P A R T E I

FUNDAMENTOS DE PROGRAMAÇÃO

Capítulo 1
Introdução à ciência da computação e à programação

Sumário

INTRODUÇÃO

Os computadores eletrônicos modernos são um dos produtos mais importantes dos séculos XX e XXI e especialmente da década atual. Uma ferramenta essencial em muitas áreas: indústria, governo, ciência, educação..., na realidade, em quase todos os campos das nossas vidas. O papel dos programas de computadores é fundamental; sem uma lista de instruções a seguir, o computador é praticamente inútil. As linguagens de programação nos permitem escrever esses programas e, por conseqüência, nos comunicarmos com os computadores.

Nesta obra, você começará a estudar a ciência da computação ou informática por meio de uma das linguagens de programação mais versáteis disponíveis atualmente: a linguagem C++. Este capítulo lhe apresenta o computador e seus componentes, bem como as linguagens de programação e a metodologia a seguir para resolver problemas com computadores com uma ferramenta chamada C++.

Neste capítulo, serão descritos o conceito e a organização física (*hardware*) e lógica (*software*) de um computador, juntamente com as diferentes formas de representação da informação. Outro tema que será abordado é o conceito de algoritmo como ferramenta de resolução de problemas.

Os dois paradigmas mais populares que dão suporte à linguagem de programação C++ são: *programação estruturada* e *programação orientada a objetos*. Juntamente com as características dos diferentes tipos de software – em particular o sistema operacional – e das linguagens de programação, em particular C++ e UML 2.0, articula-se a segunda parte do capítulo.

CONCEITOS-CHAVE

- Algoritmo.
- CD-ROM, CDR/W.
- Compilador.
- Computador.
- Diagrama de fluxo.
- Diagrama N-S.
- Disquete.
- DVD.
- DVD alta definição.
- *Hardware*.
- Interpretador.
- Linguagem de programação.
- Linguagem *assembler* ("montadora").
- Memória.
- Memória auxiliar.
- Memória central.
- Microprocessador.
- Modem.
- Software.
- Unidade central de processamento.

1.1 O QUE É UM COMPUTADOR?

Um **computador** é um dispositivo eletrônico utilizado para processar informação e obter resultados. Os dados e a informação podem ser introduzidos no computador pela **entrada** (*input*) e, em seguida, são processados para produzir uma **saída** (*output – resultados*), como se observa na Figura 1.1. O computador pode ser considerado uma unidade na qual são introduzidos certos dados, *entrada de* dados, que são processados e produzidos *dados de saída*. Os dados de entrada e de saída podem ser de diferentes tipos, texto, desenhos ou som. O sistema mais simples de comunicação entre uma pessoa e o computador é essencialmente por meio de um *mouse*, um teclado e um monitor. Hoje em dia, existem outros dispositivos muito populares, tais como *scanners*, microfones, auto-falantes, câmeras de vídeo, câmeras digitais etc.; da mesma maneira, mediante *modems*, é possível conectar seu computador, por redes, com outros computadores. A rede de conexão mais importante é a **Internet**.

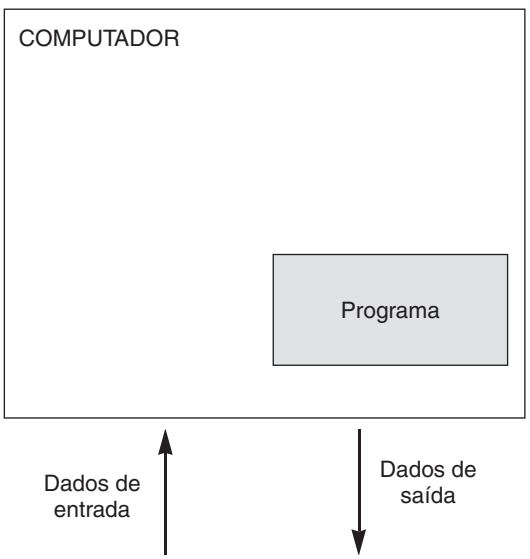


Figura 1.1 Processamento de informação em um computador.

Os componentes físicos que compõem um computador, juntamente com os dispositivos que realizam as tarefas de entrada e de saída, são conhecidos como ***hardware***. O conjunto de instruções que fazem funcionar o computador é denominado ***programa*** e se encontra armazenado em sua memória; a pessoa que escreve programas é chamada ***programador*** e ao conjunto de programas escritos para um computador se chama ***software***. Este livro se dedicará quase exclusivamente ao *software*, porém se fará uma breve revisão do *hardware* como recordação ou introdução de acordo com os conhecimentos do leitor nessa matéria. No Apêndice 1, disponível no site do livro em www.mcgraw-hill.com.br, pode ser encontrada ampla informação sobre "Introdução aos computadores", caso o aluno deseje ampliar esse tema.

1.2 ORGANIZAÇÃO FÍSICA DE UM COMPUTADOR (HARDWARE)

A maioria dos computadores, grandes ou pequenos, está organizada como mostra a Figura 1.2. Contém, fundamentalmente, três componentes principais: *Unidade Central de Processamento* (**CPU**) ou *processador* (composto pela **UAL**, **U**nidade **A**ritmética e **L**ógica, e **UC**, **U**nidade de **C**ontrole); a *memória principal* ou *central* e o *programa*.

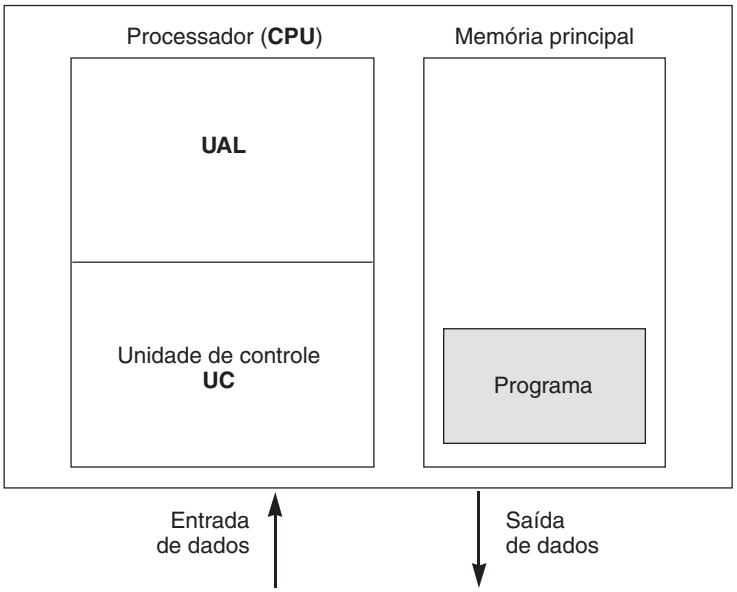


Figura 1.2 Organização física de um computador.

Se adicionarmos os dispositivos para comunicação com o computador, como a organização física da Figura 1.2, teremos a estrutura típica de um sistema de computador: dispositivos de entrada, dispositivos de saída, memória externa e o processador/memória central com seu programa (Figura 1.3).

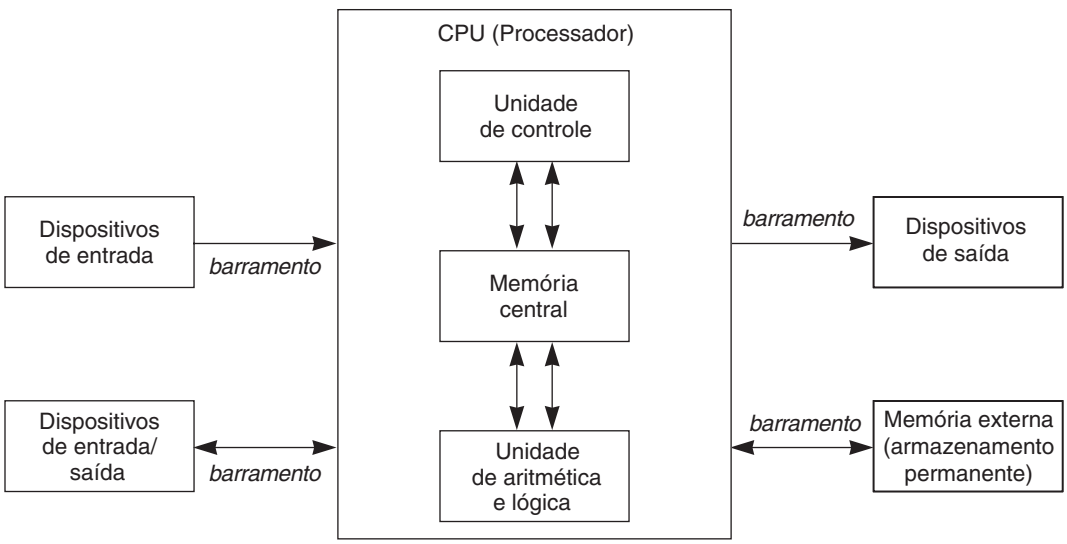


Figura 1.3 Organização física de um computador.

1.2.1 Dispositivos de Entrada/Saída (E/S)

Os dispositivos de *Entrada/Saída* (**E/S**) (em inglês, *Input/Output* **I/O**) ou *periféricos* permitem a comunicação entre o computador e o usuário.

Os *dispositivos de entrada*, como o próprio nome indica, servem para introduzir dados (informação) no computador para serem processados. Os dados são *lidos* dos dispositivos de entrada e armazenados na memória central ou interna. Os dispositivos de entrada convertem a informação de entrada em sinais elétricos que são armazenados na memória central. Dispositivos de entrada típicos são **teclados**, **leitores de cartões perfurados** (já em desuso), **canetas ópticas**, **controle** (*joystick*), **leitores de códigos de barra**, ***scanners***, **microfones**, **leitores de cartões**

digitais, **leitores RFID** (cartões de identificação por radiofreqüência) etc. Atualmente, talvez o dispositivo de entrada mais popular seja o ***mouse***, que move o ponteiro gráfico (eletrônico) sobre a tela do monitor que facilita a interação usuário-máquina.[1]

Os *dispositivos de saída* permitem representar os resultados (saída) do processamento dos dados. O dispositivo de saída típico é a **tela (CRT)**[2] ou **monitor**. Outros dispositivos de saída são: **impressoras** (imprimem resultados em papel), **traçadores gráficos** (*plotters*), **sintetizadores de voz**, **alto-falantes** etc.

Dispositivos de entrada/saída e *dispositivos de armazenamento* em massa ou auxiliar (memória externa) são: unidade de discos (disquetes, **CD-ROM**, **DVD**, fitas, discos rígidos etc.), câmera de vídeo, memórias *flash*, **USB** etc.

Figura 1.4 Dispositivo de saída (impressora HP Color Laser Jet 2600n).

1.2.2 A memória central (interna)

A **memória central** ou simplesmente **memória** (*interna* ou *principal*) é utilizada para armazenar informação (**RAM**, **R**andom, **A**ccess **M**emory). Geralmente, a informação armazenada em memória pode ser de dois tipos: *instruções*, de um programa, e *dados*, com os quais se operam as instruções. Por exemplo, para que um programa possa ser *executado* (rodar, funcionar..., em inglês *run*) deve estar situado na memória central em uma operação chamada *carga* (*load*) do programa. Depois, quando se executa o programa, qualquer dado a ser processado por ele deve ser levado à memória por meio das instruções do programa. Na memória central, há também dados diversos e espaço de armazenamento temporário que o programa necessita para que possa funcionar.

Execução

Quando se executa um programa em um computador, diz-se que se *executa*.

Com o objetivo de que o processador possa obter mais rapidamente os dados da memória central, em geral todos os processadores atuais (muito rápidos) utilizam uma *memória* denominada *caché*, que serve para armazenamento intermediário de dados entre o processador e a memória principal. A memória *caché* – atualmente – quase sempre é incorporada ao processador.

Organização da memória

A memória central de um computador é uma zona de armazenamento organizada em centenas ou milhares de unidades de armazenamento individual ou células. A memória central é formada por um conjunto de *células de memória* (essas células ou posições de memória também são chamadas *palavras*, ainda que não tenham nenhuma

[1] Todas as ações são realizadas pelo usuário com o *mouse*, exceto aquelas que necessitam da digitação de dados pelo teclado.
[2] *Cathode Ray Tube* (Tubo de raios catódicos).

analogia com as palavras da linguagem). O número de células de memória, da memória central, depende do tipo e modelo do computador, hoje esse número costuma ser milhões (512, 1.024 etc.). Cada célula de memória tem determinado número de bits (normalmente, 8, um *byte*).

A unidade elementar de memória se chama *byte* (octeto). Um *byte* tem a capacidade de armazenar um caractere de informação e é formado por um conjunto de unidades de armazenamento menores denominado *bits* que são dígitos binários (0 ou 1).

Figura 1.5 Computador digital portátil.

Por definição, é aceito que um byte contém oito bits. Conseqüentemente, se desejamos armazenar a frase:

```
Olá Mortimor está tudo bem
```

o computador utilizará exatamente 27 bytes consecutivos de memória. Observe que, além das letras, existem quatro espaços em branco e um ponto (um espaço é um caractere que utiliza também um byte). De modo similar, o número do passaporte

```
P57487891
```

ocupará 9 bytes, porém se for armazenado como

```
P5-748-7891
```

ocupará 11. Esses dados são chamados *alfanuméricos* e podem conter letras do alfabeto, dígitos e, inclusive, caracteres especiais (símbolos: $, #, * etc.).

Enquanto cada caractere de dado alfanumérico é armazenado em um byte, a informação numérica é armazenada de modo diferente. Os dados numéricos ocupam 2, 4 e, inclusive, 8 bytes consecutivos, dependendo do tipo de dado numérico (como veremos no Capítulo 12).

Existem dois conceitos importantes associados a cada célula ou posição de memória: seu *endereço* e seu *conteúdo*. A cada célula ou byte está associada um único endereço que indica sua *posição* relativa na memória e mediante a qual se pode acessar a posição para armazenar ou recuperar informação. A informação armazenada em uma posição de memória é seu *conteúdo*. A Figura 1.6 mostra uma memória de computador que contém 1.000 posições em memória com endereços de 0 a 999. O conteúdo desses endereços ou posições de memória se chama palavras, de modo que existam palavras de 8, 16, 32 e 64 bits. Por conseguinte, trabalhar com uma máquina de 32 bits significa que cada posição de memória do seu computador pode alojar 32 bits, quer dizer, 32 dígitos binários, que sejam zero ou um.

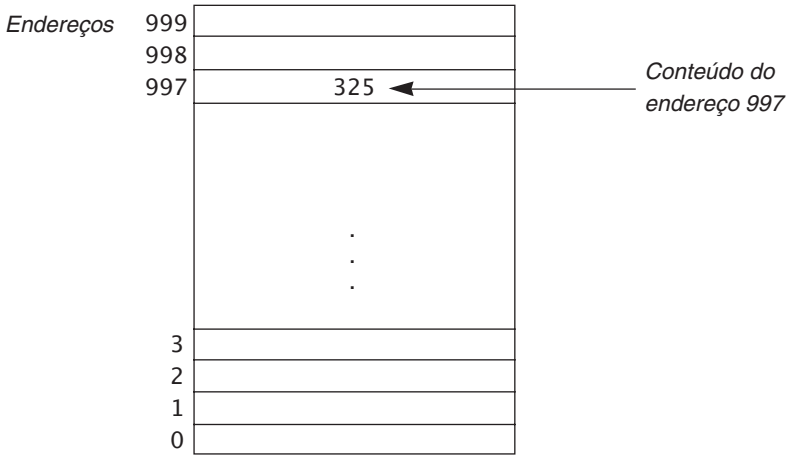


Figura 1.6 Memória central de um computador.

Sempre que é armazenada uma nova informação em uma posição, destrói-se (desaparece) qualquer informação que havia nela anteriormente e não é possível recuperá-la. O endereço é permanente e único, o conteúdo pode mudar enquanto o programa é executado.

A memória central de um computador pode ter de centenas de milhares de bytes até milhões de bytes. Como o byte é uma unidade elementar de armazenamento, são utilizados múltiplos de potência de 2 para definir o tamanho da memória central: *Kilobyte* (**KB** ou **Kb**) igual a 1.024 bytes (2^{10}) – praticamente se consideram 1.000; *Megabyte* (**MB** ou **Mb**) igual a 1.024 × 1.024 bytes = 1.048.576 (20^{20}) – normalmente se consideram 1.000.000; Gigabyte (**GB** ou **Gb**) igual a 1.024 MB (2^{30}), 1.073.741.824 = geralmente se consideram 1.000 milhões de MB.

Tabela 1.1 Unidades de medida de armazenamento

Byte	**Byte (B)**	*equivale a*	8 bits
Kilobyte	**Kbyte (KB)**	*equivale a*	1.024 bytes
Megabyte	**Mbyte (MB)**	*equivale a*	1.024 Kbytes
Gigabyte	**Gbyte (GB)**	*equivale a*	1.024 Mbytes
Terabyte	**Tbyte (TB)**	*equivale a*	1.024 Gbytes
1 **Tb** = 1.024 **Gb** = 1.024 × 1.024 **Mb** = 1.048.576 **Kb** = 1.073.741.824 **B**			

Atualmente, os computadores pessoais tipo PC costumam ter memórias centrais de 512 MB a 2 GB, ainda que seja muito freqüente encontrá-los com memórias de 4 GB e até 8 GB.

A memória principal é a encarregada de armazenar os programas e dados que estão sendo executados e sua principal característica é que o acesso aos dados ou instruções é muito rápido.

Na memória principal são armazenados:

- Os dados enviados dos dispositivos de entrada para processamento.
- Os programas que realizarão os processamentos.
- Os resultados obtidos preparados para serem enviados a um dispositivo de saída.

Tipos de memória principal

Na memória principal, podem-se distinguir dois tipos de memória: **RAM** e **ROM**. A memória **RAM** (**R**andom **A**ccess **M**emory – memória de acesso aleatório) armazena os dados e instruções a serem processados. É um tipo de memória volátil (seu conteúdo se perde quando se desliga o computador); essa memória é, na realidade, a que se conhece como memória principal ou de trabalho; nessa memória, podem ser escritos e lidos dados. A memória

ROM (**R**ead **O**nly **M**emory – memória somente de leitura) é uma memória permanente na qual não se pode escrever (vem pré-gravada pelo fabricante); é uma ***memória somente de leitura***. Os programas armazenados em ROM não são perdidos ao se desligar o computador e, ao ligá-lo, se lê a informação armazenada nessa memória. Por ser uma memória somente de leitura, os programas armazenados nos *chips* ROM não podem ser modificados e costumam ser utilizados para armazenar os programas básicos que servem para iniciar o computador.

1.2.3 A Unidade Central de Processamento (CPU): o Processador

A *Unidade Central de Processamento* – **CPU** (*Central Processing Unit*, CPU em Inglês) dirige e controla o processamento de informação realizado pelo computador. A CPU processa e manipula a informação armazenada na memória; pode recuperar informação da memória (essas informações são dados ou instruções: programas). Também pode armazenar os resultados desses processamentos na memória para utilizá-los posteriormente.

A **CPU** é formada por dois componentes: *unidade de controle* (**UC**) e unidade de aritmética e lógico (**UAL**) (Figura 1.7). A **unidade de controle** (*Control Unit*, **CU**) coordena as atividades do computador e determina quais operações devem ser realizadas e em que ordem, além disso controla e sincroniza todo o processamento do computador. A unidade de **aritmética-lógica** (*Arithmetic-Logic Unit*, **ALU**) realiza operações aritméticas e lógicas, tais como adição, subtração, multiplicação, divisão e comparações. Os dados da memória central podem ser *lidos* (recuperados) ou *escritos* (mudados) pela CPU.

1.2.4 O microprocessador

O **microprocessador** é um *chip* (**um circuito integrado)** que controla e realiza as funções e operações com os dados. É conhecido como *processador* e é o cérebro e o coração do computador. Na realidade, o microprocessador representa a Unidade Central de Processamento de um computador.

O primeiro microprocessador comercial, o Intel 4004, foi apresentado em 15 de novembro de 1971. Existem diferentes fabricantes de microprocessadores, tais como Intel, Zilog, AMD, Motorola, Cyrix etc. Microprocessadores históricos de 8 bits são o Intel 8080, Zilog Z80 ou Motorola 6800; outros microprocessadores muito populares foram: 8086, 8088 da Intel ou o Motorola MC68000. Na década de 1980, eram populares: Intel 80286, 80386, 80486; Motorola, MC 68020, MC 68400; AMD 80386, 80486.

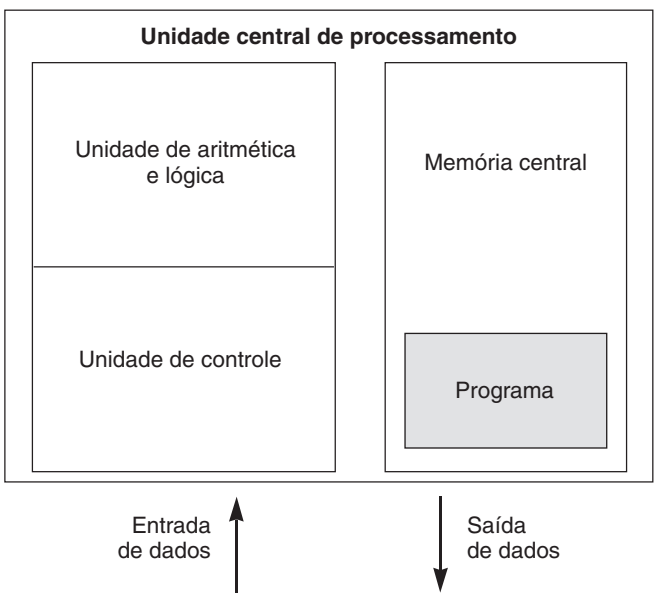


Figura 1.7 Unidade central de processamento.

Em 1993, surgiram o Intel Pentium e, durante essa década, o Intel Pentium Pro, Intel Pentium II/III e AMD K6. Em 2000, Intel e **AMD** controlam o mercado com Intel Pentium IV, Intel Titanium, Intel Pentium D ou AMD Athlon XP, AMD Duxor. Nos anos 2005 e 2006, surgiram as novas tecnologias **Intel Core Duo**, **AMD Athlon 64** etc.

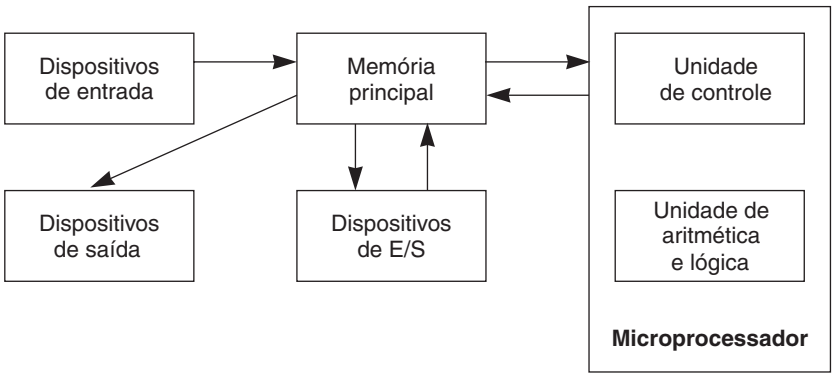


Figura 1.8 Organização física de um computador com um microprocessador.

1.2.5 Memória externa: armazenamento em massa

Quando um programa é executado, deve situar-se primeiro na memória central como acontece com os dados. No entanto, a informação armazenada na memória se perde (apaga) quando o computador é *desligado* (desconectado da rede elétrica) e, por sua vez, a memória central é limitada em sua capacidade. Por essa razão, para poder dispor de armazenamento permanente, tanto para programas quanto para dados, são necessários *dispositivos de armazenamento secundário*, *auxiliar* ou em massa (*mass storage* ou *secondary storage*).

Os **dispositivos de armazenamento** ou **memórias auxiliares** (*externas* ou *secundárias*) mais utilizados são: *fitas magnéticas*, *discos magnéticos*, *discos compactos* (**CD-ROM**, **C**ompact **D**isk **R**ead **O**nly **M**emory) e videodiscos digitais (**DVD**). As fitas são utilizadas principalmente por sistemas de computadores grandes, similares às usadas nos equipamentos de áudio. Os discos e disquetes magnéticos são utilizados por todos os computadores, especialmente os médios e pequenos – os computadores pessoais. Os discos podem ser *rígidos*, de grande capacidade de armazenamento (sua capacidade atual oscila entre 40 GB e 500 GB) e os **disquetes** ou *discos flexíveis* (*floppy disk*), estes já praticamente em desuso, porém, mesmo assim, ainda são comercializados leitores de disquetes compatíveis com computadores antigos. O disquete, praticamente fora de uso, é de 3,5" e de 1,44 MB de capacidade.

Figura 1.9 Memórias auxiliares: cartão compact flash (esquerda), memória flash USB (centro) e disco rígido (direita).

Os discos compactos (conhecidos popularmente como **CD**) são suportes digitais ópticos utilizados para armazenar qualquer tipo de informação (áudio, vídeo, documentos...). Foi desenvolvido em 1980 e começou a ser comercializado em 1982. Existem diferentes modelos: CD-ROM (somente de leitura), **CD-R** (gravável), **CD-RW** (regravável). Sua capacidade de armazenamento vai de 650 MB a 875 MB e inclusive 215 MB.

Os **DVDs** constituem um formato multimídia de armazenamento óptico e podem ser utilizados para guardar dados, incluindo filmes de alta qualidade de vídeo e áudio. Os formatos mais populares são: DVD-ROM, DVD±R, DVD±RW e DVD±RAM e suas capacidades de armazenamento vão de 4,7 GB e 8,5 GB a 17,1 GB de acordo com uma face, de duas faces, de camada simples ou dupla.

Os últimos discos ópticos lançados no mercado durante 2006 são o ***Blu-ray*** e **HD DVD**. Estes são discos de alta definição e com grande capacidade de armazenamento, de 15 GB a 50 GB, e, no futuro, poderá chegar até 200 GB.

A informação armazenada na memória central é *volátil* (desaparece quando se desliga o computador), enquanto a informação armazenada na memória externa (em massa) é *permanente*.

Essa informação é organizada em unidades independentes chamadas **arquivos** (*file* em inglês). Os resultados dos programas podem ser guardados como *arquivos de dados* e os programas escritos podem ser guardados como *arquivos de programas*, ambos na memória auxiliar. Qualquer tipo de arquivo pode ser facilmente transferido da memória auxiliar para a memória central para ser processado posteriormente.

No campo dos computadores é freqüente utilizar a palavra memória e armazenamento ou memória externa indistintamente. Neste livro se utilizará – e recomendamos seu uso – o termo memória somente para referir-se à memória central.

Comparação da memória central e memória externa

A memória central ou principal é muito mais rápida e cara que a memória externa. Os dados devem ser transferidos da memória externa para a memória central antes de serem processados. Os dados na memória central são *voláteis* e desaparecem quando se *desliga* o computador. Os dados na memória externa são *permanentes* e não desaparecem quando se *desliga* o computador.

Os computadores modernos necessitam comunicar-se com outros computadores. Se o computador possui conexão com um *cartão de rede*, pode conectar-se a uma rede de dados locais (*rede de área local*). Desse modo, podemos acessar e compartilhar cada uma das memórias de disco e outros dispositivos de entrada e de saída. Se o computador tem um *modem*, pode comunicar-se com computadores distantes. Podemos nos conectar a uma rede de dados ou *enviar correio eletrônico* por meio das redes corporativas Intranet/Extranet ou da própria rede Internet. As redes sem fio permitem conexões com a Internet de numerosos lugares, sempre que seu PC disponha de cartões ou conexões sem fio.

1.2.6 O computador pessoal ideal para programação

Hoje em dia, o estudante de informática ou de computação e, principalmente, o profissional de informática dispõem de amplo leque de computadores a preços acessíveis e com altos benefícios. No segundo semestre de 2006, podiam ser encontrados computadores pessoais (PC) com 1.024 MB de memória, gravadores de DVD de camada dupla, processador Pentium de 3.00/3.20 GH2 e um monitor plano de 17", disco rígido de 200/400 GB. Em relação a computadores portáteis, podem ser encontrados modelos de 1.024 MB, 60-80 GB, processadores Intel Pentium de 1,736 GHz, 1,83 GHz, tela de 15,4". A Tabela 1.2 resume nossa proposta e recomendação de características médias para um PC, em meados de 2006.

Tabela 1.2 Características médias de um computador portátil (*laptop*)

Processador	Intel Centrino 1.6 a 1,73/2.0 GHz; Intel CorneDuo (1.73/1.83 GHz; AMD Turion 64).
Memória	512 MB a 1.0 GB/2 GB.
Disco rígido	60-120 GB.
Internet	Cartão de rede; modem 56 Kbps; rede sem fio 802,11 b/g; *Bluetooth*.
Vídeo	Memória de vídeo de 128 a 512 MB.
Monitor	15", 15,4" ou 17" (também são comercializados de 11", 12" e 13").
Armazenamento	Gravador DVD +/− RW de camada dupla.
Portas	3/4 portas USB 2.0, 1 IEEE, leitor de cartões.
Marcas	HP, Compaq, Dell, IBM, EI System, Gateways, Acer, Toshiba, Sony, Cofiman…

1.3 REPRESENTAÇÃO DA INFORMAÇÃO NOS COMPUTADORES

Um computador é um sistema para processar informação de modo automático. Um tema fundamental no processamento de funcionamento de um computador é estudar a forma de representação da informação nesse computador. É necessário levar em consideração como pode ser codificada a informação em padrões de bits que sejam facilmente armazenados e processáveis pelos elementos internos do computador.

As formas de informação mais significativas são: textos, sons, imagens e valores numéricos, e cada uma delas apresenta peculiaridades distintas. Outros temas importantes no campo da programação se referem aos métodos de detecção dos erros que podem ser produzidos na transmissão ou no armazenamento da informação e as técnicas e os mecanismos de compressão da informação ao objeto, de modo que a informação ocupe o menor espaço nos dispositivos de armazenamento e seja mais rápida sua transmissão.

1.3.1 Representação de textos

A informação em formato de texto é representada mediante um código no qual a cada um dos distintos símbolos do texto (tais como letras do alfabeto ou sinais de pontuação) é atribuído um único padrão de bits. O texto é representado como uma extensa cadeia de bits na qual os sucessivos padrões representam os sucessivos símbolos do texto original.

Em resumo, pode ser representada qualquer informação escrita (texto) mediante caracteres. Os caracteres que são utilizados em computação costumam agrupar-se em cinco categorias:

1. **Caracteres alfabéticos** (letras maiúsculas e minúsculas, em uma primeira versão do abecedário inglês).

   A, B, C, D, E ... X, Y, Z, a, b, c ... X, Y, Z

2. **Caracteres numéricos** (dígitos do sistema de numeração).

   0, 1, 2, 3, 4, 5, 6, 7, 8, 9 *sistema decimal*

3. **Caracteres especiais** (símbolos ortográficos e matemáticos não incluídos nos grupos anteriores).

   { } Ñ ñ ! ? & > # ç ...

4. **Caracteres geométricos e gráficos** (símbolos ou módulos com os quais podem ser representados quadros, figuras geométricas, ícones etc.).

5. **Caracteres de controle** (representam ordens de controle como o caractere para passar para a linha seguinte [NL] ou para ir para o começo de uma linha [**RC**, *retorno de carro*, *carriage return*, **CR**], emitir um apito no final [BEL] etc.).

Ao introduzir um texto em um computador, por meio de um periférico, os caracteres são codificados segundo o **código de entrada/saída** de modo que a cada caractere se associe determinada combinação de *n* bits.

Hoje, os códigos mais atualizados são: **EBCDIC, ASCII** e **Unicode**.

- **Código EBCDIC** (*Extended Binary Coded Decimal Inter Change Code*).
  Esse código utiliza $n = 8$ bits de forma que se possa codificar até $m = 2^8 = 256$ símbolos diferentes. Este foi o primeiro código usado para computadores, aceito em princípio pela IBM.
- **Código ASCII** (*American Standard Code for Information Interchange*).
  O código ASCII básico utiliza 7 bits e permite representar 128 caracteres (letras maiúsculas e minúsculas do alfabeto inglês, símbolos de pontuação, dígitos 0 a 9 e certos controles de informação, tais como retorno do carro, mudança de linha, tabulações etc.). Esse código é o mais usado em computadores, ainda que ASCII ampliado com 8 bits possa chegar a 2^8 (256) caracteres distintos, entre eles símbolos e caracteres especiais de outros idiomas como o espanhol.
- **Código Unicode**
  Embora o ASCII tenha sido e continue a ser dominante nos caracteres que se lêem como referência, atualmente, é necessária a representação da informação em muitas outras línguas, como o português, espanhol, chinês, árabe etc. Esse código utiliza um padrão único de 16 bits para representar cada símbolo, o que permite 2^{16} bits, ou seja, até 65.536 padrões de bits (símbolos) diferentes.

Do ponto de vista de unidade de armazenamento de caracteres, utiliza-se o arquivo. Um **arquivo** consta de uma seqüência de símbolos de determinado comprimento codificado utilizando ASCII ou Unicode e que é

denominado **arquivo de texto**. É importante diferenciar arquivos de textos simples, que são manipulados pelos programas de utilidade denominados **editores de texto**, dos arquivos de textos mais elaborados que são produzidos pelos processadores de texto tipo Microsoft Word. Ambos são compostos por caracteres de texto, porém, enquanto o obtido com o editor de texto é um arquivo de texto puro que codifica caractere a caractere, o arquivo de texto produzido por um processador de textos contém números, códigos que representam mudanças de formato, de tipos de fontes de letra e outros e, inclusive, podem utilizar códigos proprietários distintos de ASCII ou Unicode.

1.3.2 Representação de valores numéricos

O armazenamento de informação como caracteres codificados é ineficiente quando a informação é registrada como numérica pura. Vejamos essa situação com a codificação do número 65; se o armazenamento foi efetuado como caracteres ASCII utilizando um byte por símbolo, é necessário um total de 16 bits, de modo que o maior número que se podia armazenar em 16 bits (dois bytes) seria 99. Entretanto, se utilizamos *notação binária* para armazenar inteiros, o intervalo pode ir de 0 a 65.535 ($2^{16} - 1$) para números de 16 bits. Conseqüentemente, a notação binária (ou suas variantes) é a mais utilizada para o armazenamento de dados numéricos codificados.

A solução adotada para a representação de dados numéricos é a seguinte: ao introduzir um número no computador, este é codificado e armazenado como um texto ou cadeia de caracteres, entretanto, dentro do programa, para cada dado é enviado um tipo de dado específico, e é tarefa do programador associar cada dado ao tipo adequado correspondente às tarefas e operações que serão realizadas com ele.

O método prático realizado pelo computador é que, uma vez definidos os dados numéricos de um programa, uma rotina (função interna) da biblioteca do compilador (tradutor) da linguagem de programação se encarrega de transformar a cadeia de caracteres que representa o número em sua notação binária.

Há duas formas de representar os dados numéricos: números inteiros e números reais.

Representação de inteiros

Os dados do tipo inteiro são representados no interior do computador por notação binária. A memória ocupada pelos tipos inteiros depende do sistema, mas normalmente são dois bytes (nas versões de MS-DOS e versões antigas de Windows) e quatro bytes (nos sistemas de 32 bits como Windows ou Linux). Por exemplo, um inteiro armazenado em 2 bytes (16 bits):

```
1000  1110  0101  1011
```

Os inteiros podem ser representados com sinal (`signed`, em C++) ou sem sinal (`unsigned`, em C++), ou seja, números positivos ou negativos. Geralmente, utiliza-se um bit para o sinal. Os inteiros sem sinal por não terem sinais podem conter valores positivos maiores. Normalmente, se um inteiro não é especificado «com/sem sinal», costuma-se atribuir sinal por *default* ou omissão.

O intervalo de possíveis valores de inteiros depende do tamanho em bytes ocupado pelos números e é representado com sinal ou sem sinal (a Tabela 1.3 resume características de tipos-padrão em C++).

Representação de reais

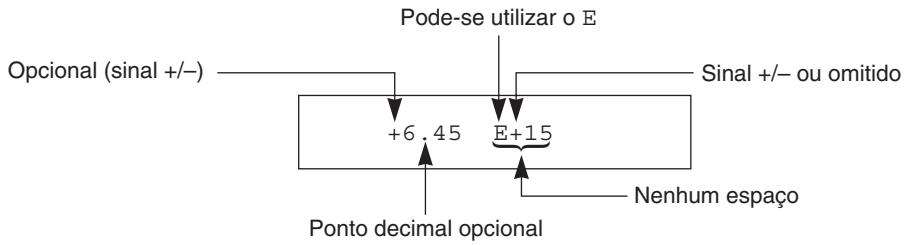
Números reais são aqueles que possuem uma parte decimal como `2,6` e `3,14152`. Os reais são representados por *notação científica* ou por *ponto flutuante*; por essa razão, nas linguagens de programação, como C++, são conhecidos como números em ponto flutuante.

Existem duas formas de representar os números reais. A primeira é utilizada com a notação do ponto decimal.

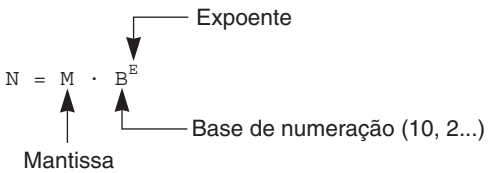
Exemplos

```
12.35   99901.32   0.00025   9.0
```

A segunda forma para representar números com ponto flutuante é a notação científica ou exponencial, também conhecida como notação `E.` Essa notação é muito útil para representar números muito grandes ou muito pequenos.



Notação exponencial



Exemplos

```
2.52   e + 8
8.34  E – 4        equivale a          8.34/10^4  =  0.000834
7E5
–18.35e15
5.95E25
9.11e – 31         equivale a     0.0000000000000000000000000000911
```

Representação de caracteres

Um documento de texto é escrito utilizando-se um conjunto de caracteres adequado ao tipo de documento. Nas linguagens de programação, são usados, principalmente, dois códigos de caracteres. O mais comum é **ASCII** (American Standard Code for Information Interchange) e algumas linguagens, como Java, utilizam **Unicode** (*www.unicode.org*). Ambos os códigos se baseiam na atribuição de um código numérico a cada um dos tipos de caracteres do código.

Em C++, os *caracteres* são processados normalmente utilizando o tipo `char`, que associa cada caractere a um código numérico que é armazenado em um byte.

O código ASCII básico que usa 7 bits (128 caracteres distintos) e o ASCII *ampliado* a 8 bits (256 caracteres distintos) são os códigos mais utilizados. Assim, podem ser representados caracteres, tais como '`A`', '`B`', '`c`', '`$`', '`4`', '`5`' etc.

No caso de C++, considera-se dado de tipo caractere denominado *`wchar_t`* que serve para representar caracteres de tipo internacional para manejar linguagens estrangeiras.

A Tabela 1.3 abrange os tipos inteiros, reais e caracteres utilizados em C++, a memória utilizada (número de bytes ocupados pelo dado) e o intervalo de números.

1.3.3 Representação de imagens

As imagens são adquiridas por meio de periféricos especializados tais como *scanners*, câmeras digitais de vídeo, câmeras fotográficas etc. Uma imagem, como outros tipos de informação, é representada por padrões de bits, gerados pelo periférico correspondente. Existem dois métodos básicos para representar imagens: *mapas de bits* e *mapas de vetores*.

Tabela 1.3 Tipos inteiros reais, em C++

		Caractere e bool
Tipo	**Tamanho**	**Intervalo**
`short (short int)`	2 bytes	`–32.738..32.767`
`int`	4 bytes	`–2.147.483.648` a `2.147.483.647`
`long (long int)`	4 bytes	`–2.147.483.648` a `2.147.483.647`
`float (real)`	4 bytes	10^{-38} a 10^{38} (*aproximadamente*)
`double`	8 bytes	10^{-308} a 10^{308} (*aproximadamente*)
`long double`	10 bytes	10^{-4932} a 10^{4932} (*aproximadamente*)
`char (caractere)`	1 byte	Todos os caracteres ASCII
`bool`	1 byte	`True` (verdadeiro) e `false` (falso)

Nas técnicas de *mapas de bits*, uma imagem é considerada uma coleção de pontos, em que cada um dos pontos se denomina *pixel* (abreviação de *picture element*). Uma imagem em branco-e-preto é representada como uma extensa cadeia de bits que representam as filas de pixels na imagem, na qual cada bit é igual a 1 ou igual a 0, dependendo de o pixel correspondente ser branco ou preto. No caso de imagens coloridas, cada pixel é representado por uma combinação de bits que indicam a cor dos pixels. Quando são utilizadas técnicas de mapas de bits, o padrão de bits resultante se chama *mapa de bits*, significando que o padrão de bits resultante que representa a imagem é pouco mais que um mapa da imagem.

Muitos periféricos de computador – tais como câmeras de vídeo, *scanners* etc. – convertem imagens coloridas em formato de mapa de bits. Os formatos mais utilizados na representação de imagens são mostrados na Tabela 1.4.

Tabela 1.4 Mapa de bits

Formato	**Origem e descrição**
`BMP`	**Microsoft**. Formato simples com imagens de grande qualidade, mas com o inconveniente de ocupar muito (o que não é útil para a Web).
`JPEG`	Grupo **JPEG**. Qualidade aceitável para imagens naturais. Inclui compressão. É utilizado na Web.
`GIF`	**CompuServe**. Muito adequado para imagens não naturais (logotipos, marcadores, desenhos aninhados...) Muito utilizado na Web.

Mapas de vetores. Outros métodos de representar uma imagem se fundamentam em decompor a imagem em uma coleção de objetos, como linhas, polígonos e textos com seus respectivos atributos ou detalhes (espessura, cor etc.) ver Tabela 1.5.

Tabela 1.5 Mapas de vetores

Formato	**Descrição**
`IGES`	ASME/ANSI. Padrão para troca de dados e modelos de (AutoCad...)
`Pict`	Apple Computer. Imagens vetoriais.
`EPS`	Adobe Computer.
`TrueType`	Apple e Microsoft para EPS.

1.3.4 Representação de sons

A representação de sons adquiriu notável importância em razão essencialmente da infinidade de aplicações multimídia tanto particular quanto na Web.

O método mais genérico de codificação da informação de áudio para armazenamento e manipulação em computador é mostrar a amplitude da onda de som em intervalos regulares e registrar as séries de valores obtidos. O sinal de som é captado por meio de microfones ou dispositivos similares e produz um sinal analógico que pode

receber qualquer valor dentro de um intervalo contínuo determinado. Em um intervalo de tempo contínuo, dispõe-se de infinitos valores do sinal analógico, que é necessário armazenar e processar, para o qual se recorre a uma *técnica de amostragem*. As amostras obtidas são digitalizadas com um conversor analógico-digital, de modo que o sinal de som seja representado por seqüências de bits (por exemplo, 8 ou 16) para cada amostra. Essa técnica é similar àquela historicamente utilizada pelas comunicações telefônicas de longa distância. Naturalmente, dependendo da qualidade do som que se queira, serão necessários mais números de bits por amostra, freqüências de amostragem mais alta e logicamente mais amostragens por período.[3]

Como dados de referência, podemos considerar que, para obter a reprodução de qualidade de som de alta fidelidade para um disco CD de música, se costuma utilizar, pelo menos, uma freqüência de amostragem de 44.000 amostras por segundo. Os dados obtidos em cada amostra são codificados em 16 bits (32 bits para gravações em estéreo). Como dado anedótico, cada segundo de música gravada em estéreo necessita mais de um milhão de bits.

Um sistema de codificação de música bastante utilizado em sintetizadores musicais é MIDI (**M**usical **I**nstruments **D**igital **I**nterface), encontrado em sintetizadores de música para sons de *videogames*, sites da Web, teclado eletrônicos etc.

1.4 CONCEITO DE ALGORITMO

O objetivo fundamental deste texto é ensinar a resolver problemas por meio de um computador. O programador de computador é acima de tudo uma pessoa que resolve problemas, por isso para chegar a ser um programador eficiente, é necessário aprender a resolver problemas de um modo rigoroso e sistemático. Neste livro, iremos nos referir à *metodologia necessária para resolver problemas por meio de programas*, conceito que é denominado **metodologia da programação**. O eixo central dessa metodologia é o conceito, já tratado, de algoritmo.

Um algoritmo é um método para resolver um problema. Ainda que a popularização do termo tenha chegado com o advento da era da informática, **algoritmo** provém de *Mohammed al-Khowârizmi*, matemático persa que viveu durante o século IX e alcançou grande reputação pelo enunciado das regras passo a passo para somar, subtrair, multiplicar e dividir números decimais. A tradução para o latim do seu sobrenome na palavra *algorismus* derivou posteriormente em algoritmo. Euclides, o grande matemático grego (século IV a.C.) que inventou o método para encontrar o máximo divisor comum de dois números, é considerado, juntamente com Al-Khowârizmi, outro grande pai da algoritmia (ciência que trata dos algoritmos).

O professor Niklaus Wirth – inventor de Pascal, Modula-2 e Oberon – titulou um dos seus mais famosos livros, *Algoritmos + Estruturas de dados = Programas*, mostrando-nos que só é possível chegar a realizar um bom programa com o projeto de um algoritmo e uma correta estrutura de dados. Essa equação será uma das hipóteses fundamentais consideradas nesta obra.

A resolução de um problema exige o projeto de um algoritmo que resolva o problema proposto.

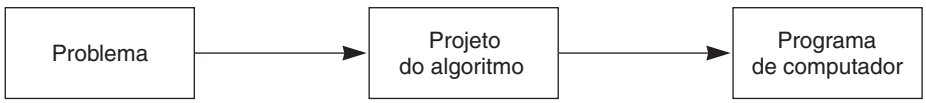


Figura 1.10 Resolução de um problema.

Os passos para a resolução de um problema são:

1. *Projeto do algoritmo*, que descreve a seqüência ordenada de passos – sem ambigüidades – que conduz à solução de um problema dado. (*Análise do problema e desenvolvimento do algoritmo*.)
2. Expressar o algoritmo como um *programa* em uma linguagem de programação adequada. (*Fase de codificação*.)
3. *Execução e validação* do programa pelo computador.

Para chegar à realização de um programa, é necessário o projeto prévio de um algoritmo, de modo que sem algoritmo não pode existir um programa.

---

[3] Nas obras do professor Alberto Prieto, Schaum, *Conceitos de informática e introdução à Informática,* publicadas pela McGraw-Hill, pode ser encontrada excelente referência sobre esses conceitos e outros complementares deste capítulo introdutório.

Os algoritmos são independentes tanto da linguagem de programação em que se expressam como do computador que os executa. Em cada problema, o algoritmo pode se expressar em uma linguagem de programação diferente e ser executado em distintos computadores; no entanto, o algoritmo será sempre o mesmo. Assim, por exemplo, em uma analogia com a vida diária, uma receita de um prato culinário pode estar em espanhol, inglês ou francês, mas em qualquer que seja o idioma, os passos para a elaboração do prato serão realizados sem importar o idioma do cozinheiro.

Na ciência da computação e na programação, os algoritmos são mais importantes que as linguagens de programação ou que o computador. Uma linguagem de programação é somente um meio para expressar um algoritmo e um computador é só um processador para executá-lo. Tanto a linguagem de programação quanto o computador são os meios para se obter um fim: conseguir que o algoritmo seja executado e que seja efetuado o processamento correspondente.

Dada a importância do algoritmo na ciência da computação, um aspecto muito importante será o *projeto de algoritmos*. Grande parte deste livro se dedica ao ensino e prática dessa tarefa denominada *algorítmica*.

O projeto da maioria dos algoritmos requer criatividade e profundos conhecimentos da técnica de programação. Essencialmente, *a solução de um problema pode ser expressa por meio de um algoritmo*.

1.4.1 Características dos algoritmos

As características fundamentais que todo algoritmo deve cumprir são:

- Um algoritmo deve ser *preciso* e indicar a ordem de realização de cada passo.
- Um algoritmo deve estar *definido*. Se se segue um algoritmo duas vezes, o resultado obtido deve ser o mesmo cada vez.
- Um algoritmo deve ser *finito*. Se se segue um algoritmo, a ação deve terminar em algum momento, ou seja, deve ter um número finito de passos.

A definição de um algoritmo deve descrever três partes: *Entrada*, *Processamento* e *Saída*. No algoritmo da receita culinária, citada anteriormente, teremos:

Entrada: ingredientes e utensílios utilizados.
Processamento: elaboração da receita culinária.
Saída: finalização do prato (por exemplo, cordeiro).

Exemplo 1.1

Um cliente faz um pedido a uma fábrica. Esta examina em seu banco de dados a ficha do cliente; se o cliente é idôneo, a fábrica aceita o pedido; caso contrário, recusará o pedido. Redigir o algoritmo correspondente.

Os passos do algoritmo são:

```
1. Início.
2. Ler o pedido.
3. Examinar a ficha do cliente.
4. Se o cliente é idôneo, aceitar pedido; caso contrário, recusar
   pedido.
5. Fim.
```

Exemplo 1.2

Se queremos projetar um algoritmo para saber se um número é primo ou não.

Um número é primo somente se puder ser dividido por ele mesmo e pela unidade (isto é, não tem mais divisores que ele mesmo e a unidade). Por exemplo, 9, 8, 6, 4, 12, 16, 20 etc. não são primos, já que são divisíveis por números diferentes deles mesmos e da unidade. Assim, 9 é divisível por 3, 8 é divisível por 2 etc.

O algoritmo de resolução do problema passa por dividir sucessivamente o número por 2, 3, 4 etc.

```
1. Início
2. Colocar X igual a 2 (x = 2, onde x é a variável que representa os divisores do
   número que se busca N).
3. Dividir N por X (N/X).
4. Se o resultado de N/X é inteiro, então N é um número primo e ir para o ponto 7;
   caso contrário, continuar o processamento.
5. Somar 1 a X (X ← X + 1).
6. Se X é igual a N, então N é um número primo; caso contrário, voltar ao ponto 3.
7. Fim.
```

Por exemplo, se `N` é igual a `131`, os passos anteriores são:

```
1. Início.
2. X = 2.
3 e 4. 131/X. Como o resultado não é inteiro, continua-se o processamento.
5. X ← 2 + 1, logo X = 3.
6. Como X não é 131, ir para o ponto 3.
3 e 4. 131/X resultado não é inteiro.
5. X ← 3 + 1, X = 4.
7. Como X não é 131 ir para o ponto 3.
3 e 4. 131/X... etc.
8. Fim.
```

Exemplo 1.3

Realizar a soma de todos os números pares entre 2 e 1.000.

O problema consiste em somar 2 + 4 + 6 + 8 ... + 1.000. Utilizaremos a palavras `SOMA` e `NÚMERO` (*variáveis* serão denominadas mais tarde) para representar as somas sucessivas (2+4), (2+4+6), (2+4+6+8) etc. A solução pode ser escrita com o seguinte algoritmo:

```
1. Início.
2. estabelecer SOMA a 0.
3. estabelecer NÚMERO a 2.
4. Somar NÚMERO e SOMA. O resultado será o novo valor da soma (SOMA).
5. Aumentar NÚMERO em 2 unidades.
6. Se NÚMERO =< 1.000 ir para o passo 4;
7. caso contrário, escrever o último valor de SOMA e terminar o processamento.
8. Fim.
```

1.5 PROGRAMAÇÃO ESTRUTURADA

A programação orientada a objetos (POO) foi desenvolvida para atenuar diversas limitações que se encontravam em enfoques anteriores de programação. Para apreciar as vantagens da POO, é necessário constatar as limitações citadas e como são produzidas pelas linguagens de programação tradicionais.

C, Pascal, FORTRAN e linguagens similares são conhecidas como *linguagens procedimentais* (por procedimentos), quer dizer, cada sentença ou instrução indica ao compilador para que realize alguma tarefa: obter uma entrada, produzir uma saída, somar três números, dividir por cinco etc. Em resumo, um programa em uma

linguagem procedimental é um conjunto de instruções ou sentenças. No caso de pequenos programas, esses princípios de organização (denominados *paradigma*) se mostram eficientes. O programador somente precisa criar essa lista de instruções em uma linguagem de programação, compilar no computador e este, por sua vez, executar essas instruções.

Quando os programas se tornam maiores, coisa que logicamente acontece quando aumenta a complexidade do problema a ser resolvido, a lista de instruções aumenta consideravelmente, de tal modo que o programador tem muitas dificuldades para controlar esse grande número de instruções. Os programadores podem controlar, de modo normal, uma centena de linhas de instruções. Para resolver esse problema, os programas foram decompostos em unidades menores que adotaram o nome de *funções* (*procedimentos*, *subprogramas* ou *subrotinas* em outras linguagens de programação). Desse modo, um programa orientado a procedimentos se divide em funções, cada função tem um propósito bem-definido e resolve uma tarefa concreta e se projeta uma interface claramente definida (o protótipo ou cabeçalho da função) para sua comunicação com outras funções.

Com o passar dos anos, a idéia de separar um programa em funções foi evoluindo e se chegou ao agrupamento das funções em outras unidades maiores conhecidas como *módulos* (normalmente, no caso de C, chamadas **arquivos**); no entanto, o princípio seguia o mesmo: agrupar componentes que executam listas de instruções (sentenças). Essa característica faz que à medida que os programas se tornem maiores e mais complexos, o paradigma estruturado comece a dar sinais de debilidade, tornando-se muito difícil terminar o programa com eficiência. Existem várias razões para a debilidade dos programas estruturados para resolver problemas complexos e talvez duas das mais evidentes sejam estas: primeiro, as funções têm acesso ilimitado aos dados globais; segundo, as funções inconexas e dados, e fundamentos do paradigma procedimental proporcionam um modelo pobre do mundo real.

1.5.1 Dados locais e dados globais

Em um programa procedimental, por exemplo, escrito em C, existem dois tipos de dados. *Dados locais* que estão ocultos no interior da função e são utilizados, exclusivamente, pela função. Esses dados locais estão estritamente relacionados com suas funções e protegidos de modificações por outras funções.

Outro tipo de dado são os *dados globais* aos quais se pode acessar de *qualquer* função do programa, isto é, duas ou mais funções podem acessar os mesmos dados sempre que estes sejam globais. Na Figura 1.11 mostra-se a disposição de variáveis locais e globais em um programa procedimental.

Um programa grande (Figura 1.12) é composto por numerosas funções e dados globais e comporta um grande número de conexões entre funções e dados que dificultam sua compreensão e leitura.

Todas essas múltiplas conexões originam diferentes problemas. Em primeiro lugar, tornam difícil conceituar a estrutura do programa. Em segundo lugar, é difícil modificar o programa já que mudanças em dados globais podem exigir a reescrita de todas as funções que os acessam. Também pode acontecer que essas modificações dos dados globais não sejam aceitas por todas ou algumas das funções.

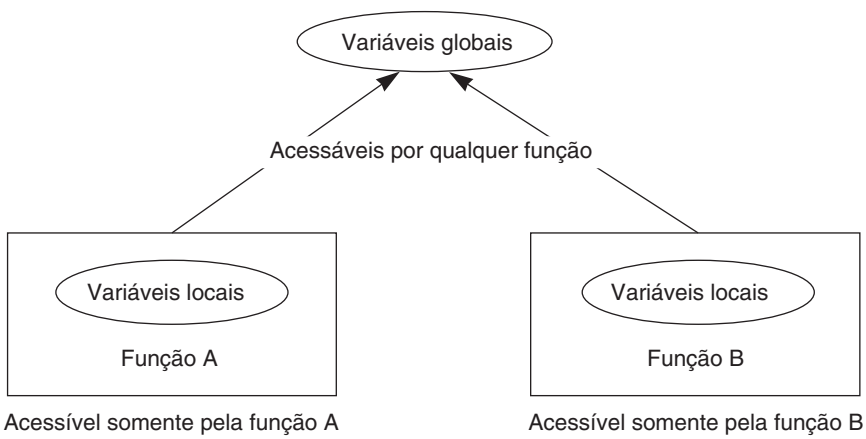


Figura 1.11 Dados locais e globais.

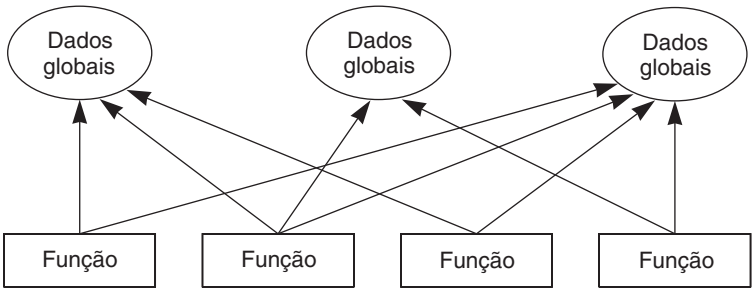


Figura 1.12 Um programa procedimental.

1.5.2 Modelagem do mundo real

Um segundo problema importante da programação estruturada consiste no fato de que a disposição separada de dados e funções não corresponde aos modelos das coisas do mundo real. No mundo físico, tratamos com objetos físicos, como pessoas, carros ou aviões. Esses objetos não são como os dados nem como as funções. Os objetos complexos ou não do mundo real têm *atributos* e *comportamento*.

Os **atributos** ou características dos objetos são, por exemplo: nas pessoas, idade, profissão, endereço etc.; em um carro, potência, número de matrícula, preço, número de portas etc.; em uma casa, aparência, preço, ano de construção, localização, etc. Na realidade, os atributos do mundo real têm o seu equivalente nos dados de um programa; possuem valor específico, tal como 200 metros quadrados, 20 mil reais, cinco portas etc.

O **comportamento** é uma ação que os objetos do mundo real executam como resposta a determinado estímulo. Se você pisa no freio de um carro, ele pára; se acelera, o carro aumenta sua velocidade etc., o comportamento é, em essência, como uma função: chama-se uma função para realizar algo (visualizar a relação de empregados de uma empresa).

Por essas razões, nem os dados nem as funções, por si mesmos, modelam os objetos do mundo real de forma eficiente.

> A programação estruturada melhora a clareza, confiabilidade e facilidade da manutenção dos programas, entretanto, para programas maiores ou de grande escala, apresentam desafios de difícil solução.

1.6 PROGRAMAÇÃO ORIENTADA A OBJETOS

A programação orientada a objetos (POO), talvez o paradigma de programação mais utilizado no mundo do desenvolvimento de software e da engenharia de software do século XXI, traz um novo enfoque aos desafios que se apresentam na programação estruturada quando os problemas a serem resolvidos são complexos. Ao contrário da programação *procedimental* que enfatiza os algoritmos, a POO enfatiza os dados. Em vez de tentar ajustar um problema ao enfoque *procedimental* de uma linguagem, a POO tenta ajustar a linguagem ao problema. A idéia é projetar formatos de dados que correspondam às principais características de um problema.

A idéia fundamental das linguagens orientadas a objetos é combinar, em uma única unidade ou módulo, tanto os dados como as funções que operam sobre esses dados. Tal unidade denomina-se um **objeto**.

As funções de um objeto são designadas por *funções-membro* em C++ ou *métodos* (este é o caso de Smalltalk, uma das primeiras linguagens orientadas a objetos) e são o único meio para acessar seus dados. Os dados de um objeto são também conhecidos como *atributos* ou *variáveis de instância*. Caso pretenda ler os dados de um objeto, chama-se a função membro do objeto, acessa-se os dados e devolve-se um valor. Não é possível acessar diretamente os dados. Os dados estão ocultos, de modo que estão protegidos de alterações acidentais. Diz-se que os dados e as funções estão *encapsulados em uma única entidade*. *Encapsulamento* e *ocultação* de dados são termos-chave na descrição de linguagens orientadas a objetos.

Para modificar os dados de um objeto, é necessário saber exatamente quais são as funções que interagem com as funções-membro do objeto. Nenhuma outra função pode acessar os dados. Isso simplifica a escrita, depuração e manutenção do programa. Um programa C++ é composto normalmente por um número de objetos que se

comunicam uns com os outros por meio da chamada a outras funções-membro. A organização de um programa em C++ é mostrada na Figura 1.13. A chamada a uma função-membro de um objeto é denominada *enviar uma mensagem* a outro objeto.

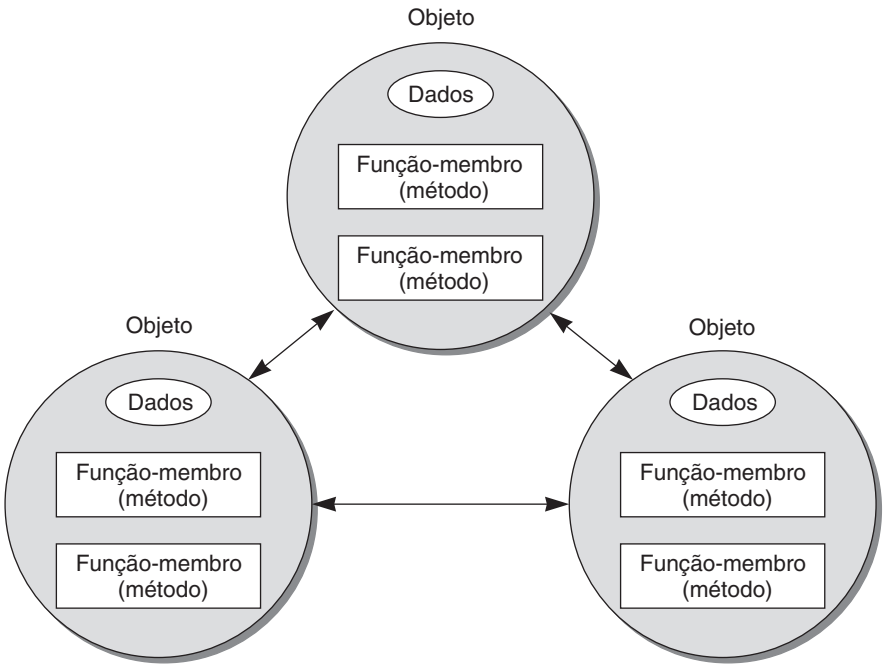


Figura 1.13 Organização típica de um programa orientado a objetos.

No paradigma orientado a objetos, o programa é organizado como um conjunto finito de objetos que contenha dados e operações (funções-membro em C++) que chamam a esses dados e que se comunicam entre si por meio de mensagens.

1.6.1 Propriedades fundamentais da orientação a objetos

Existem diversas características relacionadas à orientação a objetos. Todas as propriedades consideradas não são exclusivas desse paradigma, já que podem existir em outros paradigmas, mas em seu conjunto definem claramente as linguagens orientadas a objetos. Essas propriedades são:

- **Abstração (tipos abstratos de dados e classes).**
- **Encapsulamento de dados.**
- **Ocultação de dados.**
- **Herança.**
- **Polimorfismo.**

C++ suporta todas as características anteriores que definem a orientação a objetos, ainda que haja numerosas discussões em torno da consideração de C++ como linguagem orientada a objetos. A razão é que, em contraste com linguagens, tais como Smalltalk, Java ou C#, C++ não é uma linguagem puramente orientada a objetos. C++ suporta orientação a objetos, porém é compatível com C e permite que programas em C++ sejam escritos sem utilizar características orientadas a objetos. De fato, C++ é uma linguagem *multiparadigma* que permite programação estruturada, *procedimental*, orientada a objetos e genérica.

1.6.2 Abstração

A abstração é uma propriedade que consiste em levar em conta somente os aspectos importantes de determinado ponto de vista e não levar em consideração os aspectos restantes. O termo **abstração** que é utilizado em programação se refere ao fato de diferenciar entre as propriedades externas de uma entidade e os detalhes da sua composição interna. É a abstração o que permite ignorar os detalhes internos de um dispositivo complexo, tal como um computador, um automóvel, uma lavadora ou um forno de microondas etc., e utilizá-lo como uma única entidade compreensível. Por meio da abstração, em primeiro lugar, são projetados e fabricados esses complexos sistemas e posteriormente os componentes menores que o compõem. Cada componente representa um nível de abstração no qual o uso do componente se isola dos detalhes da composição interna do componente. A abstração possui diversos graus denominados níveis de abstração.

Em conseqüência, a abstração possui diversos graus de complexidade que são denominados *níveis de abstração* que ajudam a estruturar a complexidade intrínseca que possuem os sistemas do mundo real. No modelo orientado a objetos de um sistema, isso significa concentrar-se no *que* é e *no que* faz um objeto e não em *como* se deve implementar. É durante o processo de abstração que se decide quais características e comportamento deve ter o modelo.

Aplicando a abstração, pode-se construir, analisar e gerenciar sistemas de computadores grandes e complexos que não poderiam ser projetados, caso modelasse a um nível detalhado. Em cada nível de abstração, é visualizado o sistema em termos de componentes, denominados **ferramentas abstratas**, cuja composição interna se ignora. Isso nos permite nos concentrarmos em como cada componente interage com outros componentes e nos centrarmos na parte do sistema que é mais relevante para a tarefa a realizar em lugar de nos perdermos em nível de detalhes menos significativos.

Em estruturas ou registros, as propriedades individuais dos objetos podem ser armazenadas nos membros. Para os objetos, é de interesse não só *como* estão organizados como também *o que* se pode fazer com eles, isto é, as operações que formam o interior de um objeto também são importantes. O primeiro conceito no mundo da orientação a objetos nasceu com os tipos abstratos de dados (TAD). Um tipo abstrato de dados descreve não somente os atributos de um objeto, como também seu comportamento (as operações). Isso pode também incluir uma descrição dos estados que pode alcançar um objeto.

A abstração é um meio de reduzir a complexidade. As características e os processamentos se reduzem às propriedades essenciais, e são resumidas ou combinadas entre si. Desse modo, as características complexas se tornam mais manejáveis.

Exemplo 1.4

Diferentes modelos de abstração do termo carro.

- Um `carro` é a combinação (ou composição) de diferentes partes, como motor, carroceria, quatro rodas, cinco portas etc.
- Um `carro` é um conceito comum para diferentes tipos de carro. Pode ser classificado pelo nome do fabricante (Audi, BMW, SEAT, Toyota, Chrisler...), por sua categoria (turismo, esportivo, *off-road*...), pelo combustível que utiliza (gasolina, óleo disel, gás, híbrido...).

A abstração `carro` será utilizada sempre que a marca, a categoria ou o combustível não forem significativos. Assim, um `carro` será utilizado para transportar pessoas ou ir de um lugar a outro.

1.6.3 Encapsulamento e ocultação de dados

O *encapsulamento de dados* é o processamento de agrupar dados e operações relacionadas na mesma unidade de programação. Os objetos que possuem as mesmas características e comportamento são agrupados em classes, que nada mais são que unidades ou módulos de programação que encapsulam dados e operações.

A ocultação de dados permite separar o aspecto de um componente, definido por sua *interface* com o exterior, dos seus detalhes internos de implementação. As expressões ocultação de informação (*information hiding*) e encapsulamento de dados (*data encapsulation*) são utilizadas como sinônimos, mas nem sempre é assim, são expressões similares mas distintas. Em C++, não é o mesmo, já os dados internos estão protegidos do exterior e não

podemos acessá-los a não ser do seu próprio interior e, portanto, não estão ocultos. O acesso ao objeto está restrito só por meio de uma interface bem-definida.

O projeto de um programa orientado a objetos contém, pelo menos, os seguintes passos:

1. Identificar os *objetos* do sistema.
2. Agrupar em *classes* todos os objetos que tenham características e comportamento comuns.
3. Identificar os *dados* e *operações* de cada uma das classes.
4. Identificar as *relações* que podem existir entre as classes.

Em C++, um **objeto** é um elemento individual com sua própria identidade; por exemplo, um livro, um automóvel... Uma **classe** pode descrever as propriedades genéricas de um executivo de uma empresa (nome, título, salário, cargo...) enquanto um objeto representará um executivo específico (Luis Mackoy, diretor geral). Geralmente, uma classe define quais dados são utilizados para representar um objeto e as operações que podem ser executadas sobre esses dados.

Cada classe tem suas próprias características e comportamento; em geral, uma classe define os dados utilizados e as operações executadas sobre esses dados. Uma classe define um objeto. No sentido estrito de programação, uma classe é um tipo de dados. Diferentes variáveis desse tipo podem ser criadas. Em programação orientada a objetos, estas chamam-se *instâncias*. As instâncias são, por conseqüência, a realização dos objetos descritos em uma classe. Essas instâncias são formadas por dados ou atributos descritos na classe e podem ser manipuladas pelas operações definidas dentro delas.

Os termos *objeto* e *instância* são freqüentemente utilizados como sinônimos (especialmente em C++). Se uma variável do tipo `Carro` é declarada, é criado um objeto `Carro` (uma instância da classe `Carro`).

As operações definidas nos objetos são conhecidas como *métodos*. Cada operação chamada por um objeto é interpretada como uma *mensagem* para o objeto, que utiliza um método específico para processar a operação.

No projeto de programas orientados a objetos, é realizado em primeiro lugar o projeto das classes que representam com precisão as coisas das quais trata o programa. Por exemplo, um programa de projeto pode definir classes que representam retângulos, linhas, pincéis, cores etc. As definições de classes incluem uma descrição das operações admissíveis para cada classe, tais como deslocamento de um círculo ou rotação de uma linha. Depois, prossegue-se o projeto de um programa utilizando objetos das classes.

O projeto de classes viáveis e úteis pode ser uma difícil tarefa. Afortunadamente, as linguagens POO facilitam essa tarefa, já que incorporam classes existentes em sua própria programação. Os fabricantes de software proporcionam numerosas bibliotecas de classes, inclusive aquelas projetadas para simplificar a criação de programas para ambientes, tais como Windows, Linux, Macintosh ou Unix. Um dos benefícios reais de C++ é que ele permite a reutilização e adaptação de códigos existentes e já testados e depurados.

1.6.4 Objetos

O objeto é o centro da programação orientada a objetos. Um objeto é algo que se visualiza, se utiliza e assume um papel. Quando se programa com um enfoque orientado a objetos, tenta-se descobrir e implementar os objetos que assumem um papel no domínio do problema e, conseqüentemente, do programa. A estrutura interna e o comportamento de um objeto não têm prioridade na primeira fase. É importante que um objeto, por exemplo, um carro ou uma casa, assuma um papel.

Dependendo do problema, os diferentes aspectos de um ponto de vista, são relevantes. Um carro pode ser montado com diferentes partes, tais como motor, carroceria, portas, ou pode ser descrito utilizando-se propriedades, como sua velocidade, quilometragem ou seu fabricante. Esses atributos indicam o objeto. De modo semelhante, uma pessoa também pode ser vista como um objeto do qual se dispõem diferentes atributos. Dependendo da definição do problema, esses atributos podem ser nome, sobrenome, endereço, número de telefone, cor dos cabelos, altura, peso, profissão etc.

Um objeto não tem necessariamente que realizar algo tangível ou concreto. Pode ser totalmente abstrato e também pode descrever um processamento. Uma partida de basquete ou de *rugby*, por exemplo, pode ser descrita como um objeto. Os atributos desse objeto podem ser os jogadores, o treinador, a pontuação e o tempo transcorrido da partida.

Quando se trata de resolver um problema com orientação a objetos, o problema não se descompõe em funções como ocorre na programação estruturada tradicional, caso de C, mas sim em objetos. Pensar em termos de objetos tem grande vantagem: os objetos do problema são associados aos objetos do mundo real.

Que tipos de coisas são objetos nos programas orientados a objetos? A resposta é limitada por sua imaginação, ainda que possam se agrupar em categorias típicas que facilitarão sua busca na definição do problema de um modo mais rápido e simples.

- Recursos humanos:
  - Empregados.
  - Estudantes.
  - Clientes.
  - Vendedores.
  - Sócios.

- Coleção de dados:
  - Arrays.
  - Listas.
  - Pilhas.
  - Árvores.
  - Árvores binárias.
  - Grafos.

- Tipos de dados definidos pelos usuários:
  - Hora.
  - Números complexos.
  - Pontos do plano.
  - Pontos do espaço.
  - Ângulos.
  - Lados.

- Elementos do computador:
  - Menus.
  - Janelas.
  - Objetos gráficos (retângulos, círculos, retas, pontos...).
  - Mouse.
  - Teclado.
  - Impressora.
  - USB.
  - Cartões de memória de câmeras fotográficas.

- Objetos físicos:
  - Carros.
  - Aviões.
  - Trens.
  - Barcos.
  - Motocicletas.
  - Casas.

- Componentes de um *videogame*:
  - Console.
  - Controle remoto.
  - Volante.
  - Conectores.
  - Memória.
  - Acesso a Internet.

A correspondência entre objetos de programação e objetos do mundo real é o resultado eficiente de combinar dados e funções que manipulam esses dados. Os objetos resultantes oferecem melhor solução ao desenvolvimento do programa no caso das linguagens orientadas a procedimentos.

Um **objeto** pode ser definido do ponto de vista conceitual como uma entidade individual de um sistema que é caracterizado por um estado e um comportamento. Do ponto de vista de implementação, um **objeto** é uma entidade que possui um conjunto de *dados* e um conjunto de *operações* (*funções* ou *métodos*).

O estado de um objeto é determinado pelos valores que tomam seus dados, valores estes que podem ter restrições impostas na definição do problema. Os dados também são denominados *atributos* e compõem a estrutura do objeto e as operações – também chamadas *métodos* – representam os serviços que proporcionam o objeto.

A representação gráfica de um objeto em **UML** (Linguagem Unificada de Modelagem) é mostrada na Figura 1.14.

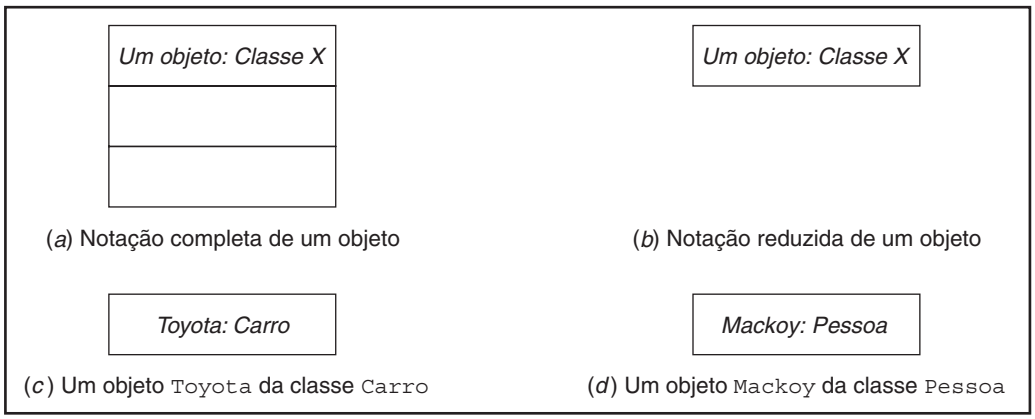


Figura 1.14 Representação de objetos em UML (Linguagem Unificada de Modelagem).

1.6.5 Classes

Em POO, os objetos são membros das **classes**. Essencialmente, uma classe é um tipo de dado como qualquer outro definido em uma linguagem de programação. A diferença está no fato de a classe ser um tipo de dado que contém dados e funções. Uma classe contém muitos objetos, por isso é preciso defini-la, ainda que sua definição não implique a criação de objetos (Figura 1.15).

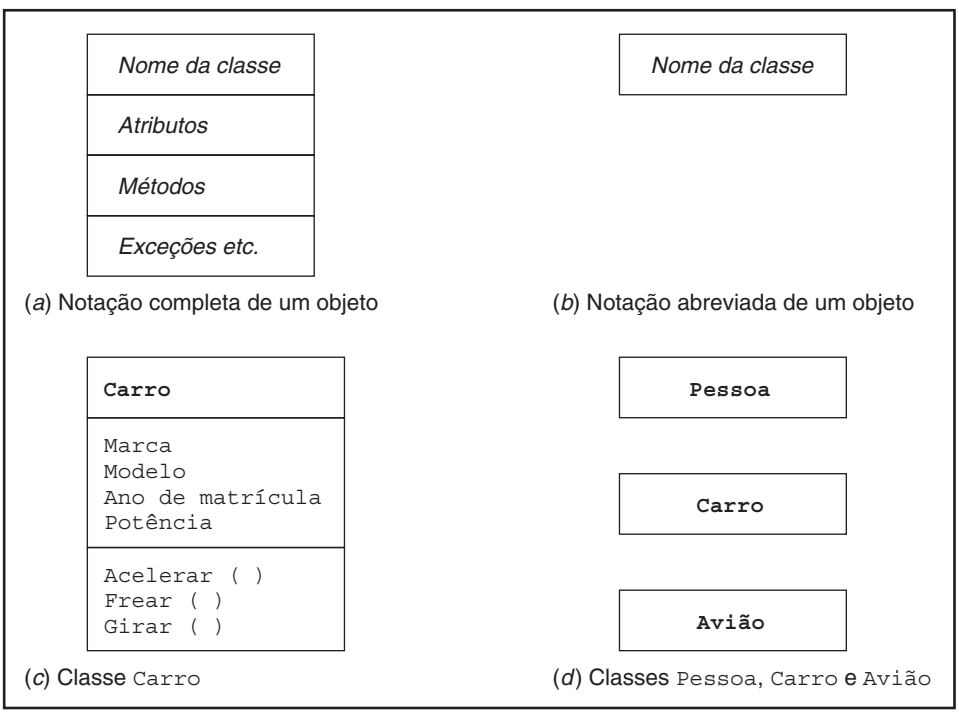


Figura 1.15 Representação de classes em UML.

Uma classe é, conseqüentemente, uma descrição de um número de objetos similares. `Madonna, Sting, Prince, Juanes, Carlos Vives` ou `Juan Luis Guerra` são membros ou objetos da classe `músicos de rock`. Um objeto concreto, `Juanes` ou `Carlos Vives`, são instâncias da classe `músicos de rock`.

Em **C++,** uma classe é uma estrutura de dado ou tipo de dado que contém funções (métodos) como membros e dados. Uma classe é uma descrição geral de um conjunto de objetos similares. Por definição, todos os objetos de uma classe compartilham os mesmos atributos (dados) e as mesmas operações (métodos). Uma classe encapsula as abstrações de dados e operações necessárias para descrever uma entidade ou objeto do mundo real.

Em **UML**, uma classe é representada por meio de um retângulo que contém uma faixa com o nome da classe e opcionalmente outras duas faixas com o nome de seus atributos e de suas operações ou métodos (Figura 1.16).

1.6.6 Generalização e especialização: herança

A *generalização* é a propriedade que permite compartilhar informação entre duas entidades evitando a redundância. No comportamento de objetos, existem, com freqüência, propriedades que são comuns em diferentes objetos e a essa propriedade se denomina generalização.

Por exemplo, máquina de lavar, geladeira, forno de microondas, torradeira, lava-louças etc. são todos eletrodomésticos (aparelhos para o lar). No mundo da orientação a objetos, cada um desses aparelhos é uma **`subclasse`** da classe `Eletrodoméstico` e, por sua vez, `Eletrodoméstico` é uma **`superclasse`** de todas as outras classes (`máquina de lavar`, `geladeira`, `forno de microondas`, `torradeira`, `lava-louças`...). O processo inverso da generalização pelo qual são definidas novas classes de outras já existentes se chama *especialização*.

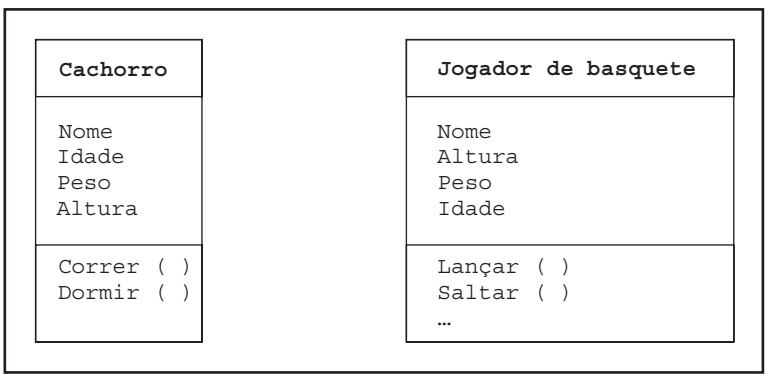


Figura 1.16 Representação de classes em UML com atributos e métodos.

Em orientação a objetos, se denomina **herança** ao mecanismo que implementa a propriedade de generalização. A herança permite definir novas classes de outras classes já existentes, de modo que estas apresentem as mesmas características e comportamento das anteriores, assim como outras adicionais.

A idéia de classes leva à idéia de herança. Classes diferentes podem ser conectadas umas as outras de modo hierárquico. Como já foi comentado nas relações de generalização e especialização, em nosso cotidiano, utilizamos o conceito de classes dividido em subclasses. A classe `animal` é dividida em `anfíbios`, `mamíferos`, `insetos`, `pássaros` etc., e a classe `veículo` em `carros`, `motos`, `caminhões`, `ônibus` etc.

O princípio da divisão ou classificação é que cada subclasse compartilha características comuns com a classe à qual pertence. Os carros, motos, caminhões e ônibus têm rodas, motores e carrocerias; estas são as características que definem um veículo. Além das características em comum com os outros membros da classe, cada subclasse tem suas próprias características. Os caminhões, por exemplo, têm uma cabine independente do contêiner que transporta a carga; os ônibus têm um grande número de assentos independentes para os passageiros etc. Na Figura 1.17, são mostradas classes pertencentes a uma hierarquia ou herança de classes.

De modo semelhante, uma classe pode se converter em pai ou raiz de outras subclasses. Em C++, a classe original é chamada *classe base* e as classes dela derivadas são denominadas *classes derivadas* e sempre são uma especialização ou *concretização* de sua classe base. Ao contrário, a classe base é a generalização da classe derivada. Isso significa que todas as propriedades (atributos e operações) da classe base são herdadas pela classe derivada, normalmente complementada com propriedades adicionais.

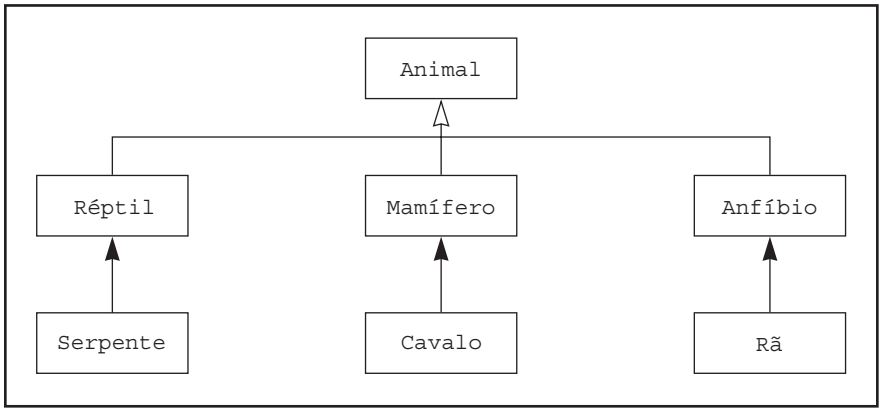


Figura 1.17 Herança de classes em UML.

1.6.7 Reutilização

Uma vez que uma classe foi criada, escrita e depurada, pode ser distribuída a outros programadores para que utilizem em seus próprios programas. Essa propriedade se chama *reutilização*.[4] Seu conceito é semelhante ao das funções incluídas nas bibliotecas de funções de uma linguagem procedimental como C e pode incorporar-se em diferentes programas.

Em C++, o conceito de herança proporciona uma extensão ou ampliação do conceito de *reutilização*. Um programador pode considerar uma classe existente, modificá-la, acrescentar competências e propriedades adicionais. Isso se consegue derivando uma nova classe de uma já existente. A nova classe herdará as características da classe antiga, mas será livre para acrescentar novas características próprias.

A facilidade de reutilizar um software existente é um dos grandes benefícios da POO: muitas empresas conseguem com a reutilização de classe em novos projetos a redução de custos de inversão em seus orçamentos de programação. Em essência, quais são as vantagens da herança? Primeiro, é utilizada para coerência e para reduzir código. As propriedades comuns de várias classes somente necessitam ser implementadas uma vez e só necessitam ser modificadas uma vez, caso seja necessário. A outra vantagem é que suporta o conceito de abstração da funcionalidade comum.

1.6.8 Polimorfismo

Além das vantagens de coerência e redução de código, a herança contribui também com outra grande vantagem: facilitar o polimorfismo. Polimorfismo é a propriedade que permite que um operador ou uma função atue de modo diferente em função do objeto sobre o qual se aplicam. Na prática, o polimorfismo significa a capacidade de uma operação ser interpretada apenas pelo próprio objeto que a chama. Do ponto de vista prático de execução do programa, o polimorfismo é realizado na execução já que durante a compilação não se conhece o tipo de objeto e, conseqüentemente, que operação foi solicitada. No Capítulo 14, descreveremos em profundidade a propriedade de polimorfismo e seus diferentes modos de implementação.

A propriedade de **polimorfismo** é aquela em que uma operação tem o mesmo nome em diferentes classes, mas é executada de diferentes formas em cada classe. Assim, por exemplo, a operação de abrir pode ocorrer em diferentes classes: abrir uma porta, uma janela, um jornal, um arquivo, uma conta corrente em um banco, um livro etc. Em cada caso, uma operação diferente é executada, ainda que tenha o mesmo nome em todas elas, "abrir". O polimorfismo é a propriedade de uma operação de ser interpretada somente pelo objeto ao qual pertence. Existem diferentes formas de implementar o polimorfismo e a variação acorrerá dependendo da linguagem de programação.

Vejamos o conceito com exemplos da vida diária.

Em uma oficina de automóveis, existem muitos carros de marcas, modelos, tipos e potências diferentes etc. Constituem uma classe ou coleção heterogênea de carros. Suponhamos que tenhamos de realizar uma operação

[4] O termo provém do conceito inglês *reusability*. A tradução não foi aprovada pela RAE (Academia Real da Espanha) mas se incorpora ao texto pelo seu grande uso e pela difusão entre os profissionais de informática.

comum: "trocar os freios do carro". A operação a ser realizada é a mesma, inclui os mesmos princípios, no entanto, dependendo do carro em particular, a operação será muito diferente, incluirá diferentes ações para cada caso. Outro exemplo a considerar é relativo aos operadores "+" e "*" aplicados a números inteiros ou números complexos. Ainda que ambos sejam números, no primeiro caso, a soma e a multiplicação são operações simples, enquanto no caso dos números complexos, como são compostos por uma parte real e uma parte imaginária, será necessário seguir um método específico para tratar ambas as partes e obter um resultado que também será um número complexo.

O uso de operadores ou funções de forma diferente, dependendo dos objetos sobre os quais estão atuando, se chama polimorfismo (uma coisa com diferentes formas). Entretanto, quando se permite a um operador existente, tal como + ou =, a possibilidade de operar sobre novos tipos de dados, se diz que o operador está sobrecarregado. A sobrecarga é um tipo de polimorfismo e uma característica importante da POO. No Capítulo 10 se ampliará, também em profundidade, esse novo conceito.

1.7 O *SOFTWARE* (OS PROGRAMAS)

O *software* de um computador é um conjunto de instruções de programa detalhadas que controlam e coordenam os componentes *hardware* de um computador e controlam as operações de um sistema informático. O auge do computador no século passado e no século atual se deve, essencialmente, ao desenvolvimento de sucessivas gerações de *software* potentes e cada vez mais *amistosos* ("fáceis de utilizar").

As operações que o *hardware* deve realizar são especificadas por uma lista de instruções denominada programa ou *software*. Um programa de software é um conjunto de **sentenças** ou **instruções** ao computador. O processo de codificação de um programa é conhecido como **programação** e as pessoas que se especializam nessa atividade são chamadas **programadores**. Existem dois tipos importantes de software: software do sistema e software de aplicações. Cada tipo realiza uma função diferente.

Sofware do sistema é um conjunto generalizado de programas que administra os recursos do computador, tal como o processador central, enlaces de comunicações e dispositivos periféricos. Os programadores que escrevem software do sistema são chamados **programadores de sistemas**. **Sofware de aplicações** é um conjunto de programas escritos por empresas, usuários ou em equipe que instruem o computador para que execute uma tarefa específica. Os programadores que escrevem software de aplicações são chamados **programadores de aplicações**.

Os dois tipos de software estão relacionados entre si de modo que os usuários e os programadores possam fazer um uso eficiente do computador. A Figura 1.18 mostra uma visão organizacional de um computador onde aparecem os diferentes tipos de software como camadas do computador desde o seu interior (hardware) até o seu exterior (usuário). As diferentes camadas funcionam graças às instruções específicas que fazem parte do software do sistema e chegam ao software de aplicação, programado pelos programadores de aplicações, que é utilizado pelo usuário, que não necessita ser um especialista.

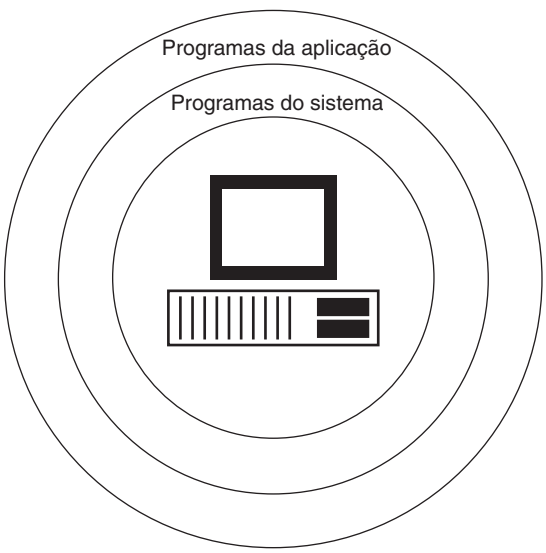


Figura 1.18 Relação entre programas de aplicação e programas do sistema.

1.7.1 Software do sistema

O software do sistema coordena as diferentes partes de um sistema de computador, conecta e interage entre o software de aplicação e o hardware do computador. Outro tipo de software do sistema que administra, controla as atividades do computador e realiza tarefas comuns de processamento é denominado **utilitário**. O software do sistema que administra e controla as atividades do computador é chamado **sistema operacional**. Outro software do sistema são os programas tradutores ou de tradução de linguagens de computador, que convertem as linguagens de programação, entendíveis pelos programadores, em linguagem de máquina, entendíveis pelos computadores.

O **software do sistema** é um conjunto de programas indispensáveis para que a máquina funcione e é também chamado *programas do sistema*. Esses programas são, basicamente, o *sistema operacional*, os *editores de texto*, os *compiladores/interpretadores* (linguagens de programação) e os *programas utilitários*.

1.7.2 Software de aplicação

O software de aplicação tem como função principal assistir e ajudar um usuário de um computador a executar tarefas específicas. Os programas de aplicação podem ser desenvolvidos com diferentes linguagens e ferramentas de software. Por exemplo, uma aplicação de processamento de textos (*word processing*), tal como Word ou Word Perfect que ajuda a criar documentos, uma folha de cálculo como Lótus 1-2-3 ou Excel que ajudam a automatizar tarefas entediantes ou repetitivas de cálculos matemáticos ou estatísticos, a gerar diagramas ou gráficos, apresentações visuais como PowerPoint ou a criar bases de dados como Acces ou Oracle que ajudam a criar arquivos e registros de dados.

Os usuários, normalmente, compram o software de aplicações em discos CDs ou DVDs (antigamente em disquetes) ou baixam da Internet e têm de instalar o software copiando os programas correspondentes dos discos no disco rígido do computador. Quando comprar esses programas, certifique-se que são compatíveis com seu computador e com seu sistema operacional. Existe uma grande diversidade de programas de aplicação para todo tipo de atividade, tanto pessoal quanto de negócios, navegação e manipulação da Internet, gráficos e apresentações visuais etc.

As *linguagens de programação* servem para escrever programas que permitam a comunicação usuário/máquina. Programas especiais chamados *tradutores* (**compiladores ou interpretadores**) convertem as instruções escritas em linguagens de programação em instruções escritas em linguagens de máquina (0 e 1, *bits*) para que a máquina possa entender.

Os *programas utilitários* facilitam o uso do computador. Um bom exemplo é um *editor de textos* que permite a edição de documentos. Este livro foi escrito utilizando-se um editor de textos ou *processador de palavras* (***word processor***).

Os programas que realizam tarefas concretas, folha de pagamento, contabilidade, análise estatística etc., ou seja, os programas que poderão escrever em Turbo Pascal, são denominados *programas de aplicação*. Neste livro, veremos pequenos programas de aplicação que mostram os princípios de uma boa programação de computador.

Devemos diferenciar entre o ato de criar um programa e a ação do computador quando executa as instruções do programa. A criação de um programa se faz inicialmente no papel e, em seguida, é introduzido no computador e convertido em linguagem entendível pelo computador. A Figura 1.19 mostra o processamento geral de execução de um programa com uma entrada (*dados*) ao programa e a obtenção de uma saída (*resultados*). A entrada pode ter uma variedade de formas, como texto, informação numérica, imagens ou som. A saída também pode ter formas, tais como dados numéricos ou caracteres, sinais para controlar equipamentos ou robôs etc.

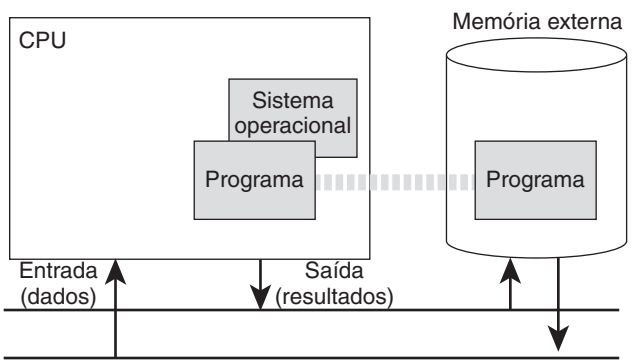


Figura 1.19 Execução de um programa.

1.8 SISTEMA OPERACIONAL

Um sistema operacional **SO** (*Operating System*) é, talvez, a parte mais importante do software do sistema e é o software que controla e administra os recursos do computador. Na prática, o sistema operacional é a coleção de programas de computador que controla a interação do usuário e do hardware do computador. O sistema operacional é o administrador principal do computador e, por isso, às vezes, é comparado ao maestro de uma orquestra, já que esse software é o responsável por dirigir todas as operações do computador e administrar todos os seus recursos.

O sistema operacional atribui recursos, planeja o uso de recursos e tarefas do computador e monitora as atividades do sistema informático. Esses recursos incluem memória, dispositivos de **E/S** (Entrada/Saída) e **CPU** (Unidade Central de Processamento). O sistema operacional proporciona serviços tais como atribuir memória a um programa e manipulação do controle de dispositivos E/S como monitor, teclado ou as unidades de disco. A Tabela 1.6 mostra alguns dos sistemas operacionais mais populares utilizados no ensino e na informática profissional.

Quando um usuário interage com um computador, a interação é controlada pelo sistema operacional. Um usuário se comunica com um sistema operacional por meio de uma interface de usuário desse sistema operacional. Os sistemas operacionais modernos utilizam uma interface gráfica de usuário, **IGU** (*Graphical User Interface, GUI*), que usa ícones, botões, barras e quadros de diálogo para realizar tarefas que são controladas, entre outros dispositivos, pelo teclado ou pelo mouse.

Geralmente, o sistema operacional é armazenado de modo permanente em um *chip* de memória somente de leitura (ROM) de modo que esteja disponível sempre que se ligue o computador. Outra parte do sistema operacional pode estar em disco e é armazenado na memória RAM na inicialização do sistema, pela primeira vez, por meio de uma operação chamada *carga* do sistema (*booting*).

Um dos programas mais importantes é o **sistema operacional** que serve, essencialmente, para facilitar a escrita e o uso dos seus próprios programas. O sistema operacional dirige as operações globais do computador, instrui o computador para executar outros programas e controla o armazenamento e recuperação de arquivos (programas e dados) de fitas e discos. Graças ao sistema operacional, é possível ao programador introduzir e gravar novos programas, bem como instruir o computador para que os execute. Os sistemas operacionais podem ser: *monousuários* (um só usuário), *multiusuários* ou tempo compartilhado (diferentes usuários), atendendo ao número de usuários e *monocarga* (uma só carga) ou *multitarefa* (tarefas múltiplas) de acordo com as tarefas (processamentos) que possa realizar simultaneamente. C++ roda em quase todos os sistemas operacionais, Windows XP, Windows 95, Windows NT, Windows 2000, Unix, Linux, Vista..., e em quase todos os computadores pessoais atuais PC, Mac, Sun etc.

Tabela 1.6 Sistemas operacionais mais utilizados no ensino e na empresa

Sistema operacional	Características
Windows Vista[5]	Novo sistema operacional da Microsoft apresentado em 2006, mas que foi lançado comercialmente em 2007.
Windows XP	Sistema operacional mais utilizado na atualidade, tanto no campo do ensino como na indústria e negócios. Fabricado pela Microsoft.
Windows 98/ME/2000	Versões anteriores do Windows mas que ainda são muito utilizados.
Unix	Sistema operacional aberto, escrito em C e muito usado no campo profissional.
Linux	Sistema operacional de software aberto, gratuito e de livre distribuição, similar a Unix e uma grande alternativa para o Windows. Muito utilizado atualmente em servidores de aplicações para a Internet.
Mac OS	Sistema operacional de computadores Apple Macintosh.
DOS e OS/2	Sistemas operacionais criados respectivamente pela Microsoft e IBM, pouco utilizados, mas que serviram como base para todos os atuais sistemas operacionais.
CP/M	Sistema operacional de 8 bits para os primeiros computadores surgidos na década de 1970.
Symbian	Sistema operacional para telefones móveis apoiado fundamentalmente pelo fabricante de telefones celulares Nokia.
PalmOS	Sistema operacional para agendas digitais, PDA, do fabricante Palm.
Windows Mobile, CE	Sistema operacional para telefones móveis com arquitetura e aparências similares ao Windows XP.

[5] A Microsoft lançou um novo sistema operacional chamado Windows Vista, uma atualização do Windows XP com numerosas funcionalidades, especialmente para a Internet e de segurança, incluindo no sistema operacional programas que atualmente são comercializados de forma independente, como reprodução de música e vídeo e, fundamentalmente, um sistema de representação gráfica muito potente, que permite construir aplicações em três dimensões, além de um sistema de busca, um sistema antivírus e outras funcionalidades.

1.8.1 Tipos de sistemas operacionais

As diferentes características especializadas do sistema operacional permitem aos computadores manejar diferentes tarefas, bem como múltiplos usuários de modo simultâneo, em paralelo ou de modo seqüencial... Com base em suas características específicas, os sistemas operacionais podem classificar-se em vários grupos, descritos a seguir:

Multiprogramação/Multitarefa

A multiprogramação permite a múltiplos programas compartilhar recursos de um sistema de computador a qualquer momento pelo uso paralelo de uma CPU. Só um programa utiliza realmente a CPU em dado momento, no entanto, as necessidades de entrada/saída podem ser atendidas ao mesmo tempo. Dois ou mais programas estão ativos ao mesmo tempo, mas não utilizam os recursos do computador simultaneamente. Com a multiprogramação um grupo de programas é executado alternativamente e se alternam no uso do computador. Quando se utiliza um sistema operacional de um único usuário, a multiprogramação adota o nome de **multitarefa**.

> ***Multiprogramação***
>
> Método de execução de dois ou mais programas concorrentemente utilizando o mesmo computador. A UCO executa apenas um programa, mas pode, ao mesmo tempo, atender os serviços de entrada/saída dos outros.

Tempo compartilhado (múltiplos usuários, time sharing)

Um sistema operacional multiusuário é um sistema operacional que tem a capacidade de permitir que muitos usuários compartilhem simultaneamente os recursos de processamento do computador. Centenas ou milhares de usuários podem conectar-se ao computador que atribui um tempo de computador a cada usuário, de modo que, à medida que libera a tarefa de um usuário, realiza a tarefa do seguinte e assim sucessivamente. Dada a alta velocidade de transferência das operações, a sensação é que todos os usuários estão simultaneamente conectados à CPU com cada usuário recebendo unicamente um tempo de máquina.

Multiprocessamento

Um sistema operacional trabalha em multiprocessamento quando pode enlaçar dois ou mais computadores para trabalhar em paralelo em um único sistema de computador. O sistema operacional pode atribuir múltiplas CPUs para executar simultaneamente diferentes instruções do mesmo programa ou de programas diferentes, dividindo o trabalho entre as diferentes CPUs.

> A multiprogramação utiliza processamento em paralelo com uma CPU; o multiprocessamento usa processamento simultâneo com múltiplas CPUs.

1.9 LINGUAGENS DE PROGRAMAÇÃO

Como vimos na seção anterior, para que um processador realize um processamento, é necessário, em primeiro lugar, fornecer-lhe um algoritmo adequado. O processador deve ser capaz de *interpretar* o algoritmo, o que significa:

- compreender as instruções de cada passo,
- realizar as operações correspondentes.

Quando o processador é um computador, o algoritmo deve se expressar em um formato denominado *programa*, já que o pseudocódigo ou o diagrama de fluxos não são compreensíveis para o computador, embora qualquer programador possa entendê-los. Um programa é escrito em uma *linguagem de programação*, e as operações que levam a expressar um algoritmo em forma de programa são chamadas *programação*. Sendo assim, as linguagens utilizadas para escrever programas de computadores são as *linguagens de programação* e os **programadores** são as pessoas que escrevem e desenvolvem programas. O processamento de tradução de um algoritmo em *pseudocódigo* em uma linguagem de programação é denominado **codificação**, e o algoritmo escrito em uma linguagem de programação é denominado **código-fonte**.

Na realidade, o computador não entende diretamente as linguagens de programação, por isso é necessário um programa que traduza o código-fonte em outra linguagem que o computador entenda, mas que é muito complexa para as pessoas. Essa linguagem é conhecida como **linguagem de máquina** e o código correspondente como **código de máquina**. Os programas que traduzem o código-fonte escrito em uma linguagem de programação — tal como C++ — em código de máquina são chamados **tradutores**. O processamento de conversão de um algoritmo escrito em pseudocódigo a um programa executável compreendido pela máquina é mostrado na Figura 1.20.

Atualmente, a maioria dos programadores emprega linguagens de programação como C++, C, C#, Java, Visual Basic, XML, HTML, Perl, PHP, JavaScript..., ainda que profissionalmente se utilizem as linguagens clássicas *COBOL*, *FORTRAN*, *Pascal* ou o mítico BASIC. Estas linguagens são denominadas **linguagens de alto nível** e permitem aos profissionais resolver problemas convertendo seus algoritmos em programas escritos em alguma dessas linguagens de programação.

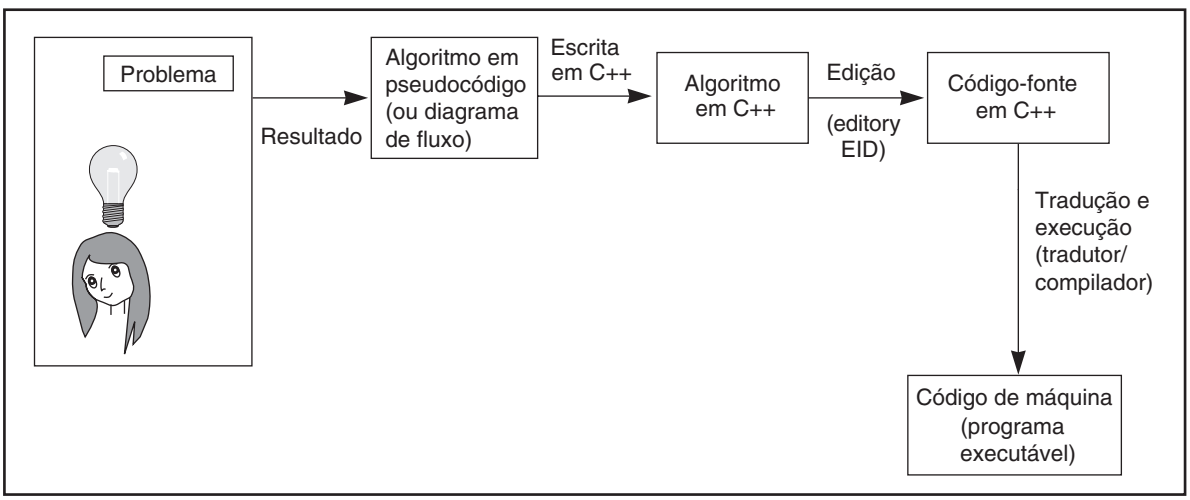


Figura 1.20 Processamento de transformação de um algoritmo em pseudocódigo em um programa executável.

1.9.1 Tradutores de linguagem: o processamento de tradução de um programa

O processamento de tradução de um programa-fonte escrito em uma linguagem de alto nível em uma linguagem de máquina compreensível para o computador é realizado por meio de programas chamados "tradutores". Os **tradutores de linguagem** são programas que traduzem os programas-fonte escritos em linguagem de alto nível em código de máquina. Os tradutores de dividem em **compiladores** e **interpretadores**.

Interpretadores

Um *interpretador* é um tradutor que toma um programa-fonte, o traduz e em seguida o executa. Os programas interpretadores clássicos como BASIC têm sido utilizados somente em ocasiões especiais, entretanto, está muito difundida a versão interpretada da linguagem Smalltalk que é uma linguagem orientada a objetos puros. O sistema de tradução consiste em: traduzir a primeira sentença do programa em linguagem de máquina, deter a tradução e executar a sentença; em seguida se traduz a sentença seguinte, se detém a tradução e se executa a sentença e assim sucessivamente até terminar o programa.

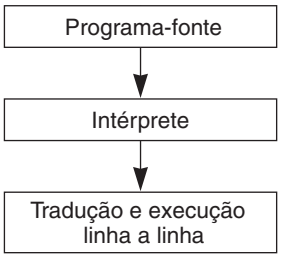


Figura 1.21 Interpretador.

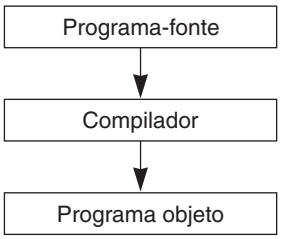


Figura 1.22 Compilação de programas.

Compiladores

Um *compilador* é um programa que traduz os programas-fonte escritos em linguagem de alto nível em linguagem de máquina. A tradução do programa completa é realizada em uma só operação denominada **compilação** do programa, isto é, todas as instruções do programa são traduzidas em um só bloco. O programa compilado e depurado (eliminados os erros do código-fonte) é chamado *programa executável*, porque já pode ser executado diretamente e quantas vezes forem necessárias. A compilação só deverá ser novamente executada no caso de modificação de alguma instrução do programa. Desse modo, o programa executável não necessita de compilador para sua execução. As linguagens compiladas típicas mais utilizadas são: **C**, **C++**, **Java**, **C#**, **Pascal**, **FORTRAN** e **COBOL**.

1.9.2 A compilação e suas fases

A *compilação* é o processamento de tradução de programas-fonte em programas-objeto. O programa-objeto obtido da compilação foi traduzido normalmente em código de máquina.

Para conseguir o programa máquina real, devemos utilizar um programa chamado *montador* ou *enlaçador* (*linker*). O processamento de montagem leva a um programa em linguagem de máquina diretamente executável (Figura 1.23).

O processamento de execução de um programa escrito em uma linguagem de programação e por meio de um compilador costuma obedecer aos seguintes passos:

1. Escrever o *programa-fonte* com um *editor* (programa que permite a um computador atuar de modo similar a uma máquina de escrever eletrônica) e guardá-lo em um dispositivo de armazenamento (por exemplo, um disco).
2. Introduzir o programa-fonte em memória.
3. *Compilar* o programa com o compilador C.
4. *Verificar* e *corrigir erros de compilação* (lista de erros).
5. Obtenção do *programa-objeto*.
6. O montador (*linker*) obtém o *programa executável*.
7. Executa-se o programa e, se não houver erros, se obterá a saída do programa.

O processamento de execução é mostrado nas Figuras 1.24 e 1.25.

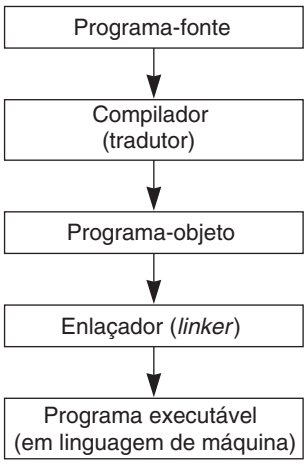


Figura 1.23 Fases da compilação.

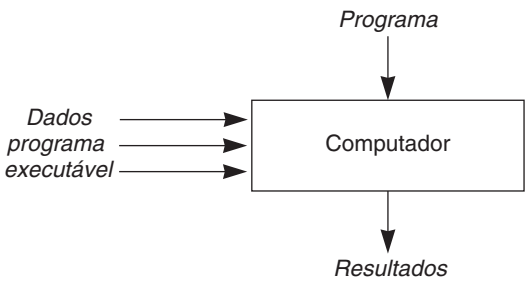


Figura 1.24 Execução de um programa.

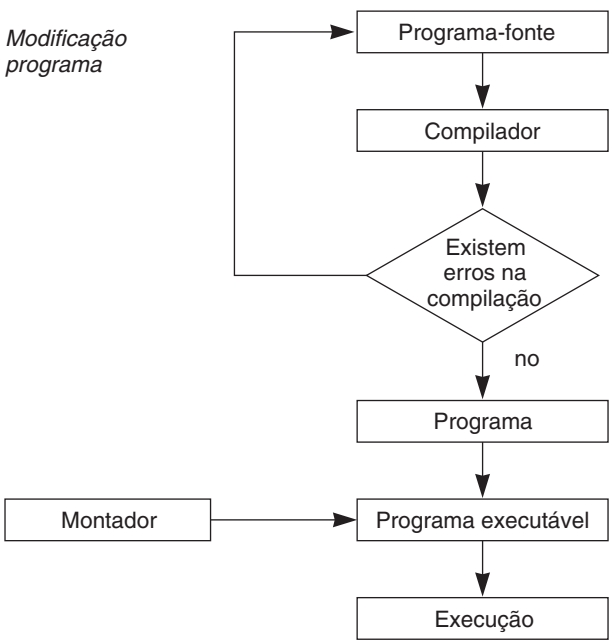


Figura 1.25 Fases de execução de um programa.

No Capítulo 2 será descrito, em detalhes, o processamento completo e específico de execução de programas em linguagem C.

1.10 C: A ORIGEM DE C++ COMO LINGUAGEM UNIVERSAL

C é a linguagem de programação especialmente associada, de modo universal, ao sistema operacional Unix. Entretanto, a popularidade, eficácia e potência de C se deve ao fato de essa linguagem não estar associada a nenhum sistema operacional nem a nenhuma máquina em especial. Esta é a razão fundamental pela qual C é conhecida como a ***linguagem de programação de sistemas, por excelência***.

C é uma evolução das linguagens BCPL – desenvolvida por Martin Richards – e B – desenvolvida por Ken Thompson em 1970 – para o primitivo Inix do computador DEC PDP-7.

C surgiu em 1978 com a publicação de *The C Programming languaje*, por Brian Kernighan e Dennis Ritchie (Prentice-Hall, 1978) e foi crescendo em popularidade desde seu surgimento. As sucessivas mudanças na linguagem ao longo dos anos com a criação de compiladores por grupos não envolvidos em sua criação, fizeram pensar na padronização da definição da linguagem C.

Assim, em 1983, o American National Standard Institute (ANSI), uma organização internacional de padronização, criou um comitê (o chamado X3J11) cuja tarefa principal consistia em fazer "*uma definição não ambígua da linguagem C e independente da máquina*". Havia nascido o padrão ANSI da linguagem C. Com essa definição de C se assegura que qualquer fabricante de software que vende um compilador ANSI C incorpora todas as características da linguagem especificadas pelo padrão. Isso significa também que os programadores que escrevam programas em C padrão tenham a segurança de que suas modificações poderão ser feitas em qualquer sistema que tenha um compilador C.

C é uma *linguagem de alto nível* que permite programar com instruções de linguagem de propósito geral. C também é definida como uma linguagem de programação estruturada de propósito geral, ainda que em seu projeto se destacou o fato de que foi especificada como uma linguagem de programação de sistemas, o que proporciona enorme quantidade de potência e flexibilidade.

O padrão ANSI C formaliza construções não propostas na primeira versão de C, em especial atribuições de estruturas e enumerações. Entre outras contribuições, definiu-se principalmente, uma nova forma de declaração de funções (protótipos). Outra das grandes contribuições é, essencialmente, a biblioteca-padrão de funções.

Hoje, no século XXI, C continua sendo uma das linguagens de programação mais utilizadas na indústria de software, bem como em institutos tecnológicos, escolas de engenharia e universidades. Praticamente, todos os fabricantes de sistemas operacionais, Windows, Unix, Linux, Mac OS, Solaris... suportam diferentes tipos de compiladores de linguagem C e, em muitas ocasiões, fazem distribuições gratuitas de qualquer dos sistemas operacionais citados. Todos os compiladores de C++ podem executar programas escritos na linguagem C, principalmente se atendem ao padrão ANSI C.[6]

1.11 A LINGUAGEM C++: HISTÓRIA E CARACTERÍSTICAS

C++, Java e C#, as três linguagens mais populares juntamente com C, nesta primeira década do século XXI, são herdeiras diretas do próprio C com características orientadas a objetos e a Internet. Atualmente, embora C esteja sendo, talvez, a primeira linguagem de programação mais utilizada no mundo da **educação**, e também ganhe uma porcentagem alta de utilização no campo profissional, as três linguagens com características técnicas de orientação a objetos formam com C o *poker* de linguagens mais empregadas no mundo educativo, profissional e científico atual e previsivelmente dos próximos anos.

C++ é herdeira direta da linguagem C que, por sua vez, é derivada da linguagem B (Richards, 1980). C se mantém como um subconjunto de C++. Outra fonte de inspiração, como assinala seu autor Bjarne Stroustrup [Stroustrup, 1997],[7] foi Simula 67 [Dahl, 1972], em que baseou o conceito de classe (com classes derivadas e funções virtuais).

A linguagem de programação C foi desenvolvida por **Dennis Ritche** da AT&T Bell Laboratories, que a utilizou para escrever e manter o sistema operacional Unix (até que apareceu C, o sistema operacional Unix foi desenvolvido por **Ken Thompson** na AT&T Laboratories por meio da linguagem "montadora" ou em B). C é uma linguagem

---

[6] Boas opções gratuitas podem ser encontradas no site do fabricante de software Borland. Podem-se localizar e descarregar um excelente compilador Dev-C++ em software livre para compilar código C e em C++, nos sites *www.bloodshed.net* e *www.download.com*, podem-se encontrar diferentes compiladores totalmente gratuitos. É possível achar outros numerosos sites em software gratuito em vários sites da rede. Os fabricantes de software e de computadores (IBM, Microsoft, HP...) oferecem versões a seus clientes ainda que não sejam gratuitas.

[7] P. 11.

de propósito geral que pode ser utilizada para escrever qualquer tipo de programa, mas seu êxito e popularidade estão especialmente relacionados com o sistema operacional Unix. (Foi desenvolvida como *linguagem de programação de sistemas*, isto é, uma linguagem de programação para escrever *sistemas operacionais* e utilitários (programas) do sistema.) Os sistemas operacionais são os programas que administram os *recursos do computador*. Além de Unix, exemplos bem-conhecidos de sistemas operacionais são MS/DOS, OS/2, MVS, Lynux, Windows 95/98, Windows NT, Windows 2000, OS Mac etc.

A especificação formal da linguagem C é um documento escrito por Ritche, entitulado *The C reference manual*. Em 1997, **Ritche** e **Brian Kernighan** ampliaram esse documento e publicaram um livro referência da linguagem *The C programming lenguage* (também conhecido por K&R).

Embora C seja uma linguagem muito potente, possui duas características que a tornam inapropriadas com uma introdução moderna à programação. Primeiro, porque C requer de seus usuários um nível de sofisticação, o que obriga os programadores principiantes a uma difícil aprendizagem, já que é de difícil compreensão. Segundo, porque C foi projetada em princípios dos anos de 1970 e a natureza da programação mudou de modo significativo nas décadas de 1980 e 1990.

Para sanar essas "deficiências", Bjarne Stroustrup da AT&T Bell Laboratories desenvolveu, no início da década de 1980, C++. Stroustrup projetou C++ como um C melhor. Em geral, C padrão é um subconjunto de C++ e a maioria dos programas C também são programas C++ (a afirmação inversa não é verdadeira). C++, além de acrescentar propriedades a C, apresenta características e propriedades de *programação orientada a objetos*, que é uma técnica de programação muito potente e que será vista na última parte deste livro.

Foram apresentadas várias versões de C++ e sua evolução foi estudada em [Stroustrup 94]. As principais características que foram sendo incorporadas a C++ são: herança múltipla, *generalidade*, modelos, funções virtuais, exceções etc. C++ foi evoluindo ano a ano e como seu autor explicou *evoluiu sempre para solucionar problemas encontrados pelos usuários e como conseqüência de discussões entre o autor, seus amigos e colegas.*[8]

IMPORTANTE: Página oficial de Bjarne Stroustrup

Bjarne Stroustrup, projetista e implementador da linguagem de programação C++, é a referência fundamental e definitiva para qualquer estudante e programador de C++. Suas obras *The C++ programming lenguage*,[9] *The design and evolution of C++* e *C++* e *Reference manual* são leitura e consulta obrigatórias.

Sua página da Web pessoal da AT&T Labs Researchs é o primeiro site de "favoritos" e que recomendamos seja visitada com certa freqüência.

www.resarch.att.com/~bs

O site é atualizado com freqüência por Stroustrup e contém grande quantidade de informação e uma excelente seção de FAQ (frequently asked questions).

O projeto de padronização de C++ começou com o comitê ANSI e sua primeira referência é *The Annotated C++ Reference Manual* [**Ellis**, 89].[10] Em dezembro de 1989, o comitê X3J16 do ANSI se reuniu por iniciativa de Hewlett-Packard. Em junho de 1991, a ISO (*International Organization of Standardization*), com seu próprio comitê (ISO-WG-21), se uniu ao comitê de padronização de ANSI, criando um esforço comum ANSI/ISO para desenvolver um padrão para C++. Esses comitês se reúnem três vezes por ano para somar esforços e chegar a uma decisão de criação de um padrão que convertesse C++ em uma linguagem importante e de ampla difusão.

Em 1995, o comitê publicou diferentes artigos (*working paper*) e, em abril desse ano, foi publicado um rascunho do padrão (Comitê Draft) para seu exame público. Em dezembro de 1996, foi lançada também para domínio público uma segunda versão (CD2). Esses documentos não somente refinaram a descrição das características

[8] [Stroustrup, 98], p.12.

[9] Essa obra foi traduzida por uma equipe de professores da Universidad Pontificia de Salamanca, coordenada e dirigida pelo autor deste livro.

[10] Existe versão espanhola de Addison-Wesley Díaz de Santos traduzida, pelos professores Manuel Katrib e Luis Joyanes.

existentes de C++, como também ampliaram a linguagem com exceções, identificação em tempo de execução (*RTII, run_time type identification*), modelos (*templates*) e a biblioteca-padrão de modelos STL (*Standard Template Library*). Stroustrup publicou, em 1997, a 3ª edição de seu livro *The C++ programming lenguage*, o qual segue o padrão ANSI/ISO C++.

C++ é uma linguagem padronizada. Em 1998, um comitê de ANSI/ISO (*American Nacional Standard Institute/Internacional Organization for Standardization*) adotou uma especificação para a linguagem conhecida como Padrão C++ de 1998, que oficialmente leva o título ISSO/ANSI C++ Padrão (ISO/IEC 14882:1998) [Standard98]. Em 2003, veio à luz uma segunda versão do Padrão e foi adotada também como [Standard03]. O padrão está disponível na Internet como arquivo PDF com o número de documento 14882 em *http://www.ansi.org*, onde pode ser adquirido. A nova edição é uma revisão técnica, significando que se ordena a primeira edição — fixação de tipos, redução de ambigüidades e similares, mas não se mudam as características da linguagem. Este livro se baseia no padrão C++ como um superconjunto válido de C. Existem diferenças entre o padrão ANSI C e as regras correspondentes de C++, mas são poucas. Realmente, ANSI C incorpora algumas características primitivas introduzidas em C++, por exemplo, o prototipado de funções e o qualificador de tipo `const`.

Antes do surgimento de ANSI C, a comunidade de C foi seguida de um padrão de fato, baseado no livro *The C programming language,* de Kernighan e Ritche (Reading, MA: Addison-Wesley Publishing Company, 1978). A princípio, esse padrão foi conhecido como K&R; com o surgimento de ANSI C, o K&R mais simples foi, com freqüência, chamado C clássico.

O padrão ANSI C não apenas definiu a linguagem C como também estabeleceu uma biblioteca-padrão de C que dá suporte a todas as implementações de ANSI C. C++ também utiliza a biblioteca e este livro se refere a ela como a biblioteca-padrão de C ou simplesmente *biblioteca-padrão*. O padrão ANSI/ISO C++ também proporciona uma biblioteca-padrão de classes.

Recentemente, o *C* padrão passou por revisão e o novo padrão chamado C99 foi adotado pela ISO, em 1999, e pela ANSI, em 2000. Esse padrão agrega algumas características de C, tais como um novo tipo integer que suporta alguns compiladores de C++. Ainda que não façam parte do padrão C++ atual, essas características podem converter-se como parte do C++ Padrão. Antes de o comitê ANSI/ISO C++ começar seu trabalho, muitas pessoas aceitaram como padrão a versão de C++ mais recente de Bell Labs. Assim um compilador pode ser descrito como compatível com a versão 2.0 ou 3.0 de C++.

Como leva muito tempo até que um fabricante de compiladores implemente as últimas especificações de uma linguagem, existem compiladores que não estão de acordo com o padrão. Isso, em alguns casos, pode conduzir a restrições. No entanto, todos os programas deste livro foram compilados com as últimas versões de diferentes compiladores.

C++ é uma linguagem orientada a objetos que se tornou muito popular e com uma grande difusão no mundo do software. Atualmente constitui uma linguagem-padrão para programação orientada a objetos, mas também pode ser utilizada como linguagem estruturada ao estilo de C quando se deseja trabalhar no modo clássico de programação estruturada, sobretudo quando se deseja trabalhar com algoritmos e estrutura de dados.

1.11.1 C *versus* C++

C++ é uma extensão de C com características mais potentes. Estritamente falando, é um superconjunto de C, como acontece com Java e C# que são supersconjuntos de C++. O ANSI C padrão não apenas define a linguagem C como também estabelece uma biblioteca-padrão de C que deve ter suporte nas implementações ANSI C. Além de utilizar sua própria biblioteca-padrão de classes, C++ ainda usa essa biblioteca além da sua própria. Há alguns anos, o novo padrão de C foi revisado e denominado C99, que foi adotado pela ISO, em 1999, e pela ANSI, em 2000. O padrão de C agregou algumas características que suportam alguns compiladores de C++.

Por essas razões, quase todas as sentenças de C também têm uma sentença correta em C++, porém, o inverso não é correto. Elementos mais importantes foram agregados a C para criar classes C, objetos e programação orientada a objetos (C++ foi chamada originalmente "C com classes"). Entretanto, a C++ foram agregadas novas características, incluindo um enfoque melhorado da entrada/saída (E/S) e um novo meio para escrever comentários. A Figura 1.26 mostra as relações entre C e C++.

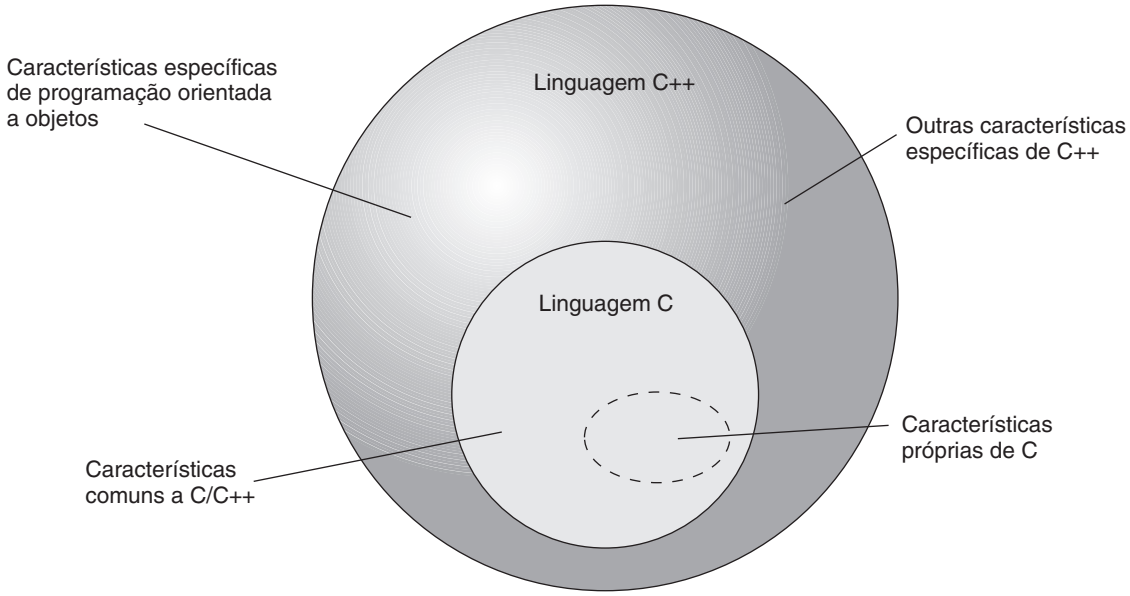


Figura 1.26 Características de C e C++.

De fato, as diferenças práticas entre C e C++ são muito maiores do que se possa pensar. Embora seja possível escrever um programa em C++ similar a um programa em C, isso raramente é feito. C++ proporciona aos programadores as novas características de C++ e os estimula a utilizá-las, mas também se costuma usar em partes dos programas as características específicas e potentes de C. Se você já conhece C, muitas propriedades lhe serão familiares e aprenderá de maneira mais rápida e eficiente, porém encontrará muitas propriedades novas e muito potentes que lhe farão desfrutar dessa nova linguagem e construir seus programas aproveitando a potência de orientação a objetos e genérica de C++.

Nosso objetivo é ajudá-lo a escrever programas em POO tão rápido quanto possível, no entanto, como já se observou, muitas características de C++ foram herdadas de C, de modo que, ainda que a estrutura global de um programa possa ser POO, consideramos que você necessite ter conhecimentos profundos do "velho estilo *procedimental*". Por isso, os capítulos da primeira parte do livro vão introduzi-lo lenta e pausadamente nas potentes propriedades orientadas a objetos das últimas partes, com o objetivo de conseguir ao término do livro o domínio da Programação em C++.

Este livro descreve o padrão ANSI/ISO C++, segunda edição, (ISO/IEC 14882:2003), de modo que os exemplos funcionem com qualquer implementação de C++ que seja compatível com o padrão. No entanto, apesar dos anos transcorridos, C++ Padrão é considerado relativamente novo e podem ser encontradas discrepâncias com o compilador que você esteja utilizando. Por exemplo, se o seu compilador não for uma versão recente, pode necessitar de espaços de nomes ou das características mais novas de modelos, como a classe "`String`" e a *Standard Template Library* que não são compatíveis com os compiladores mais antigos. Também pode acontecer que se um compilador não atende ao padrão, algumas propriedades, por exemplo, o número de bytes usados para conter um inteiro, são dependentes de implementação.

1.11.2 O futuro de C++

C++ continua evoluindo e já foram realizados trabalhos para produzir a próxima versão do padrão. A nova versão foi informalmente chamada **C++0X**, já que se espera sua conclusão para o final desta década, por volta de 2009. Bjarne Stroustrup publicou em 2 de janeiro de 2206 um artigo intitulado "*The Sc++ source: a brief look at C++0X*" no qual coloca brevemente os princípios do trabalho do novo padrão e as esperanças que existem de que o comitê ISO C++ conclua seu trabalho em 2008 e que este possa de fato converter-se no novo padrão **C++09**. Em outras palavras, comenta Stroustrup, trata-se de reforçar o padrão **C++98** que já é uma linguagem fortemente implantada. O que se pretende é fazer de C++ uma linguagem melhor para programação de sistemas e construção de bibliotecas, bem como uma linguagem C++ que seja mais fácil de ensinar e de aprender.

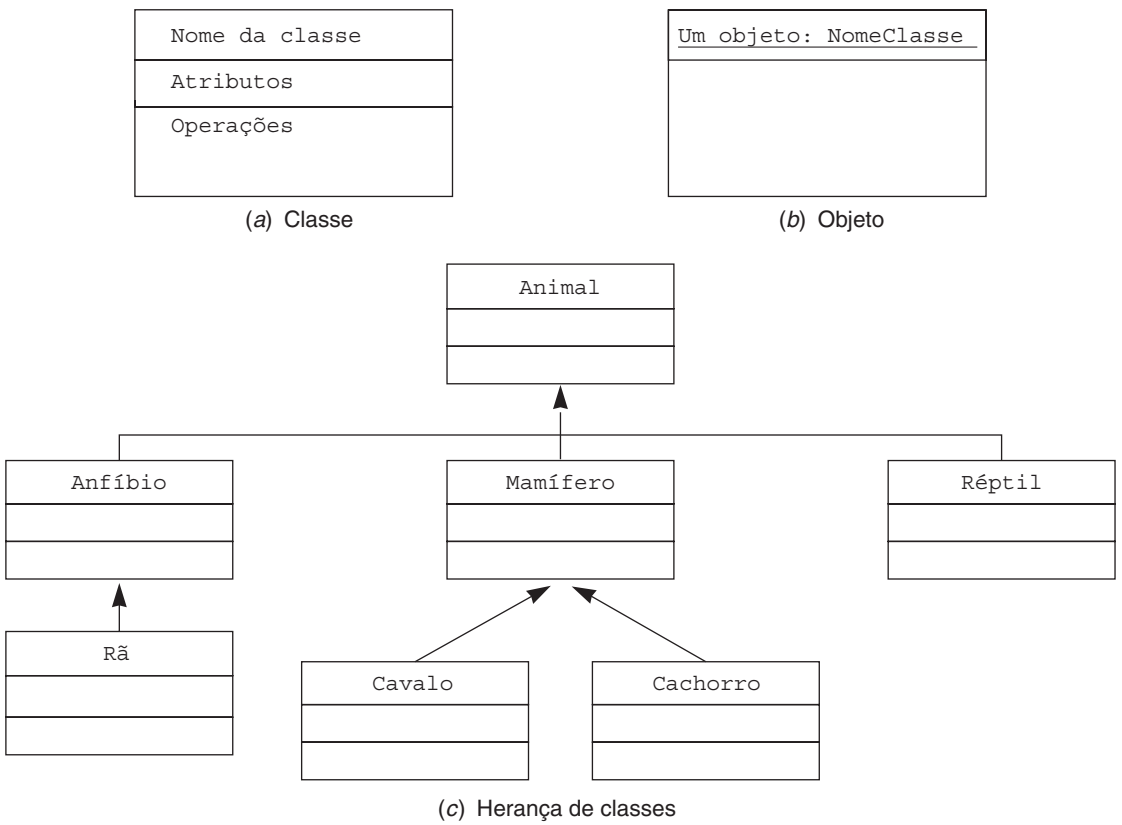
1.12 A LINGUAGEM UNIFICADA DE MODELAGEM (UML 2.0)

UML é uma linguagem gráfica para modelagem de programas de computadores. "Modelagem" significa, como seu nome indica, criar modelos ou representações de "algo" como um plano de uma casa ou similar. UML proporciona um meio de visualizar a organização de alto nível dos programas sem fixar-se nos detalhes do código real.

A linguagem de modelagem é unificada porque está baseada em vários modelos prévios (métodos de Booch, Rumbaugh e Jacobson). Atualmente, UML é adotada pelo OMG (Object Management Group), um consórcio de mais de 1.000 sociedades e universidades ativas no campo de tecnologias orientadas a objetos e dedicado especialmente à unificação de padrões. UML possui uma grande riqueza semântica que a abstrai de numerosos aspectos técnicos e esta é sua grande fortaleza. Por que necessitamos de UML em programação orientada a objetos? Em primeiro lugar, porque UML nasceu como uma ferramenta gráfica e metodológica para implementar análise e projeto orientado a objetos e, em segundo lugar, porque em programas grandes de computador é difícil entender o funcionamento de um programa examinando somente seu código, é necessário ver como as partes se relacionam umas com as outras.

A parte mais importante de UML, pelo menos no nível de iniciação ou médio em programação, reside em um rico conjunto de diagramas gráficos. Diagramas de classes que mostram as relações entre classes, diagramas de objetos que mostram como se relacionam objetos específicos entre si, diagramas de casos de uso que mostram como os usuários de um programa interagem com o programa etc. Quando se modelam classes, UML é, de fato, padrão para representações gráficas. UML não é um processo de desenvolvimento de software, é simplesmente um meio para examinar o software que está sendo desenvolvido. Ainda que, ao ser um padrão UML, possa ser aplicada a qualquer tipo de linguagem de programação, é especialmente adotada pela POO. A Figura 1.27 mostra representações gráficas de classes, objetos e relações de generalização e especialização (herança) entre classes.

Uma breve resenha da UML é a seguinte: em 1994, James Rumbaugh e Grady Booch decidem unir-se para unificar suas notações e métodos: OMT e Booch; em 1995, Juar Jacobson se une à equipe de «os três amigos» na empresa Rational Software; em 1997, é publicada UML 1.0 e, nesse mesmo ano, OMG adota a notação e cria uma comissão (*Tark Forcs*) encarregada de estudar a evolução de UML; em 2003, é apresentada UML 1.5; em 2005, é apresentada a UML 2.0 e, em 2006, Rumbaugh, Booch e Jacobson publicam as novas edições do *Guía de usuario y Manual de referencia*.



(*a*) Classe (*b*) Objeto

(*c*) Herança de classes

Figura 1.27 Representações gráficas de objetos, classes e herança em UML 2.0.

RESUMO

Neste capítulo, é realizada uma breve descrição da organização física de um computador. São descritos os conceitos fundamentais e seus blocos e componentes mais importantes, tais como CPU (UCP), ALU (UAL), VC, microprocessador, memórias RAM, ROM, dispositivos de E/S e de armazenamento (fitas, disquetes, discos rígidos, CDs, DVDs, memórias *flash* com e sem conexão USB etc.).

É feita também uma breve introdução à representação da informação em textos, imagens, sons e dados numéricos. Também é analisada uma primeira introdução à programação orientada a objetos e suas propriedades mais importantes: abstração, encapsulamento de dados, polimorfismo e herança junto às classes e aos objetos.

O capítulo termina com uma breve história e evolução das linguagens de programação C e C++.

REFERÊNCIAS E LEITURAS RECOMENDADAS

BROOKSHEAR, J. Glenn. *Computer science*. 8. ed. Boston: Pearson/Addison-Wesley, 2005.

Um dos melhores livros para aprender a ciência da computação no nível de introdução e médio. Trata todos os temas relacionados com o mundo da programação com uma notável visão acadêmica e científica. Do estudo de algoritmos até linguagens de programação, engenharia de software, base de dados e uma excelente introdução à inteligência artificial e à teoria da computação.

British Standards Institute. *C++ padrão*. Madri: Anaya. Prólogo de Bjarne Stroustrup, inventor de C++ (2005).

Essa obra é o Padrão Internacional da linguagem de programação C++. Versão traduzida do padrão original em inglês pela editora Anaya, contém as especificações completas de C++ padrão, oficialmente conhecida como BS ISO/IEC 14882:1998 com as atualizações (Corrigendum Técnico 1) adotadas entre 1997 e 2001. Obra de referência indispensável para o trabalho profissional em C++. As 1.006 páginas dessa enciclopédia contêm as especificações oficiais. É um livro de consulta para resolver qualquer dúvida de aplicação de sintaxe, especificações, bibliotecas de informações, de classe etc.

JOYANES AGUILAR, Luis. *Fundamentos de programación. Algoritmos, estructuras de datos y objetos*. 3. ed. Madri: McGraw-Hill, 2003.

Livro de referência para a aprendizagem da programação com uma linguagem algorítmica. Livro complementar dessa obra que já completou quinze anos desde a publicação de sua primeira edição.

JOYANES AGUILAR, Luis. *Programación orientada a objetos*. 2. ed. Madri: McGraw-Hill, 2003.

Livro de referência para a aprendizagem da programação orientada a objetos e que foi escrito na segunda metade da década de 1990 quando UML começa a emergir. Agora está em fase de revisão e melhora, esperando que a terceira edição seja publicada em breve.

JOYANES, Luis; SÁNCHEZ, Lucas. *Programación en C++*. Colección Schaum. Madri: McGraw-Hill, 2006.

Livro complementar dessa obra e com um caráter eminentemente teórico-prático. Ainda que possa ser lido e estudado de modo independente, foi escrito para completar de modo prático toda a teoria necessária para a iniciação em algoritmos e estruturas de dados, bem como em programação orientada a objetos. Contém um grande número de exemplos, exercícios e problemas resolvidos.

LAFORE, Robert. *Object-Oriented Programming in C++*. 4. ed. Indianápolis (Indiana): Sams, 2002.

Um dos melhores livros escritos sobre programação orientada a objetos e C++. Seu autor reedita a obra de tempos em tempos, o que permite que o leitor esteja sempre atualizado. Essa edição contém 1.040 páginas e é um dos livros mais completos que se encontra no mercado.

PRATA, Stephen. *C++ Primer plus*. 5. ed. Indianápolis (Indiana): Sams, 2005.

Outro grande livro para aprender a programar em C++ e aprender métodos de orientação a objetos. É também um livro enciclopédico (1.075 páginas) muito completo com uma profusão de exemplos e exercícios complexos resolvidos.

PRIETO ESPINOSA, Alberto; PRIETO CAMPOS, Beatriz. *Conceptos de informática*. Colección Shaum. Madri: McGraw-Hill, 2005.

Excelente livro teórico-prático para conhecer todos os fundamentos de informática e de grande utilidade para a aprendizagem e o domínio da programação.

PRIETO, A. et al. *Introducción a la informática*. 3. ed. Madri: McGraw-Hill, 2005.

Um dos melhores livros de informática para conhecer todos os fundamentos de informática necessários para iniciar-se em carreiras de engenharia e ciências. Leitura imprescindível e consulta obrigatória para aprofundar-se em técnicas de programação tanto no campo de hardware como de software.