# Capítulo 1

# Computadores, algoritmos e programas

*'Would you tell me, please, which way I ought to go from here?'*
*'That depends a good deal on where you want to get to,' said the Cat.*
*'I don't much care where –' said Alice.*
*'Then it doesn't matter which way you go,' said the Cat.*
*'– so long as I get somewhere,' Alice added as an explanation.*
*'Oh, you're sure to do that,' said the Cat, 'if you only walk long enough.'*

Lewis Carroll, *Alice's Adventures in Wonderland*

Uma das caraterísticas de um engenheiro é a capacidade para resolver problemas técnicos. A resolução deste tipo de problemas envolve uma combinação de ciência e de arte. Por *ciência* entende-se um conhecimento dos princípios matemáticos, físicos e dos apectos técnicos que têm de ser bem compreendidos, para que sejam aplicados corretamente. Por *arte* entende-se a avaliação correta, a experiência, o bom senso e o conhecimento que permitem representar um problema do mundo real por um modelo ao qual o conhecimento técnico pode ser aplicado para produzir uma solução.

De um modo geral, qualquer problema de engenharia é resolvido recorrendo a uma sequência de fases: a *compreensão do problema* é a fase que corresponde

a perceber e a identificar de um modo preciso o problema que tem de ser resolvido; após a compreensão do problema, entra-se na fase correspondente à *especificação do problema*, na qual o problema é claramente descrito e documentado, de modo a remover dúvidas e imprecisões; no *desenvolvimento da solução* (ou *modelação da solução*) utiliza-se a especificação do problema para produzir um esboço da solução do problema, identificando métodos apropriados de resolução de problemas e as suposições necessárias; durante o desenvolvimento da solução, o esboço da solução é progressivamente pormenorizado até se atingir um nível de especificação que seja adequado para a sua realização; na *concretização da solução*, as especificações desenvolvidas são concretizadas (seja num objeto físico, por exemplo, uma ponte ou um avião, seja num objeto imaterial, por exemplo, um programa de computador); finalmente, na fase de *verificações e testes*, o resultado produzido é validado, verificado e testado.

A Engenharia Informática difere das engenharias tradicionais, no sentido em que trabalha com entidades imateriais. Ao passo que as engenharias tradicionais lidam com forças físicas, diretamente mensuráveis (por exemplo, a gravidade, os campos elétricos e magnéticos) e com objetos materiais que interagem com essas forças (por exemplo, rodas dentadas, vigas, circuitos), a Engenharia Informática lida com entidades intangíveis que apenas podem ser observadas indiretamente através dos efeitos que produzem.

A *Engenharia Informática* tem como finalidade a conceção e realização de abstrações ou modelos de entidades abstratas que, quando aplicadas por um computador, fazem com que este apresente um comportamento que corresponde à solução de um dado problema.

Sob a perspetiva da Informática que apresentamos neste livro, um computador é uma máquina cuja função é manipular informação. Por *informação* entende-se qualquer coisa que pode ser transmitida ou registada e que tem um significado associado à sua representação simbólica. A informação pode ser transmitida de pessoa para pessoa, pode ser extraída diretamente da natureza através de observação e de medida, pode ser adquirida através de livros, de filmes, da televisão, etc. Uma das caraterísticas que distinguem o computador de outras máquinas que lidam com informação é o facto de este poder manipular a informação, para além de a armazenar e transmitir. A manipulação da informação feita por um computador segue uma sequência de instruções a que se chama um *programa*. Apesar de sabermos que um computador é uma máquina complexa,

constituída por componentes eletrónicos, nem os seus componentes nem a interligação entre eles serão aqui estudados. Para a finalidade que nos propomos atingir, o ensino da programação, podemos abstrair de toda a constituição física de um computador, considerando-o uma "caixa eletrónica", vulgarmente designada pela palavra inglesa "hardware", que tem a capacidade de compreender e de executar programas.

A *Informática* é o ramo da ciência que se dedica ao estudo dos computadores e dos processos com eles relacionados: como se desenvolve um computador, como se especifica o trabalho a ser realizado por um computador, de que forma se pode tornar mais fácil de utilizar, como se definem as suas limitações e, principalmente, como aumentar as suas capacidades e o seu domínio de aplicação.

Um dos objetivos da Informática corresponde ao estudo e desenvolvimento de entidades abstratas geradas durante a execução de programas – os processos computacionais. Um *processo computacional* é um ente imaterial que evolui ao longo do tempo, executando ações que levam à solução de um problema. Um processo computacional pode afetar objetos existentes no mundo real (por exemplo, guiar a aterragem de um avião, distribuir dinheiro em caixas multibanco, comprar e vender ações na bolsa), pode responder a perguntas (por exemplo, indicar quais as páginas da internet que fazem referência a um dado termo), entre muitos outros aspetos.

A evolução de um processo computacional é ditada por uma sequência de instruções a que se chama *programa*, e a atividade de desenvolver programas é chamada *programação*. A programação é uma atividade intelectual fascinante, que não é difícil, mas que requer muita disciplina. O principal objetivo deste livro é fornecer uma introdução à programação disciplinada, ensinando os princípios e os conceitos subjacentes, os passos envolvidos no desenvolvimento de um programa e o modo de desenvolver programas bem estruturados, eficientes e sem erros.

A programação utiliza muitas atividades e técnicas que são comuns às utilizadas em projetos nos vários ramos da engenharia: a compreensão de um problema; a separação entre a informação essencial ao problema e a informação acessória; a criação de especificações pormenorizadas para o resolver; a realização destas especificações; a verificação e os testes.

Neste primeiro capítulo definimos as principais caraterísticas de um computador

e introduzimos um conceito essencial para a informática, o conceito de algoritmo.

## 1.1 Caraterísticas de um computador

Um *computador* é uma máquina cuja função é manipular símbolos. Embora os computadores difiram em tamanho, aparência e custo, eles partilham quatro caraterísticas fundamentais: são automáticos, universais, eletrónicos e digitais.

Um computador diz-se *automático* no sentido em que, uma vez alimentado com a informação necessária, trabalha por si só, sem a intervenção humana. Não pretendemos, com isto, dizer que o computador começa a trabalhar por si só (necessita, para isso, da intervenção humana), mas que o computador procura por si só a solução dos problemas. Ou seja, o computador é automático no sentido em que, uma vez o trabalho começado, ele será levado até ao final sem a intervenção humana. Para isso, o computador recebe um *programa*, um conjunto de instruções quanto ao modo de resolver o problema. As instruções do programa são escritas numa notação compreendida pelo computador (uma *linguagem de programação*), e especificam *exatamente* como o trabalho deve ser executado. Enquanto o trabalho está a ser executado, o programa está armazenado dentro do computador e as suas instruções estão a ser seguidas.

Um computador diz-se *universal*, porque pode efetuar qualquer tarefa cuja solução possa ser expressa através de um programa. Ao executar um dado programa, um computador pode ser considerado uma máquina orientada para um fim particular. Por exemplo, ao executar um programa para o tratamento de texto, um computador pode ser considerado como uma máquina para produzir cartas ou texto; ao executar um programa correspondente a um jogo, o computador pode ser considerado como uma máquina para jogar. A palavra "universal" provém do facto de o computador poder executar qualquer programa, resolvendo problemas em diferentes áreas de aplicação. Ao resolver um problema, o computador manipula os símbolos que representam a informação pertinente para esse problema, sem lhes atribuir qualquer significado. Devemos, no entanto, salientar que um computador não pode resolver qualquer tipo de problema. A classe dos problemas que podem ser resolvidos através de um computador foi estudada por matemáticos antes da construção dos primeiros computadores. Durante a década de 1930, matemáticos como Alonzo Church (1903–1995), Kurt Gödel (1906–1978), Stephen C. Kleene (1909–1994), Emil

Leon Post (1897–1954) e Alan Turing (1912–1954) tentaram definir matematicamente a classe das funções que podiam ser calculadas mecanicamente. Embora os métodos utilizados por estes matemáticos fossem muito diferentes, todos os formalismos desenvolvidos são equivalentes, no sentido em que todos definem a mesma classe de funções, as funções recursivas parciais. Pensa-se, hoje em dia, que as funções recursivas parciais são exatamente as funções que podem ser calculadas através de um computador. Este facto é expresso através da *tese de Church-Turing*[1].

De acordo com a tese de Church-Turing, qualquer computação pode ser baseada num pequeno número de operações elementares. Nos nossos programas, estas operações correspondem fundamentalmente às seguintes:

1. *Operações de entrada de dados*, as quais obtêm valores do exterior do programa;
2. *Operações de saída de dados*, as quais mostram valores existentes no programa;
3. *Operações matemáticas*, as quais efetuam cálculos sobre os dados existentes no programa;
4. *Execução condicional*, a qual corresponde ao teste de certas condições e à execução de instruções, ou não, dependendo do resultado do teste;
5. *Repetição*, a qual corresponde à execução repetitiva de certas instruções.

A tarefa de programação corresponde a dividir um problema grande e complexo, em vários problemas, cada vez menores e menos complexos, até que esses problemas sejam suficientemente simples para poderem ser expressos em termos de operações elementares.

Um computador é *eletrónico*. A palavra "eletrónico" refere-se aos componentes da máquina, componentes esses que são responsáveis pela grande velocidade das operações efetuadas por um computador.

Um computador é também *digital*. Um computador efectua operações sobre informação que é codificada recorrendo a duas grandezas discretas (tipicamente

[1] Uma discussão sobre a tese de Church-Turing e sobre as funções recursivas parciais está para além da matéria deste livro. Este assunto pode ser consultado em [Brainerd and Landwebber, 1974], [Hennie, 1977] ou [Kleene, 1975].

referidas como sendo 0 e 1) e não sobre grandezas que variam de um modo contínuo. Por exemplo, num computador o símbolo "J", poderá ser representado por 1001010.

## 1.2 Algoritmos

Ao apresentarmos as características de um computador, dissemos que durante o seu funcionamento ele segue um programa, um conjunto de instruções bem definidas que especificam exactamente o que tem que ser feito. Este conjunto de instruções é caracterizado matematicamente como um algoritmo[2]. Os algoritmos foram estudados e utilizados muito antes do aparecimento dos computadores modernos. Um programa corresponde a um algoritmo escrito numa linguagem que é entendida pelo computador, uma linguagem de programação.

Um *algoritmo* é uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais pode ser executada mecanicamente num período de tempo finito com uma quantidade de esforço finita.

Antes de continuar, vamos analisar a definição de algoritmo que acabámos de apresentar. Em primeiro lugar, um algoritmo consiste numa sequência finita de instruções. Isto quer dizer que existe uma ordem pela qual as instruções aparecem no algoritmo, e que estas instruções são em número finito. Em segundo lugar, as instruções de um algoritmo são bem definidas e não ambíguas, ou seja, o significado de cada uma das instruções é claro, não havendo lugar para múltiplas interpretações do significado de uma instrução. Em terceiro lugar, cada uma das instruções pode ser executada mecanicamente, isto quer dizer que a execução das instruções não requer imaginação por parte do executante. Finalmente, as instruções devem ser executadas num período de tempo finito e com uma quantidade de esforço finita, o que significa que a execução de cada uma das instruções termina.

Um algoritmo está sempre associado a um objetivo, ou seja, à solução de um dado problema. A execução das instruções do algoritmo garante que o seu objetivo é atingido.

[2] A palavra "algoritmo" provém de uma variação fonética da pronúncia do último nome do matemático persa Abu Ja'far Mohammed ibu-Musa al-Khowarizmi (c. 780–c. 850), que desenvolveu um conjunto de regras para efetuar operações aritméticas com números decimais. Al-Khowarizmi foi ainda o criador do termo "Álgebra" (ver [Boyer, 1974], páginas 166–167).

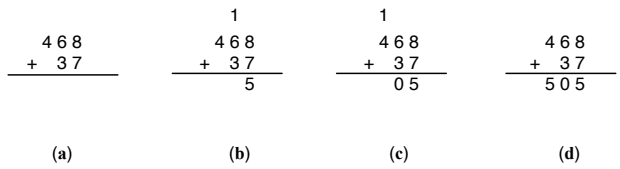


Figura 1.1: Aplicação do algoritmo para somar dois números.

### 1.2.1 Exemplos de algoritmos

Um dos primeiros algoritmos que nos é ensinado na instrução primária é o algoritmo para somar dois números arbitrariamente grandes: escrevemos os números um sobre o outro com os algarismos das unidades alinhados e traçamos uma linha horizontal por baixo dos números (Figura 1.1 (a)) e começamos a trabalhar na coluna da direita, somando todos os algarismos dessa coluna, o algarismo das unidades da soma resultante é escrito na mesma coluna por baixo da linha horizontal e, se a soma originar um número superior a 9, o algarismo das dezenas é colocado no topo da coluna imediatamente à esquerda, dizendo-se vulgarmente "e vai um"(Figura 1.1 (b)). Este processo é repetido para cada uma das colunas da direita para a esquerda até que não existam mais colunas para somar (Figura 1.1 (c), (d)). No final, a soma aparece debaixo da linha horizontal.

À medida que crescemos, apercebemo-nos que a descrição de sequências de ações para atingir objetivos tem um papel fundamental na nossa vida quotidiana e está relacionada com a nossa facilidade de comunicar. Estamos constantemente a transmitir ou a seguir sequências de instruções, por exemplo, para preencher impressos, para operar máquinas, para nos deslocarmos para certo local, para montar objetos, etc.

Vamos examinar algumas sequências de instruções utilizadas na nossa vida quotidiana. Consideremos, em primeiro lugar, a receita de "Rebuçados de ovos"[3]:

REBUÇADOS DE OVOS

500 g de açúcar;
2 colheres de sopa de amêndoas peladas e raladas;

[3]De [Modesto, 1982], página 134. Reproduzida com autorização da Editorial Verbo.

5 gemas de ovos;
250 g de açúcar para a cobertura;
e farinha.

Leva-se o açúcar ao lume com um copo de água e deixa-se ferver até fazer ponto de pérola. Junta-se a amêndoa e deixa-se ferver um pouco. Retira-se do calor e adicionam-se as gemas. Leva-se o preparado novamente ao lume e deixa-se ferver até se ver o fundo do tacho. Deixa-se arrefecer completamente. Em seguida, com a ajuda de um pouco de farinha, molda-se a massa de ovos em bolas. Leva-se o restante açúcar ao lume com 1 dl de água e deixa-se ferver até fazer ponto de rebuçado. Passam-se as bolas de ovos por este açúcar e põem-se a secar sobre uma pedra untada, após o que se embrulham em papel celofane de várias cores.

Esta receita é constituída por duas partes distintas: (1) uma descrição dos objetos a manipular; (2) uma descrição das ações que devem ser executadas sobre esses objetos. A segunda parte da receita é uma sequência finita de instruções bem definidas (para uma pessoa que saiba de culinária e portanto entenda o significado de expressões como "ponto de pérola", "ponto de rebuçado", etc., todas as instruções desta receita são perfeitamente definidas), cada uma das quais pode ser executada mecanicamente (isto é, sem requerer imaginação por parte do executante), num período de tempo finito e com uma quantidade de esforço finita. Ou seja, a segunda parte desta receita é um exemplo informal de um algoritmo.

Consideremos as instruções para montar um papagaio voador, as quais estão associadas ao diagrama representado na Figura 1.2[4].

PAPAGAIO VOADOR

1. A haste de madeira central já se encontra colocada com as pontas metidas nas bolsas A. e B;
2. Dobre ligeiramente a haste transversal e introduza as suas extremidades nas bolsas C. e D;
3. Prenda as pontas das fitas da cauda à haste central no ponto B;
4. Prenda com um nó a ponta do fio numa das argolas da aba do papagaio. Se o vento soprar forte deverá prender na argola inferior. Se o vento soprar fraco deverá prender na argola superior.

[4] Adaptado das instruções para montar um papagaio voador oferecido pela Telecom Portugal.

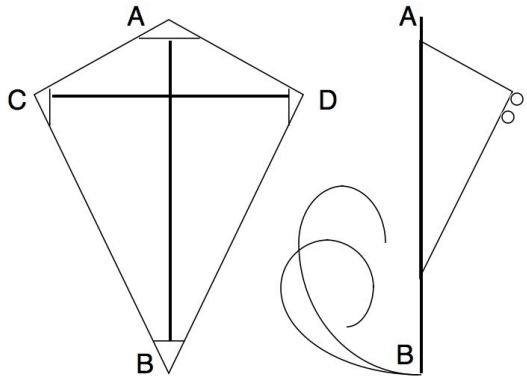


Figura 1.2: Diagrama para montar o papagaio voador.

As instruções para montar o papagaio voador são constituídas por uma descrição implícita dos objetos a manipular (mostrados na Figura 1.2) e por uma sequência de passos a seguir. Tal como anteriormente, estas instruções podem ser descritas como um algoritmo informal.

Suponhamos que desejamos deslocar-nos do Instituto Superior Técnico na Avenida Rovisco Pais (campus da Alameda) para o campus do Tagus Parque (na Av. Aníbal Cavaco Silva em Oeiras). Recorrendo ao Google Maps, obtemos a descrição apresentada na Figura 1.3. Nesta figura, para além de um mapa ilustrativo, aparecem no lado esquerdo uma sequência detalhada de instruções do percurso a seguir para a deslocação pretendida. Novamente, estas instruções podem ser consideradas como um algoritmo informal.

### 1.2.2 Caraterísticas de um algoritmo

A sequência de passos de um algoritmo deve ser executada por um agente, o qual pode ser humano, mecânico, eletrónico, ou qualquer outra coisa. Cada algoritmo está associado a um agente (ou a uma classe de agentes) que deve executar as suas instruções. Aquilo que representa um algoritmo para um agente pode não o ser para outro agente. Por exemplo, as instruções da receita dos rebuçados de ovos são um algoritmo para quem sabe de culinária e não o são para quem não

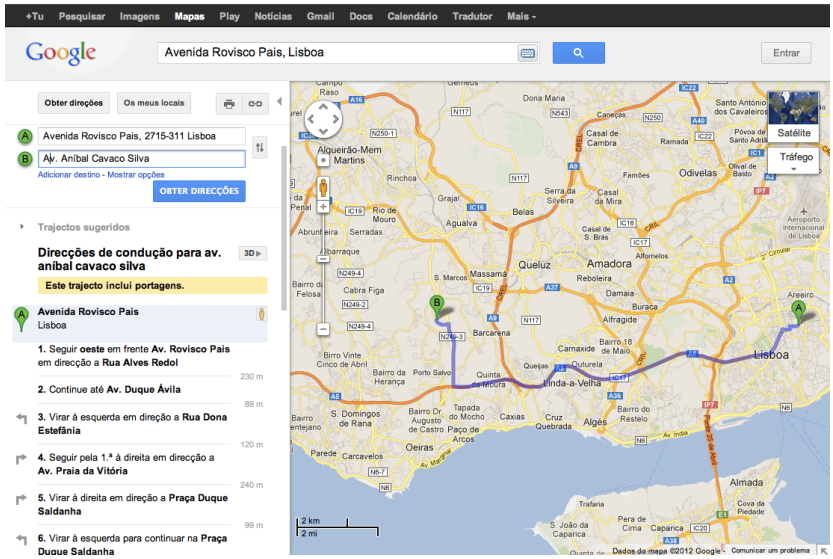


Figura 1.3: Instruções para ir do Campus da Alameda ao do Tagus Parque.

sabe.

Embora um algoritmo não seja mais do que uma descrição da sequência de passos a seguir para atingir um determinado objetivo, nem todas as sequências de passos para atingir um dado objetivo podem ser consideradas um algoritmo, pois um algoritmo deve possuir três caraterísticas, ser rigoroso, ser eficaz e ter a garantia de terminar.

1. *Um algoritmo é rigoroso.* Cada instrução do algoritmo deve especificar exata e rigorosamente o que deve ser feito, não havendo lugar para ambiguidade. O facto de um algoritmo poder ser executado mecanicamente obriga a que cada uma das suas instruções tenha uma e só uma interpretação. Por exemplo, a instrução contida na receita dos rebuçados de ovos "leva-se o açúcar ao lume com um copo de água" pode ter várias interpretações. Uma pessoa completamente ignorante de processos culinários pode ser levada a colocar um copo de água (objeto de vidro) dentro de uma panela (ou sobre o lume, interpretando a frase à letra) juntamente com o açúcar.

   Para evitar a ambiguidade inerente à linguagem utilizada pelos seres hu-

manos (chamada *linguagem natural*, de que o português é um exemplo) criaram-se novas linguagens (chamadas *linguagens artificiais*) para exprimir os algoritmos de um modo rigoroso. Como exemplos de linguagens artificiais, já conhecemos a notação matemática, a qual permite escrever frases de um modo compacto e sem ambiguidade, por exemplo, $\forall x \; \exists y \; : \; y > x$, e a notação química, que permite descrever compostos e reações químicas de um modo compacto e não ambíguo, por exemplo, $\mathrm{MgO} + \mathrm{H_2} \rightarrow \mathrm{Mg} + \mathrm{H_2O}$. A linguagem Python, discutida neste livro, é mais um exemplo de uma linguagem artificial;

2. *Um algoritmo é eficaz.* Cada instrução do algoritmo deve ser suficientemente básica e bem compreendida de modo a poder ser executada num intervalo de tempo finito, com uma quantidade de esforço finita. Para ilustrar este aspeto, suponhamos que estávamos a consultar as instruções que apareciam na embalagem do adubo "Crescimento Gigantesco", as quais incluíam a seguinte frase: "Se a temperatura máxima do mês de abril for superior a 23°, misture o conteúdo de duas embalagens em 5 litros de água, caso contrário, misture apenas o conteúdo de uma embalagem". Uma vez que não é difícil determinar qual a temperatura máxima do mês de abril, podemos decidir se deveremos utilizar o conteúdo de duas embalagens ou apenas o conteúdo de uma. Contudo, se o texto fosse: "Se a temperatura máxima do mês de abril do ano de 1143 for superior a 23°, misture o conteúdo de duas embalagens em 5 litros de água, caso contrário, misture apenas o conteúdo de uma embalagem", não seriamos capazes de determinar qual a temperatura máxima do mês de abril de 1143 e, consequentemente, não seríamos capazes de executar esta instrução. Uma instrução como a segunda que acabamos de descrever não pode fazer parte de um algoritmo, pois não pode ser executada com uma quantidade de esforço finita, num intervalo de tempo finito;

3. *Um algoritmo deve terminar.* O algoritmo deve levar a uma situação em que o objetivo tenha sido atingido e não existam mais instruções para ser executadas. Consideremos o seguinte algoritmo para elevar a pressão de um pneu acima de 28 libras: "enquanto a pressão for inferior a 28 libras, continue a introduzir ar". É evidente que, se o pneu estiver furado, o algoritmo anterior pode não terminar (dependendo do tamanho do furo) e, portanto, não o vamos classificar como algoritmo.

O conceito de algoritmo é fundamental em informática. Existem mesmo pessoas que consideram a informática como o estudo dos algoritmos: o estudo de máquinas para executar algoritmos, o estudo dos fundamentos dos algoritmos e a análise de algoritmos.

## 1.3 Programas e algoritmos

Um algoritmo, escrito de modo a poder ser executado por um computador, tem o nome de *programa*. Uma grande parte deste livro é dedicada ao desenvolvimento de algoritmos, e à sua codificação utilizando uma linguagem de programação, o Python. Os programas que desenvolvemos apresentam aspetos semelhantes aos algoritmos informais apresentados na secção anterior. Nesta secção, discutimos alguns desses aspetos.

Vimos que a receita dos rebuçados de ovos era constituída por uma descrição dos objetos a manipular (500 g de açúcar, 5 gemas de ovos) e uma descrição das ações a efetuar sobre esses objetos (leva-se o açúcar ao lume, deixa-se ferver até fazer ponto de pérola). A constituição de um programa é semelhante à de uma receita.

Num programa, existem entidades que são manipuladas pelo programa e existe uma descrição, numa linguagem apropriada, de um algoritmo que especifica as operações a realizar sobre essas entidades. Em algumas linguagens de programação, por exemplo, o C e o Java, todas as entidades manipuladas por um programa têm que ser descritas no início do programa, noutras linguagens, como é o caso do Python, isso não é necessário.

No caso das receitas de culinária, as entidades a manipular podem existir antes do início da execução do algoritmo (por exemplo, 500 g de açúcar) ou entidades que são criadas durante a sua execução (por exemplo, a massa de ovos). A manipulação destas entidades vai originar um produto que é o objetivo do algoritmo (no nosso exemplo, os rebuçados de ovos). Analogamente, nos nossos programas, iremos manipular valores de variáveis. As variáveis vão-se comportar de um modo análogo aos ingredientes da receita dos rebuçados de ovos. Tipicamente, o computador começa por receber certos valores para algumas das variáveis, após o que efetua operações sobre essas variáveis, possivelmente atribuindo valores a novas variáveis e, finalmente, chega a um conjunto de valores

que constituem o resultado do programa.

As operações a efetuar sobre as entidades devem ser compreendidas pelo agente que executa o algoritmo. Essas ações devem ser suficientemente elementares para poderem ser executadas facilmente pelo agente que executa o algoritmo. É importante notar que, pelo facto de nos referirmos a estas ações como "ações elementares", isto não significa que elas sejam operações atómicas (isto é, indecomponíveis). Elas podem referir-se a um conjunto de ações mais simples a serem executadas numa sequência bem definida.

### 1.3.1 Linguagens de programação

Definimos uma *linguagem de programação* como uma linguagem utilizada para escrever programas de computador. Existem muitos tipos de linguagens de programação. De acordo com as afinidades que estas apresentam com o modo como os humanos resolvem problemas, podem ser classificadas em linguagens máquina, linguagens "assembly" e linguagens de alto nível.

A *linguagem máquina* é a linguagem utilizada para comandar diretamente as ações do computador. As instruções em linguagem máquina são constituídas por uma sequência de dois símbolos discretos, correspondendo à existência ou à ausência de sinal (normalmente representados por 1 e por 0, respetivamente) e manipulam diretamente entidades dentro do computador. A linguagem máquina é difícil de usar e de compreender por humanos e varia de computador para computador (é a sua linguagem nativa). A *linguagem "assembly"* é semelhante à linguagem máquina, diferindo desta no sentido em que usa nomes simbólicos com significado para humanos em lugar de sequências de zeros e de uns. Tal como a linguagem máquina, a linguagem "assembly" varia de computador para computador. As *linguagens de alto nível* aproximam-se das linguagens que os humanos usam para resolver problemas e, consequentemente, são muito mais fáceis de utilizar do que as linguagens máquina ou "assembly", para além de poderem ser utilizadas em computadores diferentes. O Python é um exemplo de uma linguagem de alto nível.

Num computador, podemos identificar vários níveis de abstração (Figura 1.4): ao nível mais baixo existem os circuitos eletrónicos, o "hardware", os quais são responsáveis por executar com grande velocidade as ordens dadas ao computador; o "hardware"pode ser diretamente comandado através da linguagem

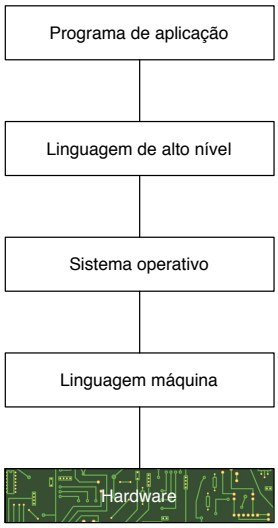


Figura 1.4: Alguns níveis de abstração existentes num computador.

máquina ou da linguagem "assembly"; o nível do *sistema operativo* permite-nos interagir com o computador, considerando que este contém ficheiros, organizados de acordo com certa hierarquia, permite a manipulação desses ficheiros, e permite a interação entre o nosso computador e o mundo exterior, o qual é composto por outros computadores e equipamento periférico, por exemplo impressoras; através do sistema operativo, podemos utilizar linguagens de programação de alto nível, de que o Python é um exemplo; finalmente, através das linguagens de alto nível, escrevemos programas de aplicação que fazem com que o computador resolva problemas específicos.

Para que os computadores possam "entender" os programas escritos numa linguagem de alto nível (recorde-se que a linguagem máquina é a linguagem que o computador compreende), existem programas que traduzem as instruções de linguagens de alto nível em linguagem máquina, chamados *processadores da linguagem*. Existem fundamentalmente dois processos para fazer esta tradução, conhecidos por *compilação* e por *interpretação*. No caso do Python, isto é feito através de um programa chamado o *interpretador*, que recebe instruções em Python e que é capaz de executar as ações correspondentes a cada uma delas.

### 1.3.2 Exemplo de um programa

Apresentamos um exemplo de algoritmo para calcular a soma dos 100 primeiros inteiros positivos e o programa correspondente em Python. Para isso, começamos por analisar o que fazemos para resolver este problema utilizando uma calculadora. O objetivo da nossa apresentação é fornecer uma ideia intuitiva dos passos e do raciocínio envolvidos na criação de um programa e, simultaneamente, mostrar um primeiro programa em Python.

Podemos descrever as ações a executar para resolver este problema através da seguinte sequência de comandos a fornecer à calculadora:

Limpar o visor da calculadora
Carregar na tecla 1
Carregar na tecla +
Carregar na tecla 2
Carregar na tecla +
Carregar na tecla 3
Carregar na tecla +
⋮
Carregar na tecla 1
Carregar na tecla 0
Carregar na tecla 0
Carregar na tecla =

Os símbolos "..." na nossa descrição de ações indicam que existe um padrão que se repete ao longo da nossa atuação e portanto não é necessário enumerar todos os passos, porque podemos facilmente gerar e executar os que estão implícitos. No entanto, a existência destes símbolos não permite qualificar o conjunto de instruções anteriores como um algoritmo, pois a caraterística do rigor deixa de se verificar. Para executar este conjunto de instruções é necessário ter a capacidade de compreender quais são os passos subentendidos por "...".

Para transformar as instruções anteriores num algoritmo que possa ser executado por um computador, é necessário tornar explícito o que está implícito. Note-se, no entanto, que para explicitar todos os passos do algoritmo anterior teríamos mais trabalho do que se executássemos o algoritmo nós próprios, pelo que deveremos encontrar uma formulação alternativa. Para isso, vamos refletir sobre o processo de cálculo aqui descrito. Existem duas grandezas envolvidas

neste processo de cálculo, a soma corrente (que aparece, em cada instante, no visor da calculadora) e o número a ser adicionado à soma (o qual é mantido na nossa cabeça). Cada vez que um número é adicionado à soma corrente, mentalmente, aumentamos em uma unidade o próximo número a ser adicionado. Se quisermos exprimir este processo de um modo rigoroso, necessitamos de recorrer a duas variáveis, uma para representar a soma corrente (à qual chamaremos *soma*), e a outra para representar o número que mantemos na nossa cabeça (a que chamaremos *numero*). Os passos que executamos sempre que adicionamos um número à soma corrente são:

A *soma* toma o valor de *soma* + *numero*
O *numero* toma o valor de *numero* + 1

Estes passos são executados repetitivamente para todos os números a somar, o que significa que a nossa tarefa corresponde a repetir estas operações *enquanto* o número a somar não exceder 100. Em programação, a repetição de uma sequência de ações chama-se um *ciclo*. O ciclo que executamos ao calcular a soma dos 100 primeiros inteiros positivos é:

enquanto o *numero* for menor ou igual a 100
    A *soma* toma o valor de *soma* + *numero*
    O *numero* toma o valor de *numero* + 1

Convém agora relembrar as operações que efetuamos antes de começar a executar esta sequência repetitiva de operações: (1) limpámos o visor da calculadora, isto é estabelecemos que o valor inicial da variável *soma* era zero; (2) estabelecemos que o primeiro *numero* a ser adicionado à soma era um. Com estes dois aspetos em mente, poderemos dizer que a sequência de passos a seguir para calcular a soma dos 100 primeiros inteiros positivos é:

A *soma* toma o valor de 0
O *numero* toma o valor de 1
enquanto o *numero* for menor ou igual a 100
    A *soma* toma o valor de *soma* + *numero*
    O *numero* toma o valor de *numero* + 1

Em matemática, operações como "toma o valor de" são normalmente representadas por um símbolo (por exemplo, =). Em programação, esta operação é também representada por um símbolo (`=`, `:=`, ou outro, dependendo da linguagem de programação utilizada). Se adotarmos o símbolo utilizado em Python,

=, o nosso algoritmo será representado por:

```
soma = 0
numero = 1
enquanto numero ≤ 100
   soma = soma + numero
   numero = numero + 1
```

Esta descrição é uma versão muito aproximada de um programa em Python para calcular a soma dos primeiros 100 inteiros positivos, o qual é o seguinte:

```
def prog_soma():
    soma = 0
    numero = 1
    while numero <= 100:
        soma = soma + numero
        numero = numero + 1
    print('O valor da soma é: ', soma)
```

A última linha do programa anterior tem por finalidade informar o mundo exterior do valor calculado pelo programa e corresponde a uma instrução de saída de dados.

É importante notar que existe uma fórmula que nos permite calcular a soma dos primeiros 100 inteiros positivos, se os considerarmos como uma progressão aritmética de razão um, a qual é dada por:

$$soma = \frac{100 \cdot (1 \ + \ 100)}{2}$$

e, consequentemente, poderíamos utilizar esta fórmula para obter o valor desejado da soma, o que poderá ser representado pelo seguinte programa em Python[5]:

```
def prog_soma():
    soma = (100 * (1 + 100)) // 2
    print('O valor da soma é: ', soma)
```

[5]Em Python, "*" representa a multiplicação e "//" representa a divisão inteira.

Um aspeto importante a reter a propósito deste exemplo é o facto de normalmente não existir apenas um algoritmo (e consequentemente apenas um programa) para resolver um dado problema. Esses algoritmos podem ser muito diferentes entre si.

## 1.4 Sintaxe e semântica

O Python, como qualquer linguagem, apresenta dois aspetos distintos: as frases da linguagem e o significado associado às frases. Estes aspetos são chamados, respetivamente, a *sintaxe* e a *semântica* da linguagem. A sintaxe determina qual a constituição das frases que podem ser fornecidas ao computador e a semântica determina o que o computador vai fazer ao seguir as indicações apresentadas em cada uma dessas frases.

### 1.4.1 Sintaxe

A *sintaxe* de uma linguagem é o conjunto de regras que definem quais as relações válidas entre os componentes da linguagem, tais como as palavras e as frases. A sintaxe nada diz em relação ao significado das frases da linguagem.

Em linguagem natural, a sintaxe é conhecida como a gramática. Analogamente, em linguagens de programação, a sintaxe também é definida através de gramáticas.

Como a sintaxe apenas se preocupa com o processo de combinação dos símbolos da linguagem, ela pode ser, na maior parte dos casos, facilmente formalizada. Os linguistas e os matemáticos estudaram as propriedades sintáticas das linguagens, e grande parte deste trabalho é aplicável às linguagens de programação.

O processo de descrição formal da sintaxe de uma linguagem consiste na apresentação de uma gramática para essa linguagem. Uma *gramática* formal é composta por:

1. Um conjunto de símbolos, os *símbolos não terminais*, que não aparecem explicitamente nas frases da linguagem mas que são utilizados para descrever os vários componentes das frases. Um símbolo não terminal está sempre associado a um conjunto de entidades da linguagem;

2. Um símbolo não terminal especial, o *símbolo inicial*, que representa o elemento principal da linguagem;

3. Um conjunto de símbolos, os *símbolos terminais*, que aparecem nas frases da linguagem;

4. Um conjunto de regras, as *regras de produção*, que descrevem a estrutura dos vários componentes da linguagem. Estas regras de produção definem todas as frases da linguagem a partir do símbolo inicial.

Para descrever a sintaxe do Python, escrevemos gramáticas utilizando uma notação conhecida por notação BNF[6], a qual utiliza as seguintes regras:

1. Os *símbolos não terminais* escrevem-se entre parênteses angulares, "⟨" e "⟩". Usando um exemplo do Python, ⟨expressão⟩ é um símbolo não terminal que corresponde a uma expressão em Python (cuja definição é apresentada na página 29). Este símbolo não terminal não representa nenhuma expressão em particular, mas sim um componente genérico da linguagem;

2. Os *símbolos terminais* escrevem-se sem qualquer símbolo especial à sua volta. Por exemplo, em Python, `+` corresponde a um símbolo terminal;

3. As regras de produção escrevem-se, usando as seguintes convenções:

   (a) O símbolo "::=" (lido "é definido como") serve para definir componentes da linguagem. Cada regra de produção define o componente que aparece à esquerda do símbolo ":=", como sendo a descrição que aparece à direita desse mesmo símbolo;

   (b) O símbolo " | " (lido "ou") representa possíveis alternativas;

   (c) A utilização do caráter "+" imediatamente após um símbolo não terminal significa que esse símbolo pode ser repetido uma ou mais vezes;

   (d) A utilização do caráter "*" imediatamente após um símbolo não terminal significa que esse símbolo pode ser repetido zero ou mais vezes;

[6] A notação BNF foi inventada por John Backus e Peter Naur, e a sua primeira utilização significativa foi na definição da sintaxe da linguagem Algol 60. O termo BNF significa "Backus-Naur Form". Alguns autores, por exemplo [Hopcroft and Ullman, 1969] e [Ginsburg, 1966], atribuem ao termo "BNF" o significado "Backus Normal Form".

(e) A utilização de chavetas, "{" e "}", englobando símbolos terminais ou não terminais, significa que esses símbolos são opcionais.

Para aumentar a facilidade de leitura das nossas gramáticas, vamos usar dois tipos de letra, respetivamente para os `símbolos terminais` e para os símbolos não terminais: os símbolos terminais são escritos utilizando uma letra correspondente ao tipo máquina de escrever (como é feito nas palavras "símbolos terminais", em cima); os símbolos não terminais são escritos usando um tipo helvética (como é feito nas palavras "símbolos não terminais", em cima). Note-se, contudo, que esta convenção apenas serve para aumentar a facilidade de leitura das expressões e não tem nada a ver com as propriedades formais das nossas gramáticas.

Como exemplo, consideremos uma gramática para definir números binários. Informalmente, um número binário é apenas constituído pelos dígitos binários `0` e `1`, podendo apresentar qualquer quantidade destes dígitos ou qualquer combinação entre eles. A seguinte gramática define números binários:

⟨número binário⟩ ::= ⟨dígito binário⟩ |
⟨dígito binário ⟩ ⟨número binário⟩

⟨dígito binário⟩ ::= 0 | 1

Nesta gramática os símbolos terminais são `0` e `1` e os símbolos não terminais são ⟨número binário⟩ (o símbolo inicial) e ⟨dígito binário⟩. A gramática tem duas regras. A primeira define a classe dos números binários, representados pelo símbolo não terminal ⟨número binário⟩, como sendo um ⟨dígito binário⟩, ou um ⟨dígito binário⟩ seguido de um ⟨número binário⟩[7]. A segunda parte desta regra diz simplesmente que um número binário é constituído por um dígito binário seguido por um número binário. Sucessivas aplicações desta regra levam-nos a concluir que um número binário pode ter tantos dígitos binários quantos queiramos (ou seja, podemos aplicar esta regra tantas vezes quantas desejarmos). Podemos agora perguntar quando é que paramos a sua aplicação. Note-se que a primeira parte desta mesma regra diz que um número binário é um dígito binário. Portanto, sempre que utilizamos a primeira parte desta regra, terminamos a sua aplicação. A segunda regra de produção define um dígito binário,

[7] É importante compreender bem esta regra. Ela representa o primeiro contacto com uma classe muito importante de definições chamadas definições *recursivas* (ou definições por *recorrência*), nas quais uma entidade é definida em termos de si própria. As definições recursivas são discutidas em pormenor no Capítulo 6.

representado pelo símbolo não terminal dígito binário, como sendo ou `0` ou `1`.

Em alternativa, poderíamos também ter apresentado a seguinte gramática para a definição de números binários:

⟨número binário⟩ ::= ⟨dígito binário⟩$^+$

⟨dígito binário⟩ ::= `0` | `1`

A notação utilizada para definir formalmente uma linguagem, no caso da notação BNF, "⟨", "⟩", "|", "::=", "{", "}", "+", "∗", os símbolos não terminais e os símbolos terminais, é denominada *metalinguagem*, visto ser a linguagem que utilizamos para falar acerca de outra linguagem (ou a linguagem que está para além da linguagem). Um dos poderes da formalização da sintaxe utilizando metalinguagem é tornar perfeitamente clara a distinção entre "falar *acerca* da linguagem" e "falar *com* a linguagem". A confusão entre linguagem e metalinguagem pode levar a paradoxos de que é exemplo a frase "esta frase é falsa".

### 1.4.2 Semântica

> *"Then you should say what you mean," the March Hare went on.*
> *"I do," Alice hastily replied; "at least–at least I mean what I say–that's the same thing, you know."*
> *"Not the same thing a bit!" said the Hatter. "You might just as well say that "I see what I eat" is the same thing as "I eat what I see"!"*
>
> Lewis Carroll, *Alice's Adventures in Wonderland*

A *semântica* de uma linguagem define qual o significado de cada frase da linguagem. A semântica nada diz quanto ao processo de geração das frases da linguagem. A descrição da semântica de uma linguagem de programação é muito mais difícil do que a descrição da sua sintaxe. Um dos processos de descrever a semântica de uma linguagem consiste em fornecer uma descrição em língua natural (por exemplo, em português) do significado, ou seja, das ações que são realizadas pelo computador, de cada um dos possíveis componentes da linguagem. Este processo, embora tenha os inconvenientes da informalidade e da ambiguidade associadas às línguas naturais, tem a vantagem de fornecer uma perspetiva intuitiva da linguagem.

Cada frase em Python tem uma semântica, a qual corresponde às ações tomadas

pelo Python ao executar essa frase, ou seja, o significado que o Python atribui à frase. Esta semântica é definida por regras para extrair o significado de cada frase, as quais são descritas neste livro de um modo incremental, à medida que novas frases são apresentadas. Utilizamos o português para exprimir a semântica do Python.

### 1.4.3 Tipos de erros num programa

De acordo com o que dissemos sobre a sintaxe e a semântica de uma linguagem, deverá ser evidente que um programa pode apresentar dois tipos distintos de erros: erros de natureza sintática e erros de natureza semântica.

Os erros de natureza sintática, ou *erros sintáticos* resultam do facto de o programador não ter escrito as frases do seu programa de acordo com as regras da gramática da linguagem de programação utilizada. A deteção destes erros é feita pelo processador da linguagem, o qual fornece normalmente um diagnóstico sobre o que provavelmente está errado. Todos os erros de natureza sintática têm que ser corrigidos antes da execução das instruções, ou seja, o computador não executará nenhuma instrução sintaticamente incorreta. Os programadores novatos passam grande parte do seu tempo a corrigir erros sintáticos, mas à medida que se tornam mais experientes, o número de erros sintáticos que originam é cada vez menor e a sua origem é detetada de um modo cada vez mais rápido.

Os erros de natureza semântica, ou *erros semânticos* (também conhecidos por *erros de lógica*) são erros em geral muito mais difíceis de detetar do que os erros de caráter sintático. Estes erros resultam do facto de o programador não ter expressado corretamente, através da linguagem de programação, as ações a serem executadas (o programador queria dizer uma coisa mas disse outra). Os erros semânticos podem-se manifestar pela geração de uma mensagem de erro durante a execução de um programa, pela produção de resultados errados ou pela geração de ciclos que nunca terminam. Neste livro apresentaremos técnicas de programação que permitem minimizar os erros semânticos e, além disso, discutiremos métodos a utilizar para a deteção e correção dos erros de natureza semântica de um programa.

Ao processo de deteção e correção, tanto dos erros sintáticos como dos erros semânticos, dá-se o nome de *depuração* (do verbo depurar, tornar puro, limpar). Em inglês, este processo é denominado "*debugging*" e aos erros que existem

num programa, tanto sintáticos como semânticos, chamam-se "*bugs*"[8]. O termo "bug" foi criado pela pioneira da informática Grace Murray Hopper (1906–1992). Em agosto de 1945, Hopper e alguns dos seus associados estavam a trabalhar em Harvard com um computador experimental, o Mark I, quando um dos circuitos deixou de funcionar. Um dos investigadores localizou o problema e, com auxílio de uma pinça, removeu-o: uma traça com cerca de 5 cm. Hopper colou a traça, com fita gomada, no seu livro de notas e disse: "A partir de agora, sempre que um computador tiver problemas direi que ele contém insetos (bugs)". A traça ainda hoje existe, juntamente com os registos das experiências, no "U.S. Naval Surface Weapons Center" em Dahlgran, Virginia, nos Estados Unidos da América[9].

Para desenvolver programas, são necessárias duas competências fundamentais, a capacidade de *resolução de problemas* que corresponde à competência para formular o problema que deve ser resolvido pelo programa, criar uma solução para esse problema, através da sua divisão em vários subproblemas mais simples, e expressar essa solução de um modo rigoroso recorrendo a uma linguagem de programação e a capacidade de *depuração* que consiste em, através de uma análise rigorosa, perceber quais os erros existentes no programa e corrigi-los adequadamente. A depuração é fundamentalmente um trabalho de detetive em que se analisa de uma forma sistemática o que está a ser feito pelo programa, formulando hipóteses sobre o que está mal e testando essas hipóteses através da modificação do programa. A depuração semântica é frequentemente uma tarefa difícil, requerendo espírito crítico e persistência.

## 1.5 Notas finais

Neste capítulo apresentámos alguns conceitos básicos em relação à programação. Um computador, como uma máquina cuja função é a manipulação de símbolos, e as suas caraterísticas fundamentais, ser automático, universal, eletrónico e digital. Uma apresentação informal muito interessante sobre as origens dos computadores e dos matemáticos ligados à sua evolução pode ser consultada em [Davis, 2004].

Introduzimos a noção de programa, uma sequência de instruções escritas numa

[8] Do inglês, insetos.
[9] Ver [Taylor, 1984], página 44.

linguagem de programação, e o resultado originado pela execução de um programa, um processo computacional.

Apresentámos o conceito de algoritmo, uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais pode ser executada mecanicamente num período de tempo finito com uma quantidade de esforço finita, bem como as suas caraterísticas, ser rigoroso, eficaz e dever terminar. O aspeto de um algoritmo ter que terminar é de certo modo controverso. Alguns autores, por exemplo [Hennie, 1977] e [Hermes, 1969], admitem que um algoritmo possa não terminar. Para estes autores, um algoritmo apenas apresenta as caraterísticas de rigor e de eficácia. Outros autores, por exemplo [Brainerd and Landwebber, 1974] e [Hopcroft and Ullman, 1969], distinguem entre procedimento mecânico – uma sequência finita de instruções que pode ser executada mecanicamente – e um algoritmo – um procedimento mecânico que é garantido terminar. Neste livro, adotamos a segunda posição.

Finalmente apresentámos os dois aspetos associados a uma linguagem, a sintaxe e a semântica, e introduzimos a notação BNF para definir a sintaxe de uma linguagem. Como a sintaxe apenas se preocupa com o processo de combinação dos símbolos de uma dada linguagem, ela pode ser, na maior parte dos casos, facilmente formalizada. Os linguistas e os matemáticos estudaram as propriedades sintáticas das linguagens, e grande parte deste trabalho é aplicável às linguagens de programação. É particularmente importante o trabalho de Noam Chomsky ([Chomsky, 1957] e [Chomsky, 1959]), que classifica as linguagens em grupos. Grande parte das linguagens de programação pertence ao grupo 2, ou grupo das linguagens livres de contexto[10].

## 1.6 Exercícios

1. Escreva uma gramática em notação BNF para definir um número inteiro. Um número inteiro é um número, com ou sem sinal, constituído por um número arbitrário de dígitos.

2. Escreva uma gramática em notação BNF para definir um número real, o qual pode ser escrito quer em notação decimal quer em notação científica. Um real em notação decimal pode ou não ter sinal, e tem que ter ponto

[10] Do inglês, "context-free languages"

decimal, o qual é rodeado por dígitos. Por exemplo, `+4.0`, `-4.0` e `4.0` são números reais em notação decimal. Um real em notação científica tem uma mantissa, a qual é um inteiro ou um real, o símbolo "`e`"e um expoente inteiro, o qual pode ou não ter sinal. Por exemplo, `4.2e-5`, `2e4` e `-24.24e+24` são números reais em notação científica.

3. Considere a seguinte gramática em notação BNF, cujo símbolo inicial é ⟨S⟩:

   ⟨S⟩ :: = ⟨A⟩ `a`

   ⟨A⟩ :: = `a` ⟨B⟩

   ⟨B⟩ :: = ⟨A⟩ `a` | `b`

   (a) Diga quais são os símbolos terminais e quais são os símbolos não terminais da gramática.

   (b) Quais das frases pertencem ou não à linguagem definida pela gramática. Justifique a sua resposta.

   `aabaa`

   `abc`

   `abaa`

   `aaaabaaaa`

4. Considere a seguinte gramática em notação BNF, cujo símbolo inicial é ⟨idt⟩:

   ⟨idt⟩ ::= ⟨letras⟩ ⟨numeros⟩

   ⟨letras⟩ ::= ⟨letra⟩ |
   ⟨letra⟩ ⟨letras⟩

   ⟨numeros⟩ ::= ⟨num⟩ |
   ⟨num⟩ ⟨numeros⟩

   ⟨letra⟩ ::= `A` | `B` | `C` | `D`

   ⟨num⟩ ::= `1` | `2` | `3` | `4`

   (a) Diga quais são os símbolos terminais e quais são os símbolos não terminais da gramática.

   (b) Quais das seguintes frases pertencem à linguagem definida pela gramática? Justifique a sua resposta.

   `ABCD`

`1CD`

`A123CD`

`AAAAB12`

(c) Descreva informalmente as frases que pertencem à linguagem.

5. Escreva uma gramática em notação BNF que defina frases da seguinte forma: (1) as frases começam por `c`; (2) as frases acabam em `r`; (3) entre o `c` e o `r` podem existir tantos `a`'s e `d`'s quantos quisermos, mas tem que existir pelo menos um deles. São exemplos de frases desta linguagem: `car`, `cadar`, `cdr` e `cdddddr`.

6. Considere a representação de tempo utilizada em relógios digitais, na qual aparecem as horas (entre 0 e 23), minutos e segundos. Por exemplo 10:23:45.

   (a) Descreva esta representação utilizado uma gramática em notação BNF.

   (b) Quais são os símbolos terminais e quais são os símbolos não terminais da sua gramática?

7. Dada a seguinte gramática em notação BNF, cujo símbolo inicial é ⟨S⟩:

   ⟨S⟩ ::= `b` ⟨B⟩

   ⟨B⟩ ::= `b` ⟨C⟩ | `a` ⟨B⟩ | `b`

   ⟨C⟩ ::= `a`

   (a) Diga quais os símbolos terminais e quais são os símbolos não terminais desta gramática.

   (b) Diga, justificando, se as seguintes frases pertencerem ou não à linguagem definida pela gramática:

   `baaab`

   `aabb`

   `bba`

   `baaaaaaba`