

ALGORITMOS E ESTRUTURAS DE DADOS



UNICAMP

UNIVERSIDADE ESTADUAL DE CAMPINAS

Reitor

ANTONIO JOSÉ DE ALMEIDA MEIRELLES

Coordenadora Geral da Universidade

MARIA LUIZA MORETTI



Conselho Editorial

Presidente

EDWIGES MARIA MORATO

CARLOS RAUL ETULAIN – CICERO ROMÃO RESENDE DE ARAUJO
DIRCE DJANIRA PACHECO E ZAN – FREDERICO AUGUSTO GARCIA FERNANDES
IARA BELELI – MARCO AURÉLIO CREMASCO – PEDRO CUNHA DE HOLANDA
SÁVIO MACHADO CAVALCANTE – VERÓNICA ANDREA GONZÁLEZ-LÓPEZ

Hélio Pedrini

**ALGORITMOS
E ESTRUTURAS DE DADOS**
Conceitos e aplicações

EDITOR A UNICAMP

FICHA CATALOGRÁFICA ELABORADA PELO
SISTEMA DE BIBLIOTECAS DA UNICAMP
DIVISÃO DE TRATAMENTO DA INFORMAÇÃO
Bibliotecária: Maria Lúcia Nery Dutra de Castro – CRB-8ª / 1724

Sa32e Pedrini, Hélio
 Algoritmos e estruturas de dados : conceitos e aplicações / Hélio Pedrini –
 Campinas, SP : Editora da Unicamp, 2024.

1. Algoritmos. 2. Estruturas de dados (Computação) 3. Linguagem de pro-
gramação (Computadores) 4. Ciência da computação. I. Título.

CDD – 518.1
 – 005.73
 – 005.13
 – 004

ISBN 978-85-268-1626-8

Copyright © by Hélio Pedrini
Copyright © 2024 by Editora da Unicamp

Opiniões, hipóteses e conclusões ou recomendações expressas
neste livro são de responsabilidade do autor e não
necessariamente refletem a visão da Editora da Unicamp.

Direitos reservados e protegidos pela lei 9.610 de 19.2.1998.
É proibida a reprodução total ou parcial sem autorização,
por escrito, dos detentores dos direitos.

Foi feito o depósito legal.

Direitos reservados à

Editora da Unicamp
Rua Sérgio Buarque de Holanda, 421 – 3ª andar
Campus Unicamp
CEP 13083-859 – Campinas – SP – Brasil
Tel./Fax: (19) 3521-7718 / 7728
www.editoraunicamp.com.br – vendas@editora.unicamp.br

Sumário

Prefácio	13
I Fundamentos de programação	15
1 Introdução	17
1.1 Algoritmos e programas	17
1.2 Organização básica de computadores	17
1.3 Linguagem de programação C	19
1.4 Estrutura de um programa em linguagem C	21
1.5 Indentação e comentários	22
1.6 Exercícios	23
2 Variáveis e constantes	25
2.1 Variáveis	25
2.1.1 Tipos básicos de variáveis	25
2.1.2 Nomes de variáveis	27
2.1.3 Variáveis em registradores	28
2.2 Constantes	28
2.3 Exercícios	29
3 Entrada e saída	31
3.1 Escrita de dados	31
3.2 Leitura de dados	33
3.3 Exercícios	37
4 Operadores	39
4.1 Operador de atribuição	39
4.2 Operadores aritméticos	40
4.3 Conversão de valores entre tipos diferentes	40
4.4 Operadores de incremento e decremento	42
4.5 Atribuições simplificadas	43
4.6 Operadores relacionais	44
4.7 Operadores lógicos	45
4.8 Operadores bit-a-bit	46
4.9 Operador ternário ? :	47
4.10 Operadores de ponteiros	48
4.11 Precedência de operadores	48
4.12 Biblioteca matemática	48
4.13 Exercícios	49

5	Comandos condicionais	53
5.1	Comando <code>if</code>	53
5.2	Comando <code>if-else</code>	54
5.3	Comando <code>switch</code>	56
5.4	Exercícios	58
6	Comandos de repetição	61
6.1	Comando <code>while</code>	61
6.2	Comando <code>do-while</code>	62
6.3	Comando <code>for</code>	63
6.4	Comandos <code>break</code> e <code>continue</code>	66
6.5	Comando <code>exit</code>	66
6.6	Comando <code>goto</code>	67
6.7	Laços aninhados	68
6.8	Exercícios	69
7	Vetores e matrizes	73
7.1	Vetores	73
7.2	Matrizes	75
7.3	Inicialização de vetores e matrizes	78
7.4	Representação de matrizes como vetores	79
7.5	Exercícios	80
8	Cadeias de caracteres	85
8.1	Cadeias de caracteres	85
8.2	Biblioteca para manipulação de cadeias de caracteres	89
8.3	Exercícios	90
9	Funções	93
9.1	Declaração de funções	93
9.2	Função <code>main</code>	94
9.3	Tipo <code>void</code>	95
9.4	Protótipos de funções	95
9.5	Escopo de variáveis	96
9.5.1	Modificador <code>auto</code>	98
9.5.2	Modificador <code>extern</code>	99
9.5.3	Modificador <code>register</code>	99
9.5.4	Modificador <code>static</code>	100
9.5.5	Modificador <code>volatile</code>	100
9.6	Vetores e funções	101
9.7	Matrizes e funções	102
9.8	Macros	104
9.9	Exercícios	106
10	Ponteiros	109
10.1	Declaração de ponteiros	109
10.2	Passagem de parâmetros para função por valor e por referência	112
10.3	Aritmética de ponteiros	114
10.4	Ponteiros para vetores	115
10.5	Ponteiros para cadeias de caracteres	118

10.6	Vetores de ponteiros	120
10.7	Ponteiros para funções	120
10.8	Alocação dinâmica de memória	121
10.9	Ponteiros de ponteiros	123
10.10	Exercícios	126
11	Tipos enumerados e estruturados	131
11.1	Tipos enumerados	131
11.2	Redefinição de tipos	132
11.3	Tipo estrutura	132
11.3.1	Vetor de estruturas	134
11.3.2	Ponteiros para estruturas	135
11.3.3	Estruturas aninhadas	136
11.4	Tipo união	137
11.5	Exercícios	139
12	Arquivos	143
12.1	Tipos de arquivos	143
12.2	Arquivos textos	143
12.3	Arquivos binários	148
12.4	Remoção de arquivos	154
12.5	Exercícios	155
II	Técnicas de programação e estruturas de dados avançadas	157
13	Recursividade	159
13.1	Algoritmos recursivos	159
13.2	Enumeração exaustiva	170
13.3	Técnica de retrocesso	171
13.3.1	Problema das n damas	171
13.3.2	Passeio do cavalo	173
13.3.3	Caminho em um labirinto	176
13.3.4	Soma de subconjuntos	178
13.4	Exercícios	179
14	Análise de complexidade	183
14.1	Análise de complexidade	183
14.2	Comportamento assintótico de funções	184
14.2.1	Notação O	185
14.2.2	Notação Ω	186
14.2.3	Notação Θ	187
14.2.4	Notação o	188
14.2.5	Notação ω	188
14.2.6	Análise de algoritmos iterativos e recursivos	189
14.3	Exercícios	196
15	Listas ligadas	199
15.1	Listas ligadas simples	199
15.2	Listas ligadas simples com nó cabeça	210

15.3	Listas ligadas simples circulares	213
15.4	Listas ligadas simples circulares com nó cabeça	217
15.5	Listas duplamente ligadas	220
15.6	Listas circulares duplamente ligadas	223
15.7	Listas generalizadas	223
15.8	Exercícios	228
16	Pilhas	231
16.1	Fundamentos	231
16.2	Operações básicas	231
16.3	Implementação de pilha com vetor	232
16.4	Implementação de pilha com lista ligada	234
16.5	Aplicações de pilhas	235
16.5.1	Balanceamento de parênteses e colchetes	235
16.5.2	Avaliação de expressões em notação posfixa	236
16.5.3	Conversão de notação infixa para posfixa	238
16.5.4	Controle de execução de um programa	239
16.6	Exercícios	240
17	Filas	243
17.1	Fundamentos	243
17.2	Operações básicas	243
17.3	Implementação de fila com vetor	243
17.4	Implementação de fila com lista ligada	246
17.5	Filas de prioridades	249
17.6	Exercícios	256
18	Ordenação e busca	259
18.1	Algoritmos de ordenação	259
18.1.1	Ordenação por trocas	259
18.1.2	Ordenação por seleção	261
18.1.3	Ordenação por inserção	262
18.1.4	<i>Shellsort</i>	263
18.1.5	<i>Mergesort</i>	264
18.1.6	<i>Quicksort</i>	266
18.1.7	<i>Heapsort</i>	269
18.1.8	Ordenação em tempo linear	269
18.2	Algoritmos de busca	277
18.2.1	Busca sequencial	277
18.2.2	Busca binária	277
18.3	Exercícios	279
19	Tabelas de espalhamento	283
19.1	Problema de busca	283
19.2	Tabelas de espalhamento	283
19.3	Função de espalhamento	283
19.3.1	Método da divisão	284
19.3.2	Meio do quadrado	285
19.3.3	Método da multiplicação	285
19.3.4	Particionamento	286

19.4	Tratamento de colisões	286
19.4.1	Endereçamento aberto	287
19.4.2	Endereçamento fechado	293
19.4.3	Redistribuição de chaves	293
19.4.4	Espalhamento perfeito	293
19.5	Exercícios	295
20	Árvores	299
20.1	Fundamentos	299
20.2	Árvores binárias	300
20.2.1	Operações básicas em árvores binárias	300
20.2.2	Percursos em árvores binárias	305
20.2.3	Conversão de árvore geral em árvore binária	309
20.2.4	Conversão de floresta em árvore binária	309
20.3	Árvore binária de busca	309
20.4	Árvore de busca AVL	316
20.4.1	Fundamentos	316
20.4.2	Operações básicas	317
20.5	Árvore de busca rubro-negra	324
20.5.1	Fundamentos	324
20.5.2	Operações básicas	325
20.6	Árvore de difusão	337
20.6.1	Fundamentos	338
20.6.2	Operações básicas	338
20.7	Árvore B	344
20.7.1	Fundamentos	344
20.7.2	Operações básicas	347
20.8	Árvore B*	357
20.9	Árvore B ⁺	357
20.10	Exercícios	359
21	Grafos	365
21.1	Fundamentos	365
21.2	Problema das pontes de Königsberg	371
21.3	Problema da coloração de grafos	371
21.4	Representações de grafos	373
21.4.1	Matriz de adjacências	373
21.4.2	Lista de adjacências	373
21.5	Percursos em grafos	374
21.5.1	Busca em profundidade	374
21.5.2	Busca em largura	375
21.6	Ordenação topológica	376
21.7	Árvore geradora mínima	377
21.7.1	Algoritmo de Kruskal	377
21.7.2	Algoritmo de Prim	377
21.8	Caminhos mínimos	379
21.8.1	Algoritmo de Dijkstra	379
21.8.2	Algoritmo de Bellman-Ford	381
21.9	Implementação de grafos e suas operações	383
21.9.1	Matriz de adjacências	383
21.9.2	Lista de adjacências	390

21.10 Exercícios	393
----------------------------	-----

III Informações suplementares 399

A Funções elementares	401
A.1 Funções piso e teto	401
A.1.1 Algumas propriedades da função piso	401
A.1.2 Algumas propriedades da função teto	401
A.2 Monotonicidade de funções	402
A.3 Funções pares e ímpares	402
B Somatórios e produtórios	403
B.1 Somatórios	403
B.1.1 Algumas propriedades de somatórios	403
B.1.2 Fórmulas explícitas para alguns somatórios	403
B.2 Produtórios	404
B.2.1 Algumas propriedades de produtórios	404
C Exponenciação e logaritmos	405
C.1 Exponenciação	405
C.2 Logaritmos	405
D Sistemas de numeração	407
D.1 Notação posicional	407
D.1.1 Sistema decimal	408
D.1.2 Sistema binário	408
D.1.3 Sistema octal	408
D.1.4 Sistema hexadecimal	408
D.2 Conversão entre bases numéricas	409
D.2.1 Conversão de base qualquer para decimal	409
D.2.2 Conversão de decimal para base qualquer	409
D.2.3 Conversão entre sistemas binário e octal	411
D.2.4 Conversão entre sistemas binário e hexadecimal	411
E Representação de números	413
E.1 Representação de números inteiros	413
E.1.1 Sinal-magnitude	413
E.1.2 Complemento de 1	413
E.1.3 Complemento de 2	414
E.1.4 Excesso- N	414
E.1.5 Comparação entre representações	414
E.2 Representação de números em ponto flutuante	414
E.2.1 Padrão IEEE 754	414
F Indução matemática	417
F.1 Princípio da indução matemática	417
F.2 Exemplos	417
G Codificação de caracteres	421

G.1	ASCII	421
G.2	EBCDIC	421
G.3	Unicode	421
Referências bibliográficas		425
Índice remissivo		437

Prefácio

Os avanços científicos e tecnológicos têm permitido o desenvolvimento de soluções eficazes e eficientes para uma variedade de problemas. Desde a sua concepção, os computadores evoluíram significativamente, auxiliando os seres humanos em suas atividades pessoais e profissionais.

Os desafios associados ao aumento contínuo da complexidade dos sistemas computacionais demandam a proposição de modelos, abordagens e processos capazes de organizar e representar os principais conceitos para a resolução de uma tarefa. Nesse sentido, os algoritmos e as estruturas de dados têm desempenhado um papel fundamental na construção de programas utilizados na solução de problemas e no apoio à tomada de decisões em diferentes domínios de conhecimento.

Este livro tem como objetivo apresentar os fundamentos de algoritmos e de estruturas de dados. Os códigos são descritos de maneira clara e abrangente, buscando-se analisá-los do ponto de vista de custo para sua implementação e ilustrá-los por meio de vários exemplos e exercícios para auxiliar a compreensão de seus aspectos teóricos e práticos. Os tópicos selecionados contemplam as principais características, operações e funcionalidades das estruturas de dados, de modo que os leitores possam elaborar seus próprios algoritmos e adaptá-los a aplicações específicas de interesse.

O texto está organizado em 21 capítulos e 7 apêndices. Os 12 primeiros capítulos, que compõem a primeira parte do livro, apresentam conceitos básicos para a construção de algoritmos e estruturas de dados elementares. Os 9 capítulos seguintes, que formam a segunda parte do livro, abordam princípios de análise de algoritmos e estruturas de dados avançadas. Os apêndices, que compõem a terceira parte do livro, complementam as informações discutidas nos capítulos para facilitar o entendimento dos temas abordados.

A linguagem de programação C é utilizada na implementação dos códigos apresentados. A linguagem provê mecanismos para a construção de programas de forma flexível, versátil e estruturada, permitindo o uso eficiente dos recursos computacionais disponíveis. A primeira parte do livro introduz os principais recursos da linguagem C, com o propósito de fornecer os fundamentos necessários para que o leitor possa inicialmente compreender códigos simples, aprimorar a habilidade de programação e então elaborar códigos avançados e capazes de manipular estruturas de dados mais complexas.

O livro é destinado a estudantes de graduação e pós-graduação, professores, pesquisadores e profissionais interessados em ingressar ou se aprofundar na construção de algoritmos e na aplicação de estruturas de dados. Os conceitos são organizados e apresentados de maneira direta e objetiva para facilitar sua assimilação. Dessa forma, buscou-se adequar o conteúdo do livro para atender não apenas às demandas dos leitores ligados à área de computação, mas também às necessidades de um público mais geral.

Espera-se, a partir da disseminação dos conceitos abordados neste material, contribuir para a ampliação do conhecimento sobre estruturas de dados, incentivar o desenvolvimento de algoritmos para exploração de novas aplicações e apoiar o fortalecimento da ciência e tecnologia no país.

Hélio Pedrini

Parte I

Fundamentos de programação

- ☐ Capítulo 1: Introdução
- ☐ Capítulo 2: Variáveis e constantes
- ☐ Capítulo 3: Entrada e saída
- ☐ Capítulo 4: Operadores
- ☐ Capítulo 5: Comandos condicionais
- ☐ Capítulo 6: Comandos de repetição
- ☐ Capítulo 7: Vetores e matrizes
- ☐ Capítulo 8: Cadeias de caracteres
- ☐ Capítulo 9: Funções
- ☐ Capítulo 10: Ponteiros
- ☐ Capítulo 11: Tipos enumerados e estruturados
- ☐ Capítulo 12: Arquivos

INTRODUÇÃO

Neste capítulo, os conceitos de algoritmos e programas são introduzidos, os quais são de fundamental importância para a ciência da computação. Os princípios de organização de computadores são apresentados e discutidos. O objetivo é descrever brevemente as unidades básicas de um computador para facilitar a compreensão de seu funcionamento. Para a implementação de um conjunto de instruções, codificado na forma de programas para execução por um computador, optou-se por empregar a linguagem de programação C, que provê mecanismos para a construção de códigos estruturados e que utiliza eficientemente os recursos computacionais para a resolução de problemas.

1.1 Algoritmos e programas

A *ciência da computação* pode ser definida como o estudo da teoria, do projeto e da implementação de processos algorítmicos e sistemas computacionais. Dois conceitos importantes em ciência da computação são algoritmos e programas.

Um *algoritmo* é um conjunto de ações ou instruções, estruturadas em uma ordem lógica e sem ambiguidades, para a resolução de um problema. Ele deve ser especificado de forma independente das características da máquina em que será executado, garantindo maior portabilidade e flexibilidade ao ser implementado com uma linguagem de programação em um computador.

Um *programa* é a implementação ou codificação de um algoritmo em uma linguagem específica. Ele está sujeito às limitações físicas da máquina em que será executado, por exemplo, a capacidade de memória, a velocidade do processador e dos periféricos, entre outras.

O desenvolvimento de uma solução computacional envolve um conjunto de ações que devem ser planejadas e executadas. As principais etapas nesse processo são listadas a seguir: (i) compreensão do problema a ser resolvido, (ii) identificação dos dados de entrada e de saída, (iii) especificação dos passos para transformar os dados de entrada em dados de saída, (iv) projeto dos algoritmos, (v) projeto das estruturas de dados, (vi) análise dos algoritmos, (vii) implementação dos algoritmos, (viii) execução dos programas, (ix) avaliação dos resultados e (x) elaboração de documentação.

Uma forma comum de descrição dos passos de um algoritmo é o *pseudocódigo*, cuja representação emprega uma linguagem simples e livre destinada à leitura humana, e não à execução de instruções pelas máquinas. Um pseudocódigo normalmente omite detalhes sintáticos específicos de uma linguagem de programação.

1.2 Organização básica de computadores

Computador é uma máquina capaz de executar sequências de instruções por meio de programação para gerar determinado resultado. Um computador normalmente é utilizado para executar tarefas extensas e complexas que, caso fossem realizadas manualmente, exigiriam um tempo muito maior.

O conhecimento dos componentes básicos presentes em um computador auxilia a compreensão de como os programas funcionam. Os principais elementos que compõem um computador podem ser organizados em quatro categorias: (i) canal de comunicação, (ii) unidade de processamento, (iii) unidades de armazenamento e (iv) dispositivos de entrada e saída. Esses elementos básicos são ilustrados na Figura 1.1.

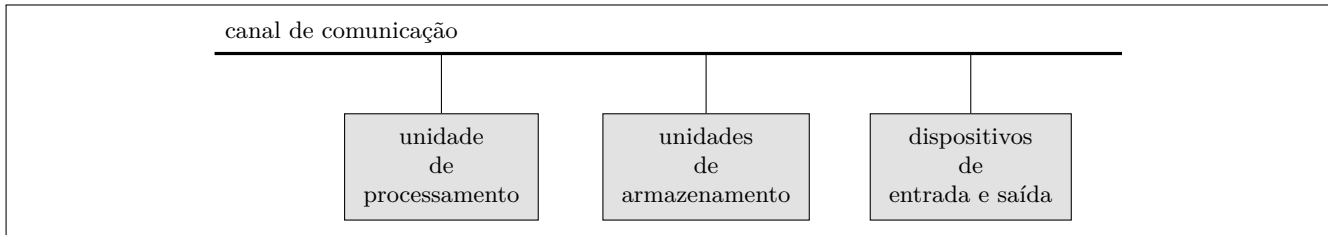


Figura 1.1: Componentes básicos de um computador.

A *unidade de processamento* é responsável pelas operações sobre os dados que trafegam no computador. Ela é composta de duas partes principais: a unidade lógica e aritmética e a unidade de controle. A *unidade lógica e aritmética* é responsável pelas operações lógicas, de deslocamento e aritmética sobre os dados. A *unidade de controle* é responsável por coordenar as operações da unidade de processamento.

As *unidades de armazenamento* são responsáveis por manter os dados manipulados pela unidade de processamento. As unidades de armazenamento são tipicamente divididas nas seguintes categorias: registradores, cache, memória principal e memória secundária. Os *registradores* são conjuntos de circuitos para manter dados temporariamente e permitir a comunicação rápida entre a unidade lógica e aritmética e a unidade de controle. A *memória cache* permite o armazenamento de dados que são acessados mais frequentemente ou mais recentemente, de modo a acelerar certas operações. A *memória principal* ou *memória primária* consiste em dispositivos para o armazenamento temporário de dados. Cada localização de armazenamento é identificada unicamente por um endereço. A *memória secundária* é responsável por armazenar dados de forma permanente e em grandes quantidades. A memória secundária tem custo mais baixo do que a memória principal, entretanto, o acesso aos dados é mais lento. Para que a unidade de processamento utilize dados armazenados na memória secundária, eles são primeiramente transferidos para a memória primária. A memória também pode ser categorizada como lógica e física. A *memória lógica* refere-se às porções de armazenamento que podem ser endereçadas e acessadas pelas instruções do processador, enquanto a *memória física* é implementada pelos circuitos integrados que formam a memória. Endereços lógicos são convertidos em endereços físicos durante a execução dos processos.

Os *dispositivos de entrada e saída* permitem a comunicação entre o computador e o mundo exterior, ou seja, usuários e outros equipamentos. Os dispositivos de entrada recebem dados e instruções, enquanto os dispositivos de saída retornam os dados processados. Alguns dispositivos de entrada comuns são teclado, *mouse* e microfone. Alguns dispositivos de saída comuns são monitor de vídeo, impressora e caixa de som.

O *canal de comunicação* ou *barramento* corresponde ao meio de transferência de dados que interliga as demais unidades do computador. As principais funções do barramento são a comunicação de dados entre as unidades, a comunicação de endereços para selecionar a origem ou o destino dos sinais transmitidos no barramento e a comunicação de controle para sincronizar as atividades do computador.

A menor unidade de informação que pode ser armazenada ou transmitida é denominada *bit*.¹ Dados são transferidos entre a memória principal e o processador em grupos de bits chamados de *palavras* ou *sequências de dígitos binários*. O número de bits em uma palavra, conhecido como *tamanho da palavra*, é uma característica importante da arquitetura de um computador e indica a unidade de transferência entre o processador e a memória principal. Uma palavra é formada tipicamente por 8, 16, 32 ou 64 bits. Se a palavra tiver 8 bits, ela é chamada de *byte*.

Um *endereço* é um identificador único para uma posição de memória do computador. O número total de endereços identificáveis na memória é chamado de *espaço de endereçamento*. Computadores que utilizam sistemas de numeração binária (descritos na Subseção D.1.2 do Apêndice D) expressam os endereços de memória como números binários. A Figura 1.2 ilustra um espaço de endereçamento com $n = 2^m$ posições de memória formadas por células de m bits.

Células são agrupadas em palavras de bits. O tamanho típico de uma célula é de 8 bits. Assim, uma palavra de 32 bits tem 4 células. Um computador que utiliza palavras de 32 bits, por exemplo, para representar endereços de memória pode endereçar um espaço de $2^{32} = 4.294.967.296$ bytes ou 4 gigabytes de memória.

¹O termo *bit* é uma contração da expressão em inglês *binary digit*, em que cada bit é normalmente representado pelo símbolo 0 ou 1. Essa representação é conveniente para manipular dados por meio de circuitos digitais capazes de diferenciar dois estados.

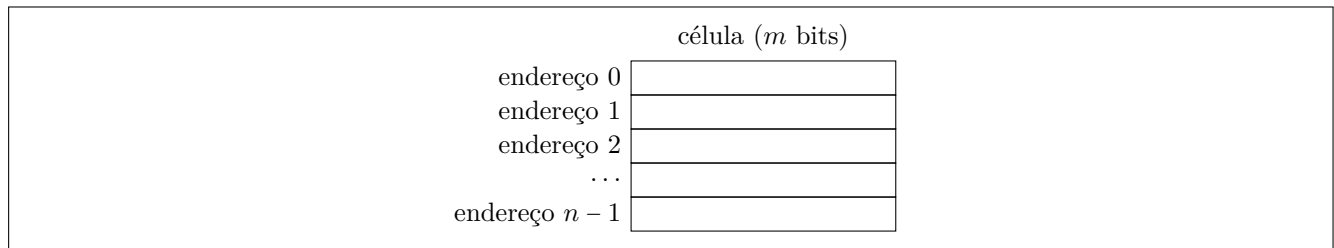


Figura 1.2: Endereços de memória.

A capacidade de armazenamento dos computadores tem aumentado significativamente com o avanço tecnológico. Algumas unidades utilizadas para se referir à capacidade de armazenamento de dados na memória são apresentadas na Tabela 1.1.

Tabela 1.1: Exemplos de unidades de memória.

Unidade	Símbolo	Número de Bytes
Byte	B	$2^0 = 1$
Kilobyte	KB	$2^{10} = 1.024$
Megabyte	MB	$2^{20} = 1.048.576$
Gigabyte	GB	$2^{30} = 1.073.741.824$
Terabyte	TB	$2^{40} = 1.099.511.627.776$
Petabyte	PB	$2^{50} = 1.125.899.906.842.624$
Exabyte	EB	$2^{60} = 1.152.921.504.606.846.976$

Sistema operacional é um conjunto de programas que gerenciam os recursos do computador, como a unidade de processamento, os dispositivos de armazenamento e os dispositivos de entrada e saída. Ele controla a comunicação entre os equipamentos e os programas, facilitando a interação entre o computador e o usuário. O sistema operacional provê mecanismos para compartilhar recursos com múltiplos usuários, de forma eficiente e segura, preservando a integridade dos recursos em decorrência de acessos indevidos e resolvendo eventuais conflitos entre processos concorrentes.

1.3 Linguagem de programação C

A escrita de um programa de computador requer a utilização de uma *linguagem de programação*, que consiste em um conjunto de símbolos predefinidos que são combinados de acordo com regras sintáticas estabelecidas. Ao longo dos anos, as linguagens de programação evoluíram em termos de nível de abstração requerido para escrever os códigos.

Em uma linguagem de programação de *baixo nível* de abstração, o programador deve conhecer as características da máquina em que o código será executado, o que não é necessário em uma linguagem de programação de *alto nível* de abstração, mais próxima à linguagem humana. Essa evolução deve-se ao propósito de aumentar a produtividade do programador, em que maior atenção pode ser concentrada ao problema ou à aplicação, e não aos detalhes da máquina. Além disso, as linguagens de alto nível são portáteis para uma variedade de computadores e sistemas operacionais diferentes.

A linguagem de programação C foi criada em 1972 por Dennis Ritchie, na empresa Bell Telephone Laboratories, que era ligada à companhia de telecomunicações American Telephone and Telegraph (AT&T). Trata-se de uma linguagem de alto nível, de grande portabilidade, estruturada e procedural. A linguagem também fornece instruções de baixo nível, permitindo ao programador acesso direto à memória e ao processador da máquina. A linguagem foi revisada e padronizada pelo American National Standards Institute (ANSI) em 1989 e, após algumas modificações, adotada em 1990 pela International Organization for Standardization (ISO).

Um código-fonte escrito em linguagem C deve inicialmente ser compilado, ou seja, convertido em instruções que possam ser executadas por um computador. A Figura 1.3 ilustra as principais etapas do processo de conversão de código-fonte em executável.

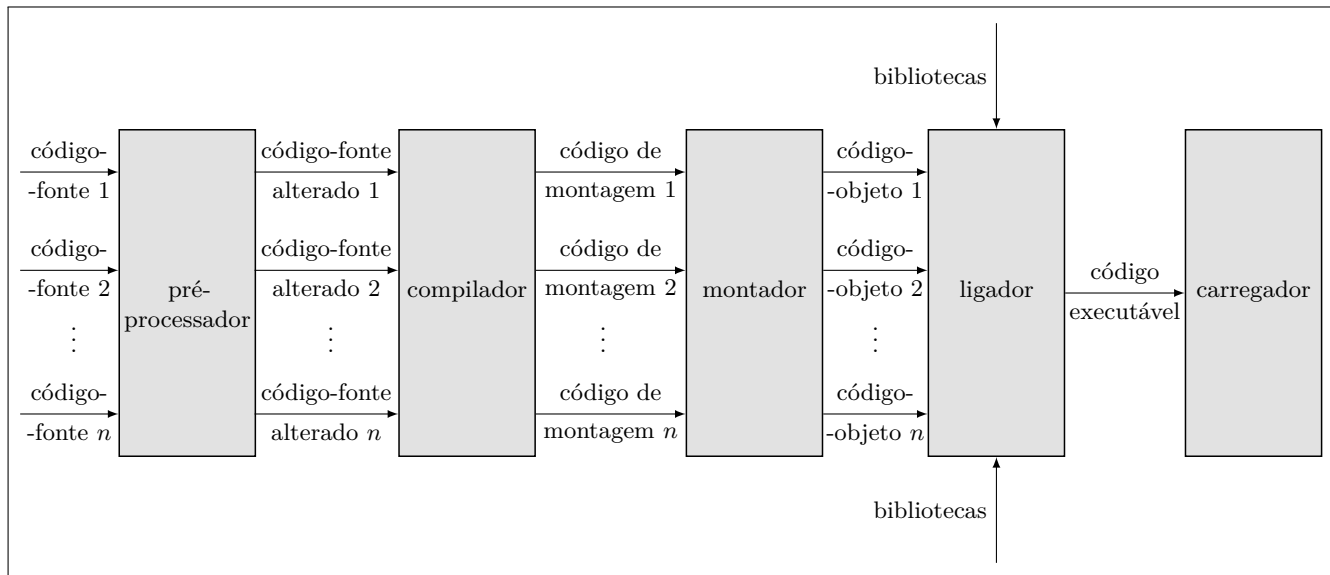


Figura 1.3: Processo de conversão de código-fonte em código executável.

O processo de tradução do código inicia-se com a *análise léxica*, que é responsável pela busca e identificação de determinados símbolos (por exemplo, palavras reservadas, identificadores, operadores, separadores) no código-fonte. Essa etapa é também responsável por algumas alterações no código, como a eliminação de espaços em branco, indentação e comentários do programa.

A *análise sintática* é o processo responsável por verificar se os símbolos contidos no código-fonte formam um programa válido, ou seja, se uma cadeia de símbolos léxicos pode ser gerada por uma gramática. Erros sintáticos são detectados em tempo de compilação, de modo que o programador deve corrigir os problemas para prosseguir com o processo de tradução do código-fonte em executável.

A *análise semântica* é responsável pela verificação da validade das estruturas construídas pelo analisador sintático. Um exemplo é a verificação de tipos de variáveis em expressões. Erros semânticos são manifestados em tempo de execução, produzindo resultado ou comportamento incorreto. O uso de um depurador pode auxiliar o programador a identificar erros sintáticos no código-fonte.

A geração de código é a etapa responsável pela conversão das sentenças válidas criadas pelo analisador semântico em um conjunto de instruções em linguagem de máquina para o computador em que o programa será executado. Dessa forma, o processo de geração de código é dependente da arquitetura da máquina.

O *pré-processador* analisa o código-fonte e efetua certas conversões léxicas baseadas em diretivas de compilação, como expansão de macros, compilação condicional e inclusão de arquivos de cabeçalho. As diretivas de compilação são recursos oferecidos pela linguagem para facilitar a escrita e a manutenção dos códigos. Todas as diretivas do pré-processador C são iniciadas com o símbolo '#'.

O *compilador* é responsável pela tradução das instruções presentes no código-fonte em sequências equivalentes de instruções em linguagem simbólica ou de montagem. Alguns compiladores, entretanto, podem não gerar esse código intermediário, convertendo as instruções do código-fonte diretamente em código executável. Erros detectados durante o processo de compilação são reportados ao programador. Após a correção dos erros, o código-fonte deve ser compilado novamente. O compilador pode aplicar um processo de otimização para transformar trechos do código em porções funcionalmente equivalentes, com a finalidade de melhorar certas características, como tempo de execução ou tamanho do código.

O *montador* converte o código de montagem em código-objeto pela tradução de cada instrução do programa para a sequência de bits que codifica a instrução a ser executada pela máquina. Referências simbólicas são resolvidas pelo montador em endereços reais de memória. Espaços em memória são reservados para o armazenamento de instruções e dados. O código gerado é dependente da arquitetura.

Quando um código-fonte é composto de vários módulos ou contém chamadas a funções de bibliotecas, a etapa de ligação dos códigos-objetos gerados ou das bibliotecas necessárias deve ser ativada para gerar o código executável