividir em dois momentos. No primeiro momento iremos realizar o processamento aritmético para encontrar a proporção dos ingredientes em relação às medidas da receita. No segundo momento iremos realizar o processamento lógico para encontrar qual foi a menor proporção calculada.

A saída deste problema será determinada justamente pelo processamento lógico que será realizado.

Veja o algoritmo e o fluxograma resultante:

```
1   INÍCIO
2       LER(A, B, C)
3       Na = A div 2
4       Nb = B div 3
5       Nc = C div 5
6       MENOR = Na
7
8       SE Nb < MENOR ENTÃO
9           MENOR = Nb
10      FIM-SE
11
12      SE Nc < MENOR ENTÃO
13          MENOR = Nc
14      FIMSE-SE
15
16      ESCREVER(MENOR)
17  FIM
```

Figura 6.12: Algoritmo receita de bolo

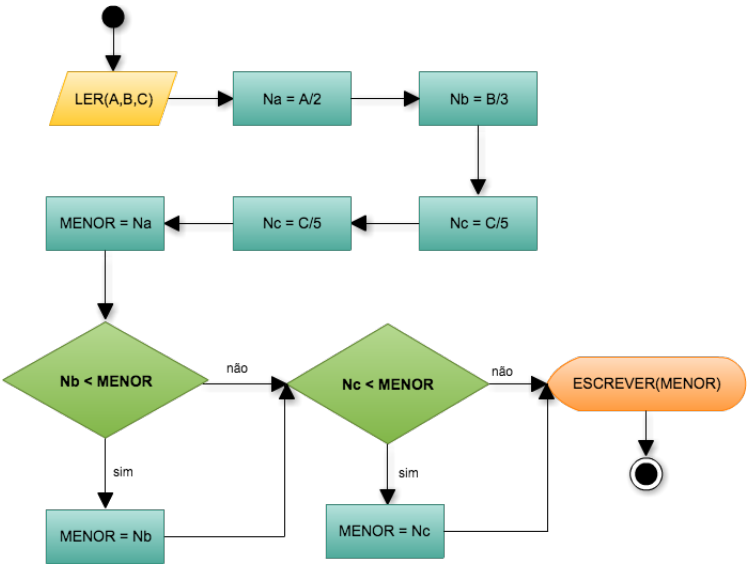


Figura 6.13: Fluxograma receita de bolo

Vamos praticar a leitura deste algoritmo:

- **LER(A,B,C):** Leia do dispositivo de entrada a quanti-

dade de xícaras de farinha de trigo; de ovos; de colheres de leite e armazene respectivamente nos endereços A, B e C.

- **Na = A div 2:** Recupere o valor do endereço A, divida por dois e armazene a parte inteira da divisão no endereço Na.
- **Nb = B div 3:** Recupere o valor do endereço B, divida por três e armazene a parte inteira da divisão no endereço Nb.
- **Nc = C div 5:** Recupere o valor do endereço C, divida por cinco e armazene a parte inteira da divisão no endereço Nc.
- **MENOR = Na:** Recupero o valor do endereço Na e armazene no endereço MENOR.
- **SE Nb < MENOR ENTÃO:** Recupere os valores dos endereços Nb e MENOR e verifique se o valor do endereço Nb é menor do que o valor do endereço MENOR.
- **MENOR = Nb:** Se for, então recupere o valor do endereço Nb e armazene no endereço MENOR.
- **FIM-SE:** Este comendo indica o fim do desvio de fluxo.
- **SE Nc < MENOR ENTÃO:** Recupere os valores dos endereços Nc e MENOR e verifique se o valor do endereço Nc é menor do que o valor do endereço MENOR.
- **MENOR = Nc:** Se for, então recupere o valor do endereço Nc e armazene no endereço MENOR.
- **FIM-SE:** Este comendo indica o fim do desvio de fluxo.
- **ESCREVER(MENOR):** Recupere o valor do endereço menor e escreva no dispositivo de saída.

Codificando

Para codificar este algoritmo na linguagem Python precisamos saber como representar o controle de fluxo SE-ENTÃO.

Anteriormente foi mencionado que a tabulação das linhas entre as instruções ENTÃO e FIM-SE eram recomendadas para demonstrar a subordinação dessas linhas em relação a verdade da expressão lógica. Por exemplo, na linha nove do nosso algoritmo a instrução MENOR = Nb está tabulada e subordinada à expressão Nb < MENOR. No caso da linguagem Python esta tabulação é OBRIGATÓRIA. Toda vez que você utilizar uma estrutura de controle de fluxo, as instruções subordinadas a essa estrutura devem estar tabuladas. A primeira linha não tabulada indica o fim da subordinação.

Em Python o comando if EXP_L: representa o controle de fluxo SE-ENTÃO. Veja o código:

```
1   A, B,  C = map(int, input().split())
2
3   Na = A/2
4   Nb = B/3
5   Nc = C/5
6
7   MENOR = Na
8
9   if Nb < MENOR:
10      MENOR = Nb
11
12  if Nc < MENOR:
13      MENOR = Nc
14
15  print(MENOR)
```

Figura 6.14: Codificação da receita de bolo

Observe que no Python não precisamos representar a instrução FIM-SE uma vez que o fim da subordinação é determinado pelo fim da tabulação.

Observe também que o dois pontos é obrigatório após a ex-

pressão lógica. Ele equivale à instrução ENTÃO do algoritmo que determina o início do desvio de fluxo.

Testando

Você pode utilizar os casos de teste apresentados no enunciado da questão para testar o seu programa. Também é recomendado que você produza outros casos de teste e verifique se o resultado será conforme o esperado.

6.5 Questão 2: Detectando Colisões

Esta questão foi adaptada da Olimpíada Brasileira de Informática.

Detecção de colisão é uma das operações mais comuns (e importantes) em jogos eletrônicos. O objetivo, basicamente, é verificar se dois objetos quaisquer colidiram, ou seja, se a interseção entre eles é diferente de vazio. Isso pode ser usado para saber se duas naves colidiram, se um monstro bateu numa parede, se um personagem pegou um item, etc. Para facilitar as coisas, muitas vezes os objetos são aproximados por figuras geométricas simples (esferas, paralelepípedos, triângulos etc). Neste problema, os objetos são aproximados por retângulos num plano 2D.

Sua tarefa é escrever um algoritmo que, dados dois retângulos, determine se eles se colidem ou não.

Entrada

A entrada contém duas linhas com quatro números inteiros cada. Na primeira linha os inteiros x1, y1 e x2, y2 representam

respectivamente os cantos inferior esquerdo e superior direito do primeiro retângulo. Na segunda linha os inteiros x3, y3 e x4, y4 representam respectivamente os cantos inferior esquerdo e superior direito do segundo retângulo.

Saída

A saída deve exibir o número 0 (zero) caso não haja colisão ou o número 1 (um) caso haja.

Exemplo

valores de entrada	valor esperado
0 0 2 2 1 1 3 3	1
0 0 1 1 2 2 3 3	0
1 2 3 4 3 4 1 2	1

Pensando

O objetivo deste problema é identificar quando há ou quando não há colisão entre dois retângulos. A colisão entre retângulos pode ser representada conforme a figura **??**.

Para resolver este problema nós temos duas possibilidades. A primeira consiste em construir uma expressão lógica para detectar colisão. A segunda possibilidade refere-se à construção da expressão lógica para detectar quando não há colisão.

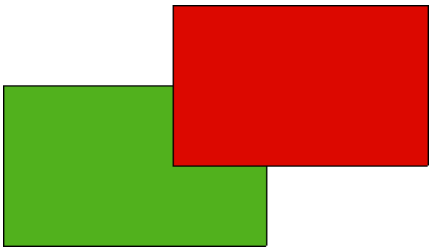


Figura 6.15: Exemplo de colisão.

Neste caso iremos solucionar este problema construindo uma expressão lógica para detectar quando não há colisão, ou seja, toda vez que a expressão for verdadeira é porque não existe colisão entre os dois retângulos, caso contrário, se a expressão for falsa é porque os retângulos se colidem.

A figura **??** ilustra as quatro possibilidades para que não haja colisão entre os retângulos.

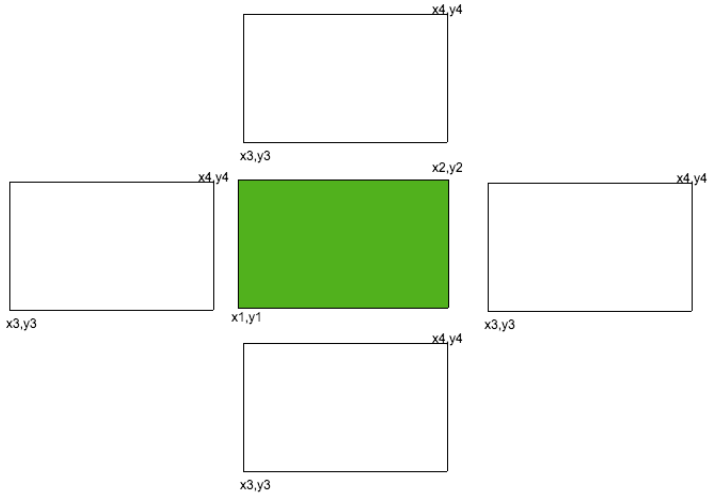


Figura 6.16: Exemplo de colisão.

Observe que para não haver colisão o retângulo de coordenadas (x3,y3 e x4,y4) deve estar ou a esquerda ou a direita

ou a cima ou a baixo do retângulo de coordenadas (x1,y1 e x2,y2). Sendo assim, para detectar a não existência de colisão basta identificar estas quatro situações.

Para facilitar vamos chamar o retângulo de coordenadas (x1,y1 e x2,y2) de retângulo (A) e o outro de retângulo (B).

Para detectar se o retângulo B está a direta ou a esquerda do retângulo A nós temos que olhar apenas para o eixo X. Já para detectar se o retângulo B está a cima ou a baixa do retângulo A nós temos que olhar apenas para o eixo Y.

Sendo assim as quatros regras são:

- **A direita:** se x3 $>$ x2.
- **A esquerda:** se x1 $>$ x4.
- **A cima:** se y3 $>$ y2.
- **A baixo:** se y1 $>$ y4.

Sabendo que basta apenas uma dessas condições serem verdadeiras para que não haja colisão, então, a expressão lógica que detecta a inexistência de colisão é: $x3 > x2$ OU $x1 > x4$ OU $y3 > y2$ OU $y1 > y4$

Planejando

Agora que você já sabe como resolver o problema, devemos planejar o algoritmo. Ao planejar o algoritmo lembre-se que este deve ser estruturado em entrada, processamento e saída.

Veja o algoritmo resultante:

```
1   INÍCIO
2       LER(x1, y1, x2, y2)
3       LER(x3, y3, x4, y4)
4
5       SE x3 > x2 OU x1 > x4 OU y3 > y2 OU y1 > y4 ENTÃO
6           ESCREVER(0)
7       SENÃO
8           ESCREVER(1)
9       FIM-SE
10  FIM
```

Figura 6.17: Algoritmo de colisão.

Para exercitar recomendo que você construa o fluxograma deste algoritmo!

Vamos praticar a leitura deste algoritmo:

- **LER(x1, y1, x2, y2):** Leia as coordenadas inferior direita e superior esquerda do retângulo A e armazene respectivamente em x1,y1 e x2,y2.

- **LER(x3, y3, x4, y4):** Leia as coordenadas inferior direita e superior esquerda do retângulo B e armazene respectivamente em x3,y3 e x4,y4.

- **SE x3>x2 OU x1>x4 OU y3>y2 OU y1>y4 ENTÃO:** Verifique se os retângulos não se colidem.

- **ESCREVER(0):** Se não se colidirem, então escreva o valor zero no dispositivo de saída.

- **SENÃO:** Caso contrário.

- **ESCREVER(1):** escreva o valor um no dispositivo de saída.

- **FIM-SE:** Fim da bifurcação.

Codificando

Para codificar este algoritmo na linguagem Python precisamos saber como representar o controle de fluxo SE-SENÃO.

Vale ressaltar que na linguagem Python, toda vez que precisamos utilizar alguma estrutura de controle de fluxo, a tabulação é OBRIGATÓRIA.

Na linguagem Python o controle de fluxo SE-SENÃO é representado pelos comandos **if else** e o operador lógico OU pelo comando **or**.

Veja o código resultante:

```
1  x1, y1, x2, y2 = map(int, input().split())
2  x3, y3, x4, y4 = map(int, input().split())
3
4  if x2 < x3 or x4 < x1 or y1 > y4 or y3 > y2:
5      print("0")
6  else:
7      print("1")
```

Figura 6.18: Codificação do algoritmo de colisão.

Testando

Você pode utilizar os casos de teste apresentados no enunciado da questão para testar o seu programa. Também é recomendado que você produza outros casos de teste e verifique se o resultado será conforme o esperado.

6.6 Questão 3: Alarme Despertador

Esta questão foi adaptada da primeira fase da Maratona de Programação promovida pela SBC em 2009.

Daniela é enfermeira em um grande hospital, e tem os horários de trabalho muito variáveis. Para piorar, ela tem sono pesado, e uma grande dificuldade para acordar com relógios despertadores.

Recentemente ela ganhou de presente um relógio digital, com alarme com vários tons, e tem esperança que isso resolva o seu problema. No entanto, ela anda muito cansada e quer aproveitar cada momento de descanso. Por isso, carrega seu relógio digital despertador para todos os lugares, e sempre que tem um tempo de descanso procura dormir, programando o alarme despertador para a hora em que tem que acordar. No entanto, com tanta ansiedade para dormir, acaba tendo dificuldades para adormecer e aproveitar o descanso.

Um problema que a tem atormentado na hora de dormir é saber quantos minutos ela teria de sono se adormecesse imediatamente e acordasse somente quando o despertador tocasse. Mas ela realmente não é muito boa com números, e pediu sua ajuda para escrever um programa que, dada a hora corrente e a hora do alarme, determine o número de minutos que ela poderia dormir.

Entrada

A entrada possui quatro números inteiros H1, M1, H2 e M2, com H1:M1 representando a hora e minuto atuais, e H2:M2 representando a hora e minuto para os quais o alarme despertador foi programado.

Saída

O programa deve informar o número de minutos que Daniela tem para dormir.

Exemplo

Entrada	Valor esperado
1 5 3 5	120
23 59 0 34	35
21 33 21 10	1417

Pensando

Considerando que Daniela está interessada em saber quantos minutos ela pode dormir, então, para resolver este problema, temos que inicialmente transformar as horas para minutos.

Sendo assim, podemos considerar A = H1*60+M1 como o momento que Daniela pretende dormir e B = H2*60+M2 como o momento em que o despertador irá tocar.

Após a conversão das horas para minutos podemos equivocadamente pensar que o número de minutos que Daniela dormiu se dá apenas pela diferença de B por A.

Observe que para alguns exemplos isso é verdade.

- Considerando que Daniela dormiu às 22:30 e acordou às 23:00 podemos claramente perceber que ela teve 30 minutos de sono pois A=1350, B=1380 e B-A=30.
- Considerando que Daniela dormiu às 23:00 e acordou às 04:00 podemos claramente perceber que ela teve 5 horas de sono o que equivale a 300 minutos. Porém observe que se fizermos a diferença de B-A o resultado mostra-se incoerente: A=1380, B=240 e B-A=-1140.

A grande sacada deste problema é verificar se Daniela dorme e acorda no mesmo dia ou se ela dorme e acorda somente no

dia seguinte. Este fato irá influenciar diretamente no nosso cálculo.

Ela acordará no outro dia se A > B, ou seja, se o valor da hora inicial for maior que o valor da hora final. Neste caso, deve-se acrescentar 1440 minutos ao valor de B e o número de minutos que Daniela dormiu é calculado pela diferença de B por A.

Caso contrário, se Daniela dorme e acorda no mesmo dia, o número de minutos que Daniela dormiu é calculado apenas pela diferença de B por A.

Planejando

Agora que você já sabe como resolver o problema, devemos planejar o algoritmo. Ao planejar o algoritmo lembre-se que este deve ser estruturado em entrada, processamento e saída.

Veja o algoritmo e o fluxograma resultante:

```
1   INÍCIO
2       LER(H1, M1, H2, M2)
3       A = H1*60+M1
4       B = H2*60+M2
5
6       SE A > B ENTÃO
7           C = B+1440-A
8       SENÃO
9           C = B-A
10      FIM-SE
11
12      ESCREVER(C)
13  FIM
```

Figura 6.19: Algoritmo alarme despertador.

Vamos praticar a leitura deste algoritmo:

- **LER(H1, M1, H2, M2):** Leia a hora e o minuto que Daniela dormiu, a hora e o minuto que o alarme tocou e

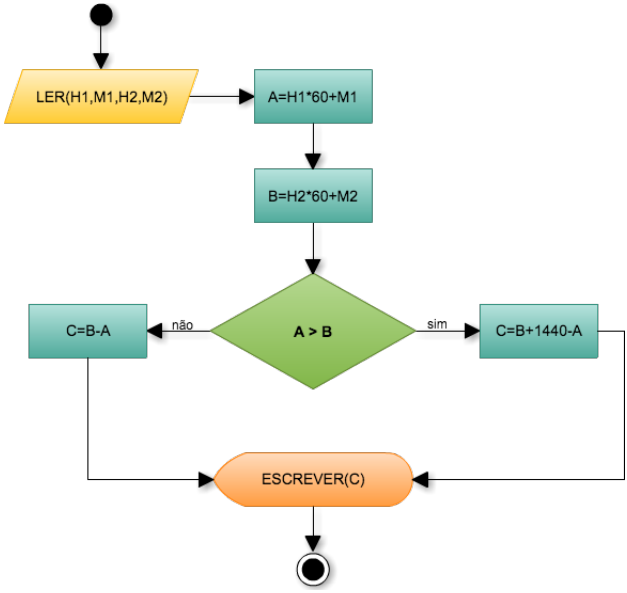


Figura 6.20: Fluxograma alarme despertador.

armazene respectivamente em H1, M1, H2 e M2.

- **A=H1*60+M1:** Recupere a hora e o minuto que Daniela dormiu, transforme tudo para minutos e armazene no endereço A.
- **B=H2*60+M2:** Recupere a hora e o minuto que o alarme tocou, transforme tudo para minutos e armazene no endereço B.
- **SE A > B ENTÃO:** Verifique se Daniela acordou no outro dia.
- **C=B+1440-A:** Se acordou, então recupera o valor dos endereços B e A, some 1440 ao valor de B, subtraia B por A e armazene o resultado no endereço C.
- **SENÃO:** Senão acordou no outro dia.

- **C=B-A:** Recupere o valor dos endereços B e A, subtraia B por A e armazene o resultado no endereço C.
- **FIM-SE:** Fim da bifurcação.
- **ESCREVER(C):** Recupere o valor de C e escreva no dispositivo de saída.

Codificando

Para codificar este algoritmo temos que utilizar o controle de fluxo if else. Não custa lembrar que na linguagem Python a tabulação é OBRIGATÓRIA em estruturas de controle de fluxo.

Veja o código resultante:

```
1  h1, m1, h2, m2 = map(int, input().split())
2
3  a = h1*60+m1
4  b = h2*60+m2
5
6  if a > b:
7      c = b+1440-a
8  else:
9      c = b-a
10
11 print(c)
```

Figura 6.21: Codificação alarme despertador.

Testando

Você pode utilizar os casos de teste apresentados no enunciado da questão para testar o seu programa. Também é recomendado que você produza outros casos de teste e verifique se o resultado será conforme o esperado.

6.7 Exercício

Neste capítulo as questões estão em ordem de dificuldade crescente. Recomendo que você resolva cada questão utilizando a metodologia PPCT conforme apresentada neste capítulo.

Questão 1: Aviões de papel

Esta questão foi adaptada da Olimpíada Brasileira de Informática.

Para descontrair os alunos após as provas da OBI, a Diretora da escola organizou um campeonato de aviões de papel. Cada aluno participante receberá uma certa quantidade de folhas de um papel especial para fazer os seus modelos de aviões. A quantidade de folhas que cada aluno deverá receber ainda não foi determinada: ela será decidida pelos juízes do campeonato.

A diretora convidou, para atuarem como juízes, engenheiros da Embraer, uma das mais empresas brasileiras, que vende aviões com tecnologia brasileira no mundo todo. O campeonato está programado para começar logo após a prova da OBI, mas os juízes ainda não chegaram à escola. A diretora está aflita, pois comprou uma boa quantidade de folhas de papel especial, mas não sabe se a quantidade comprada vai ser suficiente.

Considere, por exemplo, que a Diretora comprou 100 folhas de papel especial, e que há 33 competidores. Se os juízes decidirem que cada competidor tem direito a três folhas de papel, a quantidade comprada pela diretora é suficiente. Mas se os juízes decidirem que cada competidor tem direito a quatro folhas, a quantidade comprada pela diretora não seria suficiente.

Você foi contratado pela Diretora para escrever um pro-

grama que, dados o número de competidores, o número de folhas de papel especial compradas pela Diretora e o número de folhas que cada competidor deve receber, determine se o número de folhas comprado pela Diretora é suficiente.

Entrada

A entrada consiste de três inteiros C, P e F representando respectivamente o número de competidores, a quantidade de folhas de papel especial compradas pela Diretora e a quantidade de folhas de papel especial que cada competidor deve receber.

Saída

Seu programa deve imprimir, na saída padrão, o caractere 'S' se a quantidade de folhas compradas pela Diretora é suficiente, ou o caractere 'N' caso contrário. Note que os caracteres devem ser letras maiúsculas.

Exemplo

Valores de entrada	Valor esperado
10 100 10	S
10 90 10	N
5 40 2	S

Questão 2: Gangorra

Joãozinho acaba de mudar de escola e a primeira coisa que percebeu na nova escola é que a gangorra do parquinho não é simétrica, uma das extremidades é mais longa que a outra. Após brincar algumas vezes com um amigo de mesmo peso, ele percebeu que quando está em uma extremidade, a gangorra se desequilibra para o lado dele (ou seja, ele fica na parte de baixo, e o amigo na parte de cima), mas quando eles trocam de lado, a gangorra se desequilibra para o lado do amigo. Sem entender a situação, Joãozinho pediu ajuda a outro amigo de outra série, que explicou que o comprimento do lado interfere no equilíbrio da gangorra, pois a gangorra estará equilibrada quando P1 * C1 for igual a P2 * C2.

P1 e P2 são os pesos da criança no lado esquerdo e direito, respectivamente, e C1 e C2 são os comprimentos da gangorra do lado esquerdo e direito, respectivamente.

Com a equação, Joãozinho já consegue dizer se a gangorra está equilibrada ou não mas, além disso, ele quer saber para qual lado a gangorra descerá caso esteja desequilibrada.

Entrada

A entrada contém 4 inteiros, P1, C1, P2 e C2 , nesta ordem.

Saída

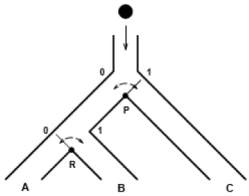
Se a gangorra estiver equilibrada, imprima ‘0’. Se ela estiver desequilibrada de modo que a criança esquerda esteja na parte de baixo, imprima ‘-1’, senão, imprima ‘1’.

Exemplo

Valores de entrada	Valor esperado
30 100 60 50	0
40 40 38 60	1
35 80 35 75	-1

Questão 3: Flíper

Flíper é um tipo de jogo onde uma bolinha de metal cai por um labirinto de caminhos até chegar na parte de baixo do labirinto. A quantidade de pontos que o jogador ganha depende do caminho que a bolinha seguir. O jogador pode controlar o percurso da bolinha mudando a posição de algumas portinhas do labirinto. Cada portinha pode estar na posição 0, que significa virada para a esquerda, ou na posição 1 que quer dizer virada para a direita. Considere o flíper da figura abaixo, que tem duas portinhas. A portinha P está na posição 1 e a portinha R, na posição 0. Desse jeito, a bolinha vai cair pelo caminho B.



Você deve escrever um programa que, dadas as posições das portinhas P e R, neste flíper da figura, diga por qual dos três caminhos, A, B ou C, a bolinha vai cair!

Entrada

A entrada é composta por apenas uma linha contendo dois números P e R, indicando as posições das duas portinhas do flíper da figura.

Saída

A saída do seu programa deve ser também apenas uma linha, contendo uma letra maiúscula que indica o caminho por onde a bolinha vai cair: 'A', 'B' ou 'C'.

Exemplo

Valores de entrada	Valor esperado
1 0	B
0 0	C
1 1	A

Questão 4: Frota de Táxi

A Companhia de Táxi Tabajara (CTT) é uma das maiores empresas de transporte do país. Possui uma vasta frota de carros e opera em todas as grandes cidades. Recentemente a CTT modernizou a sua frota, adquirindo um lote de 500 carros bi-combustíveis (carros que podem utilizar como combustível tanto álcool quanto gasolina). Além do maior conforto para os passageiros e o menor gasto com manutenção, com os novos carros é possível uma redução adicional de custo: como o preço da gasolina está sujeito a variações muito bruscas e pode ser

vantagem, em certos momentos, utilizar álcool como combustível. Entretanto, os carros possuem um melhor desempenho utilizando gasolina, ou seja, em geral, um carro percorre mais quilômetros por litro de gasolina do que por litro de álcool.

Você foi contratado pela CTT para escrever um programa que, dados o preço do litro de álcool, o preço do litro de gasolina e os quilômetros por litro que um carro bi-combustível realiza com cada um desses combustíveis, determine se é mais econômico abastecer os carros da CTT com álcool ou com gasolina. No caso de não haver diferença de custo entre abastecer com álcool ou gasolina a CTT prefere utilizar gasolina.

Entrada

A entrada é composta por quatro números reais com precisão de duas casas decimais A, G, Ra e Rg, representando respectivamente o preço por litro do álcool, o preço por litro da gasolina, o rendimento (km/l) do carro utilizando álcool e o rendimento (km/l) do carro utilizando gasolina.

Saída

A saída deve ser composta por uma única linha contendo o caractere 'A' se é mais econômico abastecer a frota com álcool ou o caractere 'G' se é mais econômico ou indiferente abastecer a frota com gasolina.

Exemplo

Valores de entrada	Valor esperado
1.20 2.30 10.00 15.00	A
1.00 1.00 9.00 9.01	G
1.00 1.00 11.00 11.00	G

Questão 5: Campeonato

Dois times, Cormengo e Flaminthians, participam de um campeonato de futebol, juntamente com outros times. Cada vitória conta três pontos, cada empate um ponto. Fica melhor classificado no campeonato um time que tenha mais pontos. Em caso de empate no número de pontos, fica melhor classificado o time que tiver maior saldo de gols. Se o número de pontos e o saldo de gols forem os mesmos para os dois times então os dois times estão empatados no campeonato. Dados os números de vitórias e empates, e os saldos de gols dos dois times, sua tarefa é determinar qual dos dois está melhor classificado, ou se eles estão empatados no campeonato.

Entrada

A entrada é descrita em uma única linha, que contém seis inteiros, separados por um espaço em branco: Cv, Ce, Cs, Fv, Fe e Fs, que são, respectivamente, o número de vitórias do Cormengo, o número de empates do Cormengo, o saldo de gols do Cormengo, o número de vitórias do Flaminthians, o número de empates do Flaminthians e o saldo de gols do Flaminthians.

Saída

Seu programa deve imprimir uma única linha. Se Cormengo é melhor classificado que Flaminthians, a linha deve conter apenas a letra 'C', se Flaminthians é melhor classificado que Cormengo, a linha deve conter apenas a letra 'F', e se os dois times estão empatados a linha deve conter apenas o caractere '='.

Exemplo

Valores de entrada	Valor esperado
10 5 18 11 1 18	C
10 5 18 11 2 18	=
9 5 -1 10 2 10	F

Questão 6: Nota da prova

Rosy é uma talentosa professora do Ensino Médio que já ganhou muitos prêmios pela qualidade de sua aula. Seu reconhecimento foi tamanho que foi convidada a dar aulas em uma escola da Inglaterra. Mesmo falando bem o inglês, Rosy ficou um pouco apreensiva com a responsabilidade, mas resolveu aceitar a proposta e encará-la como um bom desafio.

Tudo ocorreu bem para Rosy até o dia da prova. Acostumada a dar notas de 0 (zero) a 100 (cem), ela fez o mesmo na primeira prova dos alunos da Inglaterra. No entanto, os alunos acharam estranho, pois na Inglaterra o sistema de notas é diferente: as notas devem ser dadas como conceitos de A à

E. O conceito A é o mais alto, enquanto o conceito E é o mais baixo.

Conversando com outros professores, ela recebeu a sugestão de utilizar a seguinte tabela, relacionando as notas numéricas com as notas de conceitos:

Nota	Conceito
0	E
1 a 35	D
36 a 60	C
61 a 85	B
86 a 100	A

O problema é que Rosy já deu as notas no sistema numérico, e terá que converter as notas para o sistema de letras. Porém, Rosy precisa preparar as próximas aulas (para manter a qualidade que a tornou reconhecida), e não tem tempo suficiente para fazer a conversão das notas manualmente. Rosy pediu a sua ajuda para escrever um programa que recebe uma nota no sistema numérico e determina o conceito correspondente.

Entrada

A entrada contém uma única linha com um número inteiro N ($0 \leq N \leq 100$), representando uma nota de prova no sistema numérico.

Saída

Seu programa deve imprimir, na saída padrão, uma letra (A, B, C, D, ou E em maiúsculas) representando o conceito correspondente à nota dada na entrada.

Exemplo

Valor de entrada	Valor esperado
12	D
87	A
0	E

Questão 7: Cartas

Beatriz gosta muito de jogar cartas com as amigas. Para treinar memória e raciocínio lógico, ela inventou um pequeno passatempo com cartas. Ela retira as cinco primeiras cartas do topo de um baralho bem embaralhado, e as coloca em sequência, da esquerda para a direita, na mesa, com as faces voltadas para baixo.

Então ela olha, por um breve instante, cada uma das cartas da sequência (e logo as recoloca na mesa, com a face para baixo). Usando apenas a sua memória, Beatriz deve agora dizer se a sequência de cartas está ordenada crescentemente, decrescentemente, ou não está ordenada.

De tanto jogar, ela está ficando cansada, e não confia em seu próprio julgamento para saber se acertou ou errou. Por isso, ela pediu para você fazer um programa que, dada uma sequência de cinco cartas, determine se a sequência dada está ordenada crescentemente, decrescentemente, ou não está ordenada.

Entrada

A entrada consiste de uma única linha que contém as cinco cartas da sequência. Os valores das cartas são representados por inteiros entre 1 e 13. As cinco cartas têm valores distintos.

Saída

Seu programa deve produzir uma única linha, contendo um único caractere maiúsculo: 'C' caso a sequência dada esteja ordenada crescentemente, 'D' se estiver ordenada decrescentemente, ou 'N' caso contrário.

Exemplo

Valores de entrada	Valor esperado
1 2 3 4 5	C
5 7 10 9 11	N
12 10 4 3 2	D

Questão 8: Corrida

A Federação de Corridas de Charrete (FCC) organiza todo ano a Subida Brigite Cardoso (SBC), disputada nas ladeiras de paralelepípedo de Ouro Preto. A corrida é uma das mais tradicionais do esporte, completando 100 anos em 2013. Para comemorar o centenário, a FCC pretende integrar dispositivos GPS às charretes, permitindo aos espectadores desfrutarem de dados de telemetria em tempo real.

No mesmo viés de inovação tecnológica, a FCC transmitirá a SBC via satélite para todo o planeta, e quer integrar a telemetria na transmissão, indicando qual seria o vencedor da corrida se as charretes mantivessem suas velocidades até o final da corrida; ela pediu que você escrevesse um programa que, dados as distâncias até a linha de chegada, as velocidades e os números das duas charretes que lideram a corrida, determina quem seria o vencedor da corrida (você pode supor que as charretes não cruzam a linha de chegada simultaneamente).

Entrada

A entrada consiste de duas linhas; cada linha descreve uma das charretes que lidera a corrida. A descrição de uma charrete consiste de três inteiros N, D e V indicando, respectivamente, o número da charrete, a sua distância à linha de chegada em metros, e a sua velocidade, em quilômetros por hora. Os números das duas charretes são distintos.

Saída

Imprima uma única linha, contendo um único número inteiro, indicando o número da charrete que seria vencedora, conforme descrito acima.

Exemplo

Valores de entrada	Valor esperado
45 900 40 17 300 20	17
1 1000 100 2 1000 99	1
6 1000 40 3 500 15	6

Questão 9: Sedex

A Copa do Mundo de 2010 foi realizada na África do Sul. Bolas de futebol são muito fáceis de transportar, já que elas saem das fábricas vazias e só são enchidas somente pelas lojas ou pelos consumidores finais. Infelizmente o mesmo não pode ser dito das bolas de boliche. Como elas são completamente sólidas, elas só podem ser transportadas embaladas uma a uma, em caixas separadas.

A SBC - Só Boliche Cascavel - é uma fábrica de bolas de boliche que trabalha somente através de encomendas e envia todas as bolas por SEDEX. Como as bolas têm tamanhos diferentes, a SBC tem vários tamanhos de caixas diferentes para transportá-las.

Você contratado para escrever um programa que, dado o diâmetro de uma bola e as 3 dimensões de uma caixa (altura, largura e profundidade), diz se a bola de boliche cabe dentro da caixa ou não.

Entrada

A primeira linha da entrada contém um inteiro N que indica o diâmetro da bola de boliche. A segunda linha da entrada contém 3 números inteiros separados por um espaço cada: a altura A seguida da largura L e da profundidade P da caixa.

Saída

Seu programa deve imprimir uma única linha, contendo a letra 'S' caso a bola de boliche caiba dentro da caixa ou 'N' caso contrário.

Exemplo

Valores de entrada	Valor esperado
3 3 2 5	N
5 5 5 5	S
4 5 4 6	S

Questão 10: Tira teima

Uma quadra de tênis tem o formato de um retângulo, cujos lados medem 36 pés por 78 pés, que correspondem a um retângulo de 432 polegadas por 936 polegadas. No último Grand Slam da Austrália, Rafael Nadal perdeu para Novak Djokovic, num dos jogos mais bonitos de tênis dos últimos tempos. Muitas vezes, uma jogada é tão rápida, e a bola tão próxima

da borda da quadra, que o juiz pode tomar uma decisão que pode ser contestada por um dos jogadores. Para isso, existe o tira-teima, que utiliza a imagem gravada do jogo para decidir se a bola estava dentro ou fora da metade da quadra correspondente a um dos jogadores. Considere que a semi-quadra de Rafael Nadal corresponde a um retângulo em que dois vértices têm coordenadas (0,0) e (432, 468), onde todos os números são em polegadas. Você deve escrever um programa para, dadas as coordenadas (X, Y) do ponto de contato da bola com o solo, determinar se uma bola bateu no solo dentro ou fora da semi-quadra. Note que se a bola bate na linha divisória ela é considerada uma bola dentro.

Entrada

A entrada é dada em uma única linha, que contém dois inteiros X e Y, que correspondem às coordenadas do ponto (X, Y) de contato da bola com o solo, em polegada.

Saída

Seu programa deve imprimir uma única linha, contendo a palavra dentro se a bola bateu dentro da semi-quadra, e a palavra fora caso contrário.

Exemplo

Valores de entrada	Valor esperado
-2 200	Fora
432 10	Dentro
100 100	Dentro

Questão 11: Meteoros

Em noites sem nuvens pode-se muitas vezes observar pontos brilhantes no céu que se deslocam com grande velocidade, e em poucos segundos desaparecem de vista: são as chamadas estrelas cadentes, ou meteoros. Meteoros são na verdade partículas de poeira de pequenas dimensões que, ao penetrar na atmosfera terrestre, queimam-se rapidamente (normalmente a uma altura entre 60 120 quilômetros). Se os meteoros são suficientemente grandes, podem não queimar-se completamente na atmosfera e dessa forma atingem a superfície terrestre: nesse caso são chamados de meteoritos.

Zé Felício é um fazendeiro que adora astronomia e descobriu um portal na Internet que fornece uma lista das posições onde caíram meteoritos. Com base nessa lista, e conhecendo a localização de sua fazenda, Zé Felício deseja saber se algum meteorito caiu dentro de sua propriedade. Ele precisa de sua ajuda para escrever um programa de computador que faça essa verificação automaticamente. As linhas que delimitam a fazenda são paralelas aos eixos cartesianos.

Entrada

Na primeira linha de um conjunto de testes quatro números inteiros X1, Y1, X2 e Y2, onde (X1, Y1) é a coordenada do canto superior esquerdo e (X2, Y2) é a coordenada do canto inferior direito do retângulo que delimita a fazenda. A segunda linha possui dois números inteiros X e Y, correspondendo às coordenadas do meteorito.

Saída

Seu programa de exibir a mensagem 'sim' caso o meteorito tenha caído na propriedade de Zé Felício e 'não' caso contrário.

Exemplo

Valores de entrada	Valor esperado
2 4 5 1 1 2	Não
2 4 5 1 3 3	Sim
2 4 3 2 2 3	Sim

Questão 12: Vice-campeão

A OBI (Organização de Bocha Internacional) é responsável por organizar a competição mundial de bocha. Infelizmente esse esporte não é muito popular, e numa tentativa de aumentar a sua popularidade, ficou decidido que seriam chamados, para a Grande Final Mundial, o campeão e o vice-campeão de cada sede nacional, ao invés de apenas o primeiro lugar.

Tumbólia é um país pequeno que já havia realizado a sua competição nacional quando a nova regra foi instituída, e o comitê local não armazenou quem foi o segundo classificado. Felizmente eles armazenaram a pontuação de todos competidores - que foram apenas três, devido ao tamanho diminuto do país. Sabe-se também que as pontuações de todos jogadores foram diferentes, de forma que não ocorreu empate entre nenhum deles.

Resta agora descobrir quem foi o vice-campeão e para isso o comitê precisa de ajuda.

Entrada

A primeira e única linha da entrada consiste de três inteiros separados por espaços, A, B e C que representam as pontuações dos 3 competidores.

Saída

Imprima uma única linha na saída, contendo apenas um número inteiro, a pontuação do vice-campeão.

Exemplo

Valores de entrada	Valor esperado
4 5 6	5
9 10 5	9
12 13 14	13

Questão 13: Guarda Costeira

"Pega ladrão! Pega ladrão!" Roubaram a bolsa de uma inocente senhora que caminhava na praia da Nlogônia e o ladrão fugiu em direção ao mar. Seu plano parece obvio: ele pretende pegar um barco e escapar!

O fugitivo, que a essa altura já está a bordo de sua embarcação de fuga, pretende seguir perpendicularmente à costa em direção ao limite de aguas internacionais, que fica a 12 milhas náuticas de distância, onde estará são e salvo das autoridades locais. Seu barco consegue percorrer essa distância a uma velocidade constante de VF nós.

A Guarda Costeira pretende interceptá-lo, e sua embarcação tem uma velocidade constante de VG nós. Supondo que ambas as embarcações partam da costa exatamente no mesmo instante, com uma distância de D milhas náuticas entre elas, será possível a Guarda Costeira alcançar o ladrão antes do limite de aguas internacionais?

Assuma que a costa da Nlogônia é perfeitamente retilínea e o mar bastante calmo, de forma a permitir uma trajetória tão retilínea quanto a costa.

Entrada

A entrada consiste em uma linha contendo três inteiros, D, VF e VG, indicando respectivamente a distância inicial entre o fugitivo e a Guarda Costeira, a velocidade da embarcação do fugitivo e a velocidade da embarcação da Guarda Costeira.

Saída

Imprima uma linha contendo 'S' se for possível que a Guarda Costeira alcance o fugitivo antes que ele ultrapasse o limite de aguas internacionais ou 'N' caso contrário.

Exemplo

Valores de entrada	Valor esperado
5 1 12	S
12 10 7	N
9 12 15	S

Questão 14: Conta de água

A empresa local de abastecimento de água, a Saneamento Básico da Cidade (SBC), está promovendo uma campanha de conservação de água, distribuindo cartilhas e promovendo ações demonstrando a importância da água para a vida e para o meio ambiente.

Para incentivar mais ainda a economia de água, a SBC alterou os preços de seu fornecimento de forma que, proporcionalmente, aqueles clientes que consumirem menos água paguem menos pelo metro cúbico. Todo cliente paga mensalmente uma assinatura de R$ 7, que inclui uma franquia de 10 m^3 de água. Isto é, para qualquer consumo entre 0 e 10 m^3, o consumidor paga a mesma quantia de R$ 7 reais (note que o valor da assinatura deve ser pago mesmo que o consumidor não tenha consumido água). Acima de 10 m^3 cada metro cúbico subsequente tem um valor diferente, dependendo da faixa

de consumo. A SBC cobra apenas por quantidades inteiras de metros cúbicos consumidos. A tabela abaixo especifica o preço por metro cúbico para cada faixa de consumo:

Faixa de consumo (m^3)	Preço (m^3)
até 10	incluído na franquia
11 a 30	R$ 1
31 a 100	R$ 2
101 em diante	R$ 5

Assim, por exemplo, se o consumo foi de 120 m^3, o valor da conta é:

- 7 reais da assinatura básica;
- 20 reais pelo consumo no intervalo 11 - 30 m^3;
- 140 reais pelo consumo no intervalo 31 - 100 m^3;
- 100 reais pelo consumo no intervalo 101 - 120 m^3.

Logo o valor total da conta de água é R$ 267.

Para automatizar o sistema de cobrança você foi contratado para escrever um programa que, dado o consumo de uma residência em m3, calcula o valor da conta de água daquela residência.

Entrada

A única linha da entrada contém um único inteiro N, indicando o consumo de água da residência, em m^3.

Saída

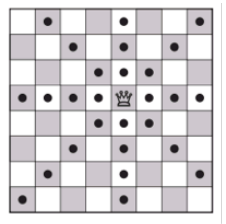
Seu programa deve imprimir uma única linha, contendo o valor da conta de água daquela residência.

Exemplo

Valor de entrada	Valor esperado
8	7
14	11
42	51

Questão 15: Dama

O jogo de xadrez possui várias peças com movimentos curiosos: uma delas é a dama, que pode se mover qualquer quantidade de casas na mesma linha, na mesma coluna, ou em uma das duas diagonais, conforme exemplifica a figura abaixo:



O grande mestre de xadrez Kary Gasparov inventou um novo tipo de problema de xadrez: dada a posição de uma dama em um tabuleiro de xadrez vazio (ou seja, um tabuleiro

8×8, com 64 casas), de quantos movimentos, no mínimo, ela precisa para chegar em outra casa do tabuleiro?

Kary achou a solução para alguns desses problemas, mas teve dificuldade com outros, e por isso pediu que você escrevesse um programa que resolve esse tipo de problema.

Entrada

A primeira e única linha de cada caso de teste contém quatro inteiros X1, Y1 , X2 e Y2. A dama começa na casa de coordenadas (X1 , Y1), e a casa de destino é a casa de coordenadas (X2 , Y2). No tabuleiro, as colunas são numeradas da esquerda para a direita de 1 a 8 e as linhas de cima para baixo também de 1 a 8. As coordenadas de uma casa na linha X e coluna Y são (X, Y).

Saída

Seu programa deve imprimir uma única linha na saída, contendo um número inteiro, indicando o menor número de movimentos necessários para a dama chegar em sua casa de destino.

Exemplo

Valores de entrada	Valor esperado
4 4 6 2	1
3 5 3 5	0
5 5 4 3	2

Questão 16: Triângulo

Ana e suas amigas estão fazendo um trabalho de geometria para o colégio, em que precisam formar vários triângulos, numa cartolina, com algumas varetas de comprimentos diferentes. Logo elas perceberam que não dá para formar triângulos com três varetas de comprimentos quaisquer. Se uma das varetas for muito grande em relação às outras duas, não dá para formar o triângulo. Ana fez uma pesquisa na internet e aprendeu que com três varetas é possível formar um triângulo quando, para todas as varetas, vale a seguinte relação: o comprimento da vareta é menor do que a soma dos comprimentos das outras duas varetas. Por exemplo, se os comprimentos forem 6, 9 e 5, vai dar para formar o triângulo, pois a relação vale para as três varetas: $6 < 9 + 5$, $9 < 6 + 5$ e $5 < 6 + 9$. Mas, se os comprimentos forem, por exemplo, 4, 10 e 3, não vai dar para formar um triângulo, porque a relação não vale para uma das varetas (pois 10 não é menor do que $3 + 4$).

Neste problema, você precisa ajudar Ana e suas amigas a descobrir se, dados os comprimentos de quatro varetas, é ou não é possível selecionar três varetas, dentre as quatro, e formar um triângulo!

Entrada

A entrada é composta por apenas uma linha contendo quatro números inteiros.

Saída

Seu programa deve produzir apenas uma linha contendo o caractere 'S', caso seja possível formar o triângulo; ou o caractere

'N', caso não seja possível formar o triângulo.

Exemplo

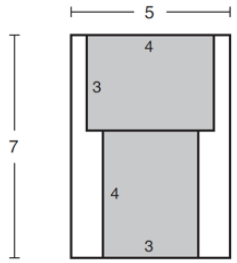
Valores de entrada	Valor esperado
6 9 22 5	S
14 40 12 60	N
4 10 3 1	N

Questão 17: Álbum de fotos

Clara está organizando as fotos da sua última viagem num álbum de fotos. Como ela tem muitas fotos, para economizar páginas do álbum ela quer colar duas fotos por página.

Como as fotos são retangulares, Clara decidiu que, para preservar o equilíbrio estético do álbum, as fotos podem ser coladas apenas verticalmente ou horizontalmente tal que elas devem sempre ficar inteiramente contidas no interior da página, e não devem se sobrepor.

Em geral, das muitas formas de posicionar as fotos do álbum só algumas (ou nenhuma) satisfazem estas restrições, então pode ser difícil decidir se é possível colar as duas fotos em uma mesma página do álbum, e por isso Clara pediu a sua ajuda para escrever um programa que, dadas as dimensões da página e das fotos, decide se é possível colar as fotos na página. Por exemplo, cada página pode ser 5×7, e duas fotos são 3×4. Nesse caso, é possível colar as duas fotos:



Entrada

A primeira linha da entrada contém dois inteiros X e Y, indicando a largura e a altura da página do álbum. Cada uma das duas linhas seguintes contém dois inteiros L e H, indicando a largura e a altura das fotos.

6.8 Saída

Imprima uma única linha, contendo um único caractere: 'S', se é possível colar as duas fotos na página do álbum, e 'N', caso contrário.

Exemplo

Valores de entrada	Valor esperado
7 5 3 4 3 4	S
10 10 6 6 6 6	N
13 8 4 9 6 5	N

Capítulo 7

Controle de fluxo de repetição

Capítulo 8

Vetores e Matrizes

Capítulo 9

Preparando o ambiente Python

Capítulo 10

Comandos básicos de Python

Referências Bibliográficas