1.2 O que É um Computador?

Um *computador* é um dispositivo capaz de realizar cálculos e tomar decisões lógicas com uma velocidade milhões ou mesmo bilhões de vezes mais rápida do que os seres humanos. Por exemplo, muitos dos computadores pessoais de hoje podem realizar dezenas de milhões de operações aritméticas por segundo. Uma pessoa utilizando uma calculadora de mesa poderia levar décadas para realizar o mesmo número de operações que um poderoso computador pessoal pode realizar em segundos. (Aspectos para reflexão: Como você saberia se a pessoa fez as operações corretamente? Como você saberia se o computador fez as operações corretamente?) Os mais rápidos *supercomputadores* de hoje podem realizar centenas de bilhões de operações por segundo — quase tantas operações quanto centenas de pessoas poderiam realizar em um ano! E os computadores que permitem trilhões de instruções por segundo já se encontram em funcionamento em laboratórios de pesquisa.

Os computadores processam *dados* sob o controle de conjuntos de instruções chamados *programas de computador.* Estes programas conduzem o computador através de um conjunto ordenado de ações especificado por pessoas chamadas *programadores de computador.*

Os vários dispositivos (como teclado, tela, discos, memória e unidades de processamento) que constituem um sistema computacional são chamados de *hardware.* Os programas executados em um computador são chamados de *software.* O custo do hardware diminuiu drasticamente nos últimos anos, chegando ao ponto de os computadores pessoais se tornarem uma utilidade. Infelizmente, o custo do desenvolvimento de software tem crescido constantemente, à medida que os programadores desenvolvem aplicações cada vez mais poderosas e complexas, sem serem capazes de fazer as melhorias correspondentes na tecnologia necessária para esse desenvolvimento. Neste livro você aprenderá métodos de desenvolvimento de software que podem reduzir substancialmente seus custos e acelerar o processo de desenvolvimento de aplicações poderosas e de alto nível. Estes métodos incluem *programação estruturada, refinamento passo a passo top-down {descendente), funcionalização* e, finalmente, *programação orientada a objetos.*

1.3 Organização dos Computadores

Independentemente das diferenças no aspecto físico, praticamente todos os computadores podem ser considerados como divididos em seis *unidades lógicas* ou seções. São elas:

1. ***Unidade de entrada (input unit).*** Esta é a seção de "recepção" do computador. Ela obtém as informações (dados e programas de computador) dos vários *dispositivos de entrada (input devices)* e as coloca à disposição de outras unidades para que possam ser processadas. A maior parte das informações é fornecida aos computadores atualmente através de teclados como os de máquinas de escrever.

2. ***Unidade de saída (output unit).*** Esta é a seção de "expedição" do computador. Ela leva as informações que foram processadas pelo computador e as envia aos vários *dispositivos de saída (output devices)* para torná-las disponíveis para o uso no ambiente externo ao computador. A maioria das informações é fornecida pelo computador através de exibição na tela ou impressão em papel.

3. ***Unidade de memória (memory unit).*** Este é a seção de "armazenamento" do computador, com acesso rápido e capacidade relativamente baixa. Ela conserva as informações que foram fornecidas através da unidade de entrada para que possam estar imediatamente disponíveis para o processamento quando se fizer necessário. A unidade de memória também conserva as informações que já foram processadas até que sejam enviadas para os dispositivos de saída pela unidade de saída. Freqüentemente a unidade de memória é chamada de *memória (memory), memória principal* ou *memória primária (primary memory).*

4. ***Unidade aritmética e lógica (arithmetic and logic unit, ALU).*** Esta é a seção de "fabricação" do computador. Ela é a responsável pela realização dos cálculos como adição, subtração, multiplicação e divisão. Ela contém os mecanismos de decisão que permitem ao computador, por exemplo, comparar dois itens da unidade de memória para determinar se são iguais ou não.

5. ***Unidade central de processamento, UCP (central processing unit, CPU).*** Esta é a seção "administrativa" do computador. Ela é o coordenador do computador e o responsável pela supervisão do funcionamento das outras seções. A CPU informa à unidade de entrada quando as informações devem ser lidas na unidade de memória, informa à ALU quando as informações da unidade de memória devem ser utilizadas em cálculos e informa à unidade de saída quando as informações devem ser enviadas da unidade de memória para determinados dispositivos de saída.

6. ***Unidade de memória secundária (secondary storage unit).*** Esta é a seção de "armazenamento" de alta capacidade e de longo prazo do computador. Os programas ou dados que não estiverem sendo usados ativamente por outras unidades são colocados normalmente em dispositivos de memória secundária (como discos) até que sejam outra vez necessários, possivelmente horas, dias, meses ou até mesmo anos mais tarde.

1.4 Processamento em Lotes (Batch Processing), Multiprogramação e Tempo Compartilhado (Timesharing)

Os primeiros computadores eram capazes de realizar apenas um trabalho ou tarefa de cada vez. Esta forma de funcionamento de computadores é chamada freqüentemente de processamento em lotes de usuário único. O computador executa um único programa de cada vez enquanto processa dados em gru¬pos ou lotes (batches). Nesses primeiros sistemas, geralmente os usuários enviavam suas tarefas ao centro computacional em pilhas de cartões perfurados. Freqüentemente os usuários precisavam esperar horas antes que as saídas impressas fossem levadas para seus locais de trabalho.

À medida que os computadores se tornaram mais poderosos, tornou-se evidente que o processamen¬to em lotes de usuário único raramente utilizava com eficiência os recursos do computador. Em vez disso, imaginava-se que muitos trabalhos ou tarefas poderiam ser executados de modo a compartilhar os re¬cursos do computador e assim conseguir utilizá-lo melhor. Isto é chamado multiprogramação. A multiprogramação envolve as "operações" simultâneas de muitas tarefas do computador — o computa¬dor compartilha seus recursos entre as tarefas que exigem sua atenção. Com os primeiros sistemas de multiprogramação, os usuários ainda enviavam seus programas em pilhas de cartões perfurados e espe¬ravam horas ou dias para obter os resultados.

Nos anos 60, vários grupos de indústrias e universidades foram pioneiros na utilização do conceito de timesharing (tempo compartilhado). Timesharing é um caso especial de multiprogramação no qual os usuários podem ter acesso ao computador através de dispositivos de entrada/saída ou terminais. Em um sistema computacional típico de timesharing, pode haver dezenas de usuários compartilhando o computador ao mesmo tempo. Na realidade o computador não atende a todos os usuários simultanea¬mente. Em vez disso, ele executa uma pequena parte da tarefa de um usuário e então passa a fazer a tarefa do próximo usuário. O computador faz isto tão rapidamente que pode executar o serviço de cada usuário várias vezes por segundo. Assim, parece que as tarefas dos usuários estão sendo executadas simultaneamente.

1.5 Computação Pessoal, Computação Distribuída e Computação Cliente/Servidor

Em 1997, a Apple Computer tornou popular o fenômeno da computação pessoal. Inicialmente, isto era um sonho de quem a tinha como um *hobby.* Computadores tornaram-se suficientemente baratos para serem comprados para uso pessoal ou comercial. Em 1981, a IBM, a maior vendedora de computadores do mundo, criou o IBM PC (Personal Computer, computador pessoal). Do dia para a noite, literalmente, a computação pessoal se tornou comum no comércio, na indústria e em organizações governamentais.

Mas esses computadores eram unidades "autônomas" — as pessoas faziam suas tarefas em seus próprios equipamentos e então transportavam os discos de um lado para outro para compartilhar as informações. Embora os primeiros computadores pessoais não fossem suficientemente poderosos para serem compartilhados por vários usuários, esses equipamentos podiam ser ligados entre si em redes de computadores, algumas vezes através de linhas telefônicas e algumas vezes em redes locais de organizações. Isto levou ao fenômeno da *computação distribuída,* na qual a carga de trabalho computacional de uma organização, em vez de ser realizada exclusivamente em uma instalação central de informática, é distribuída em redes para os locais (sites) nos quais o trabalho real da organização é efetuado. Os computadores pessoais eram suficientemente poderosos para manipular as exigências computacionais de cada usuário em particular e as tarefas básicas de comunicações de passar as informações eletronicamente de um lugar para outro.

Os computadores pessoais mais poderosos de hoje são tão poderosos quanto os equipamentos de milhões de dólares de apenas uma década atrás. Os equipamentos desktop (computadores de mesa) mais poderosos — chamados *workstations* ou *estações de trabalho* — fornecem capacidades enormes a usuários isolados. As informações são compartilhadas facilmente em redes de computadores onde alguns deles, os chamados *servidores de arquivos (file servers),* oferecem um depósito comum de programas e dados que podem ser usados pelos computadores *clientes (clients)* distribuídos ao longo da rede, daí o termo *computação cliente/servidor.* O C e o C ++ tornaram-se as linguagens preferidas de programação para a criação de software destinado a sistemas operacionais, redes de computadores e aplicações distribuídas cliente/servidor.

1.6 Linguagens de Máquina, Linguagens Assembly e Linguagens de Alto Nível

Os programadores escrevem instruções em várias linguagens de programação, algumas entendidas diretamente pelo computador e outras que exigem passos intermediários de *tradução.* Centenas de linguagens computacionais estão atualmente em uso. Estas podem ser divididas em três tipos gerais:

1. Linguagens de máquina
2. Linguagens assembly
3. Linguagens de alto nível

Qualquer computador pode entender apenas sua própria *linguagem de máquina.* A linguagem de máquina é a "linguagem natural" de um determinado computador. Ela está relacionada intimamente com o projeto de hardware daquele computador. Geralmente as linguagens de máquina consistem em strings de números (reduzidos em última análise a ls e Os) que dizem ao computador para realizar uma de suas operações mais elementares de cada vez. As linguagens de máquina são *dependentes de máquina (não-padronizadas,* ou *machine dependent),* i.e., uma determinada linguagem de máquina só pode ser usada com um tipo de computador. As linguagens de máquina são complicadas para os humanos, como se pode ver no trecho seguinte de um programa em linguagem de máquina que adiciona o pagamento de horas extras ao salário base e armazena o resultado no pagamento bruto.

+1300042774
+1400593419
+1200274027

À medida que os computadores se tornaram mais populares, ficou claro que a programação em linguagem de máquina era simplesmente muito lenta e tediosa para a maioria dos programadores. Em vez de usar strings de números que os computadores podiam entender diretamente, os programadores começaram a usar abreviações parecidas com palavras em inglês para representar as operações elementares de um computador. Estas abreviações formaram a base das *linguagens assembly.* Foram desenvolvidos *programas tradutores,* chamados *assemblers,* para converter programas em linguagem assembly para linguagem de máquina na velocidade ditada pelo computador. O trecho de um programa em linguagem assembly a seguir também soma o pagamento de horas extras ao salário base e armazena o resultado em pagamento bruto, porém isto é feito de uma forma mais clara do que o programa equivalente em linguagem de máquina.

LOAD BASE
ADD EXTRA
STORE BRUTO

O uso do computador aumentou rapidamente com o advento das linguagens assembly, mas elas ainda exigiam muitas instruções para realizar mesmo as tarefas mais simples. Para acelerar o processo de programação, foram desenvolvidas *linguagens de alto nível,* nas quais podiam ser escritas instruções simples para realizar tarefas

fundamentais. Os programas tradutores que convertiam programas de linguagem de alto nível em linguagem de máquina são chamados *compiladores.* As linguagens de alto nível permitem aos programadores escrever instruções que se parecem com o idioma inglês comum e contêm as notações matemáticas normalmente usadas. Um programa de folha de pagamento em uma linguagem de alto nível poderia conter uma instrução como esta:

Bruto = Base + Extra

Obviamente, as linguagens de alto nível são muito mais desejáveis do ponto de vista do programador do que as linguagens de máquina ou assembly. O C e o C++ estão entre as linguagens de alto nível mais poderosas e mais amplamente usadas.

1.7 A História do C

O C foi desenvolvido a partir de duas linguagens anteriores, o BCPL e o B. O BCPL foi desenvolvido em 1967 por Martin Richards como uma linguagem para escrever software de sistemas operacionais e compiladores. Ken Thompson modelou muitos recursos em sua linguagem B com base em recursos similares do BCPL e usou o B para criar as primeiras versões do sistema operacional UNIX nas instalações do Bell Laboratories em 1970, em um computador DEC PDP-7. Tanto o BCPL como o B eram linguagens "sem tipos" ("typeless") — todos os itens de dados ocupavam uma "palavra" na memória e a responsabilidade de lidar com um item de dados como um número inteiro ou real, por exemplo, recaía sobre os ombros do programador.

A linguagem C foi desenvolvida a partir do B por Dennis Ritchie, do Bell Laboratories, e implementada originalmente em um computador DEC PDP-11, em 1972. De início o C se tornou amplamente conhecido como a linguagem de desenvolvimento do sistema operacional UNIX. Hoje em dia, praticamente todos os grandes sistemas operacionais estão escritos em C e/ou C++ . Ao longo das duas últimas décadas, o C ficou disponível para a maioria dos computadores. O C independe do hardware. Elaborando um projeto cuidadoso, é possível escrever programas em C que sejam portáteis para a maioria dos computadores. O C usa muitos dos importantes conceitos do BCPL e do B ao mesmo tempo que adiciona tipos de dados e outros recursos poderosos.

No final da década de 70, o C evoluiu e chegou ao que se chama agora de "C tradicional". A publicação em 1978 do livro de Kernighan e Ritchie, *The C Programming Language,* fez com que essa linguagem recebesse muita atenção. Esta publicação se tornou um dos livros de informática mais bem-sucedidos de todos os tempos.

A rápida expansão do C em vários tipos de computadores (algumas vezes chamados de *plataformas de hardware)* levou a muitas variantes. Elas eram similares, mas freqüentemente incompatíveis. Isto foi um problema sério para os desenvolvedores de programas que precisavam criar um código que fosse executado em várias plataformas. Ficou claro que era necessário uma versão padrão do C. Em 1983, foi criado o comitê técnico X3J11 sob o American National Standards Committee on Computers and Information Processing (X3) para "fornecer à linguagem uma definição inequívoca e independente de equipamento". Em 1989, o padrão foi aprovado. O documento é conhecido como ANSI/ISO 9899:1990. Pode-se pedir cópias desse documento para o American National Standards Institute, cujo endereço consta no Prefácio a este texto. A segunda edição do livro de Kernighan e Ritchie, publicada em 1988, reflete esta versão, chamada ANSI C, agora usada em todo o mundo (Ke88).

Dicas de portabilidade 1.1

Como o C é uma linguagem independente de hardware e amplamente disponível, as aplicações escritas em C podem ser executadas com pouca ou nenhuma modificação em uma grande variedade de sistemas computacionais.

1.8 A Biblioteca Padrão (Standard Library) do C

Como você aprenderá no Capítulo 5, os programas em C consistem em módulos ou elementos chamados *funções.* Você pode programar todas as funções de que precisa para formar um programa C, mas a maioria dos programadores C tira proveito de um excelente conjunto de funções chamado *C Standard Library* (Biblioteca Padrão do C). Dessa forma, há na realidade duas partes a serem aprendidas no "mundo" do C. A primeira é a linguagem C em si, e a segunda é como usar as funções do C Standard Library. Ao longo deste livro, analisaremos muitas dessas funções. O Apêndice A (condensado e adaptado do documento padrão do ANSI C) relaciona todas as funções disponíveis na biblioteca padrão do C. A leitura do livro de Plauger (P192) é obrigatória para os programadores que necessitam de um entendimento profundo das funções da biblioteca, de como implementá-las e como usá-las para escrever códigos portáteis.

Neste curso você será estimulado a usar o *método dos blocos de construção* para criar programas. Evite reinventar a roda. Use os elementos existentes — isto é chamado *reutilização de software* e é o segredo do campo de desenvolvimento da programação orientada a objetos. Ao programar em C, você usará normalmente os seguintes blocos de construção:

- Funções da C Standard Library (biblioteca padrão)
- Funções criadas por você mesmo
- Funções criadas por outras pessoas e colocadas à sua disposição

A vantagem de criar suas próprias funções é que você saberá exatamente como elas funcionam. Você poderá examinar o código C. A desvantagem é o esforço demorado que se faz necessário para projetar e desenvolver novas funções.

Usar funções existentes evita reinventar a roda. No caso das funções standard do ANSI, você sabe que elas foram desenvolvidas cuidadosamente e sabe que, por estar usando funções disponíveis em praticamente todas as implementações do ANSI C, seus programas terão uma grande possibilidade de serem portáteis.

Dica de desempenho 1.1

Usar as funções da biblioteca standard do C em vez de você escrever suas próprias versões similares pode melho¬rar o desempenho do programa porque essas funções foram desenvolvidas cuidadosamente por pessoal eficiente.

Dicas de portabilidade 1.2

Usar as funções da biblioteca padrão do C em vez de escrever suas próprias versões similares pode melhorar a porta¬bilidade do programa porque essas funções estão colocadas em praticamente todas as implementações do ANSI C.

1.9 Outras Linguagens de Alto Nível

Centenas de linguagens de alto nível foram desenvolvidas, mas apenas algumas conseguiram grande aceitação. O *FORTRAN* (FORmula TRANslator) foi desenvolvido pela IBM entre 1954 e 1957 para ser usado em aplicações científicas e de engenharia que exigem cálculos matemáticos complexos. O FORTRAN ainda é muito usado.

O *COBOL* (COmmon Business Oriented Language) foi desenvolvido em 1959 por um grupo de fabricantes de computadores e usuários governamentais e industriais. O COBOL é usado principalmente para aplicações comerciais que necessitam de uma manipulação precisa e eficiente de grandes volumes de dados. Hoje em dia, mais de metade de todo o software comercial ainda é programada em COBOL. Mais de um milhão de pessoas estão empregadas como programadores de COBOL.

O *Pascal* foi desenvolvido quase ao mesmo tempo que o C. Ele destinava-se ao uso acadêmico. Falaremos mais sobre o Pascal na próxima seção.

1.10 Programação Estruturada

Durante os anos 60, enormes esforços para o desenvolvimento de software encontraram grandes dificuldades. Os cronogramas de desenvolvimento de software normalmente estavam atrasados, os custos superavam em muito os orçamentos e os produtos finais não eram confiáveis. As pessoas começaram a perceber que o desenvolvimento era uma atividade muito mais complexa do que haviam imaginado. A atividade de pesquisa dos anos 60 resultou na evolução da *programação estruturada* — um método disciplinado de escrever programas claros, nitidamente corretos e fáceis de modificar. Os Capítulos 3 e 4 descrevem os fundamentos da programação estruturada. O restante do texto analisa o desenvolvimento de programas em C estruturados.

Um dos resultados mais tangíveis dessa pesquisa foi o desenvolvimento da linguagem Pascal de programação pelo Professor Nicklaus Wirth em 1971. O Pascal, que recebeu este nome em homenagem ao matemático e filósofo Blaise Pascal, que viveu no século XVII, destinava-se ao ensino da programação estruturada em ambientes acadêmicos e se tornou rapidamente a linguagem preferida para a introdução à programação em muitas universidades. Infelizmente, a linguagem carecia de muitos recursos necessários para torná-la útil em aplicações comerciais, industriais e governamentais, e portanto não foi amplamente aceita nesses ambientes. Possivelmente a história registra que a grande importância do Pascal foi sua escolha para servir de base para a linguagem de programação *Ada.*

A linguagem Ada foi desenvolvida sob a responsabilidade do Departamento de Defesa dos EUA (United States Department of Defense, ou DOD) durante os anos 70 e início dos anos 80. Centenas de linguagens diferentes estavam sendo usadas para produzir os imensos sistemas de comando e controle de software do DOD. O DOD desejava uma única linguagem que pudesse atender a suas necessidades. O Pascal foi escolhido como base, mas a linguagem Ada final é muito diferente do Pascal. A linguagem Ada recebeu este nome em homenagem a Lady Ada Lovelace, filha do poeta Lorde Byron. De uma maneira geral, Lady Lovelace é considerada a primeira pessoa do mundo a escrever um programa de computador, no início do século XIX. Uma característica importante da linguagem Ada é chamada *multitarefa (multitasking);* isto permite aos programadores especificarem a ocorrência simultânea de muitas atividades. Outras linguagens de alto nível amplamente usadas que analisamos — incluindo o C e o C++ — permitem ao programador escrever programas que realizem apenas uma atividade. Saberemos no futuro se a linguagem Ada conseguiu atingir seus objetivos de produzir software confiável e reduzir substancialmente os custos de desenvolvimento e manutenção de software.

1.11 Os Fundamentos do Ambiente C

Todos os sistemas C são constituídos geralmente de três partes: o ambiente, a linguagem e a C Standard Library. A análise a seguir explica o ambiente típico de desenvolvimento do C, mostrado na Figura 1.1.

Normalmente os programas em C passam por seis fases para serem executados (Figura 1.1). São elas: *edição, pré-processamento, compilação, linking (ligação), carregamento* e *execução.* Concentrar-nos-emos aqui em um sistema típico do C baseado em UNIX. Se você não estiver usando um sistema UNIX, consulte o manual de seu sistema ou pergunte ao seu professor como realizar estas tarefas em seu ambiente.

A primeira fase consiste na edição de um arquivo. Isto é realizado com um *programa editor.* O programador digita um programa em C com o editor e faz as correções necessárias. O programa é então armazenado em um dispositivo de armazenamento secundário como um disco. Os nomes de programas em C devem ter a extensão **.c.** Dois editores muito usados em sistemas UNIX são o **vi** e o **emacs.** Os pacotes de software C/C++ como o Borland C++ para IBM PCs e compatíveis e o Symantec C++ para Apple Macintosh possuem editores embutidos que se adaptam perfeitamente ao ambiente de programação. Partimos do princípio de que o leitor sabe editar um programa.

A seguir, o programador emite o comando de *compilar* o programa. O compilador traduz o programa em C para o código de linguagem de máquina (também chamado de *código-objeto).* Em um sistema C, um programa *pré-processador é* executado automaticamente antes de a fase de tradução começar. O pré-processador C obedece a comandos especiais chamados *diretivas do pré-processador* que indicam que devem ser realizadas determinadas manipulações no programa antes da compilação. Estas manipulações consistem normalmente em incluir outros arquivos no arquivo a ser compilado e substituir símbolos especiais por texto de programa. As diretivas mais comuns do pré-processador são analisadas nos primeiros capítulos; uma análise detalhada de todos os recursos do pré-processador está presente no Capítulo 13. O pré-processador é ativado automaticamente pelo compilador antes de o programa ser convertido para linguagem de máquina.

A quarta fase é chamada *linking.* Normalmente os programas em C contêm referências a funções definidas em outros locais, como nas bibliotecas padrão ou nas bibliotecas de um grupo de programadores que estejam trabalhando em um determinado projeto. Assim, o código-objeto produzido pelo compilador C contém normalmente "lacunas" devido à falta dessas funções. Um *linker* faz a ligação do código-objeto com o código das funções que estão faltando para produzir uma *imagem executável* (sem a falta de qualquer parte). Em um sistema típico baseado em UNIX, o comando para compilar e linkar um programa é **cc.** Por exemplo, para compilar e linkar um programa chamado **bemvindo. c** digite

```
cc bemvindo.c
```

no prompt do UNIX e pressione a tecla Return (ou Enter). Se o programa for compilado e linkado corretamente, será produzido um arquivo chamado **a.out.** Este arquivo é a imagem executável de nosso programa **bemvindo. c.**
A quinta fase é chamada *carregamento.* Um programa deve ser colocado na memória antes que possa ser executado pela primeira vez. Isto é feito pelo *carregador (rotina de carga ou loader),* que apanha a imagem executável do disco e a transfere para a memória.

Finalmente, o computador, sob o controle de sua CPU, executa as instruções do programa, uma após a outra. Para carregar e executar o programa em um sistema UNIX digitamos **a.out** no prompt do UNIX e apertamos a tecla Return.

A maioria dos programas em C recebe ou envia dados. Determinadas funções do C recebem seus dados de entrada a partir do ***stdin*** (o *dispositivo padrão de entrada,* ou *standard input device)* que normalmente é definido como o teclado, mas que pode estar associado a outro dispositivo. Os dados são enviados para o **stdout** (o *dispositivo padrão de saída,* ou *standard output device),* que normalmente é a tela do computador, mas que pode estar associado a outro dispositivo. Quando dizemos que um programa fornece um resultado, normalmente queremos dizer que o resultado é exibido na tela.

Há também um *dispositivo padrão de erros (standard error device)* chamado ***stderr.*** O dispositivo **stderr** (normalmente associado à tela) é usado para exibir mensagens de erro. É comum não enviar os dados regu-lares de saída, i.e., **stdout** para a tela e manter **stderr** associado a ela para que o usuário possa ser informado imediatamente dos erros.