

## CAPITOLO I

---

# Nozioni sulla elaborazione elettronica

---

### Obiettivi

- Comprendere le nozioni elementari relative ai computer.
- Familiarizzare con i diversi tipi di linguaggi di programmazione.
- Conoscere la storia del linguaggio di programmazione C.
- Venire a conoscenza della libreria standard del C.
- Comprendere l'ambiente di sviluppo dei programmi C.
- Apprezzare le ragioni per le quali è appropriato apprendere il C in un primo corso di programmazione.
- Apprezzare le ragioni per le quali il C fornisce le basi fondamentali per i successivi studi sulla programmazione, in generale, e sul C++ in particolare.

### 1.1 Introduzione

Benvenuti nel mondo del C! Abbiamo lavorato duramente per creare quella che, speriamo sinceramente, sarà per voi una esperienza informativa e divertente. Il C è un linguaggio difficile che normalmente è insegnato soltanto ai programmatori esperti, perciò questo libro è unico tra quelli sul C:

- È adatto a chi ha una conoscenza tecnica, ma ha poca o nessuna esperienza di programmazione.
- È adatto ai programmatori esperti che desiderano una trattazione rigorosa e approfondita del linguaggio.

Come può un libro piacere a entrambi i gruppi? La risposta è che il libro enfatizza il raggiungimento della *chiarezza* del programma, attraverso le provate tecniche di “programmazione strutturata”. Quelli che non sono ancora dei programmatori impareranno sin dall’inizio il modo “giusto”. Abbiamo tentato di scrivere in un modo chiaro e diretto. Il libro è abbondantemente illustrato. Cosa probabilmente anche più importante, il libro presenta un enorme numero di programmi C funzionanti e mostra gli output (risultati) prodotti, quando questi programmi sono eseguiti su un computer.

I primi quattro capitoli introducono i principi della elaborazione elettronica, la programmazione dei computer e il linguaggio C. Le discussioni sono corredate da una introduzione alla programmazione dei computer, usando l’approccio strutturato. I principianti

che hanno seguito i nostri corsi ci hanno assicurato che il materiale di questi capitoli fornisce una solida base per l'approfondimento del C sviluppato nei Capitoli dal 5 al 14. Di solito, i programmatori esperti leggeranno velocemente i primi quattro capitoli e scopriranno in seguito che il C è trattato nei Capitoli dal 5 al 14 in modo rigoroso e, allo stesso tempo, interessante. Essi apprezzeranno in modo particolare i capitoli avanzati, con la loro trattazione dettagliata dei puntatori, delle stringhe, dei file e delle strutture di dati.

Molti programmatori esperti ci hanno riferito il loro apprezzamento per la nostra discussione sulla programmazione strutturata. Per quanto avessero programmato spesso in un linguaggio strutturato come il Pascal, non erano in grado di scrivere il miglior codice possibile perché non avevano mai conosciuto in modo formale la programmazione strutturata. Ora, dopo avere imparato il C da questo libro, hanno la possibilità di migliorare il proprio stile di programmazione. Per cui, che siate dei principianti o dei programmatori esperti, in questo libro ci sono molte cose che potranno informarvi, divertirvi e stimolarvi.

Molti hanno familiarità con le cose interessanti che i computer sono in grado di fare. In questo corso, imparerete a comandare i computer perché facciano quelle cose. È il *software* (ovverosia, le istruzioni che voi scrivete per ordinare al computer di eseguire delle azioni e prendere delle decisioni) che controlla i computer (detti spesso *hardware*), e il C è oggi uno dei linguaggi di programmazione più popolari. Questo testo fornisce una introduzione alla programmazione dell'ANSI C, la versione standardizzata nel 1989 negli Stati Uniti dall'Istituto Nazionale Americano per gli Standard (ANSI) e, nel resto del mondo, dall'Organizzazione Internazionale per gli Standard (ISO) .

L'utilizzo dei computer sta crescendo in quasi tutti i campi delle attività umane. In un'era di costi costantemente in crescita, quelli della elaborazione elettronica sono diminuiti vertiginosamente, a causa degli impressionanti sviluppi della tecnologia hardware e software. Quei computer che, solo 25 anni fa, potevano riempire grandi stanze e costavano milioni di dollari, oggi possono essere prodotti su lamine di silicio (chip) più piccole di un'unghia, che costano forse pochi dollari. Ironicamente, il silicio è uno dei materiali più abbondanti sulla terra: è un ingrediente della sabbia comune. La tecnologia del chip di silicio ha reso l'elaborazione elettronica così economica che, in tutto il mondo, sono in uso centinaia di milioni di computer general-purpose (per scopi generici), aiutando la gente negli affari, nell'industria, nel governo e nelle loro vite personali, che potrebbero facilmente raddoppiarsi in una manciata di anni.

Può il C essere insegnato in un primo corso di programmazione, ovverosia è adatto al pubblico a cui si rivolge questo libro? Noi crediamo di sì. Qualche anno fa abbiamo accettato questa sfida, quando il linguaggio affermato nei primi corsi di informatica era il Pascal. Abbiamo scritto *C How To Program*, la prima edizione di questo testo, e centinaia di università in tutto il mondo lo hanno utilizzato. I corsi basati su quel libro hanno dimostrato di essere efficaci tanto quanto i loro predecessori basati sul Pascal. Non sono state osservate differenze significative, eccetto che gli studenti sono meglio motivati, perché sanno che, nei loro corsi di livello superiore e nelle proprie carriere, useranno con maggior probabilità il C che non il Pascal. Gli studenti che apprendono il C sanno anche che saranno meglio preparati ad apprendere velocemente il C++. Il C++ è una sovrastruttura del C, rivolta ai programmatori che vogliono scrivere programmi orientati agli oggetti. Diremo qualcosa in più sul C++ nella Sezione 1.14.

Un fenomeno interessante, che si sta verificando nel mercato dei linguaggi di programmazione, è che molti dei fornitori principali, piuttosto che offrire dei prodotti separati, ora vendono semplicemente un C/C++ combinato. Ciò dà all'utente, se lo desidera, la possibilità di continuare a programmare in C e di migrare gradualmente al C++, quando lo riterrà opportuno.

State dunque per intraprendere un percorso interessante e, speriamo, remunerativo. Man mano che procederete, se vorrete comunicare con noi, inviateci una email su Internet all'indirizzo **deitel@deitel.com**. Faremo tutto il possibile per rispondervi velocemente. Buona fortuna!

## 1.2 Che cosa è un computer?

Un *computer* è un dispositivo in grado di eseguire dei calcoli e di prendere delle decisioni logiche, il tutto a una velocità superiore di milioni e anche miliardi di volte a quella degli esseri umani. Oggi per esempio, molti personal computer possono eseguire decine di milioni di addizioni per secondo. Una persona che operi a una calcolatrice da tavolo potrebbe aver bisogno di decenni, per completare lo stesso numero di calcoli che un personal computer può eseguire in un secondo. (Punto di riflessione: come potreste sapere se la persona ha sommato correttamente i numeri? Come potreste sapere se il computer ha sommato correttamente i numeri?). I *supercomputer* più veloci oggi possono eseguire centinaia di miliardi di addizioni per secondo: all'incirca quanto i calcoli che centinaia di migliaia di persone potrebbero eseguire in un anno! E nei laboratori di ricerca stanno già funzionando computer in grado di eseguire migliaia di miliardi di istruzioni per secondo.

I computer elaborano i *dati* sotto il controllo di insiemi di istruzioni chiamati *programmi per computer*. Tali programmi guidano il computer per mezzo di insiemi ordinati di azioni specificati da persone chiamate *programmatori di computer*.

I vari dispositivi che compongono un sistema di computer (come la tastiera, lo schermo, i dischi, la memoria, e l'unità di elaborazione) sono chiamati complessivamente *hardware*. I programmi che possono essere eseguiti su un computer sono chiamati *software*. Il costo dell'hardware è diminuito vertiginosamente negli ultimi anni, al punto che il personal computer è diventato un oggetto di uso comune. Sfortunatamente, i costi di sviluppo del software sono andati aumentando costantemente man mano che i programmatori hanno sviluppato applicazioni sempre più potenti e complesse, senza peraltro essere capaci di ottenere un miglioramento corrispondente nella tecnologia di sviluppo del software. In questo libro, imparerete metodi di sviluppo del software che ne riducono sostanzialmente i costi e accelerano il processo di sviluppo di applicazioni software potenti e di alta qualità. Questi metodi includono la *programmazione strutturata*, la *programmazione top-down per raffinamenti successivi*, la *programmazione modulare*.

## 1.3 L'organizzazione del computer

Indipendentemente dalle differenti apparenze fisiche, ogni computer può essere virtualmente concepito come suddiviso in sei *unità logiche* o sezioni. Queste sono:

1. *Unità di input (ingresso di dati)*. Questa è la sezione "ricevente" del computer. Essa ottiene informazioni (dati e programmi per il computer) da vari *dispositivi di input* e mette queste informazioni a disposizione delle altre unità, in modo che possano essere elaborate. La

maggior parte delle informazioni oggi giorno è immessa nei computer attraverso una tastiera simile a quella di una macchina per scrivere.

2. *Unità di output (uscita di dati)*. Questa è la sezione di “spedizione” del computer. Essa prende l’informazione che è stata elaborata dal computer e la invia a diversi *dispositivi di output*, in modo da renderla disponibile per l’utilizzo all’esterno del computer. Oggi giorno la maggior parte delle informazioni è inviata all’esterno dei computer attraverso la visualizzazione sullo schermo o la stampa su carta.
3. *Unità di memoria*. Questa è la sezione di “magazzino” con accesso rapido e capacità relativamente bassa del computer. Essa conserva le informazioni che sono state immesse attraverso l’unità di input, in modo che possano essere rese immediatamente disponibili quando saranno necessarie. L’unità di memoria conserva anche le informazioni che sono già state elaborate, fintanto che possano ancora essere inviate ai dispositivi di output dalla corrispondente unità. L’unità di memoria è spesso chiamata anche *memoria o memoria primaria*.
4. *Unità aritmetica e logica (ALU)*. Questa è la sezione di “produzione” del computer. È responsabile dell’esecuzione dei calcoli come le somme, le sottrazioni, le moltiplicazioni e le divisioni. Essa contiene i meccanismi di decisione che consentono al computer, per esempio, di confrontare due elementi prelevati dalla unità di memoria, per determinare se sono uguali oppure no.
5. *Unità di elaborazione centrale (CPU)*. Questa è la sezione “amministrativa” del computer. È il coordinatore del computer ed è responsabile della supervisione per le operazioni delle altre sezioni. La CPU indica alla unità di input il momento in cui le informazioni dovranno essere lette nella unità di memoria, indica alla ALU, il momento in cui le informazioni della memoria dovranno essere utilizzate per i calcoli, e indica alla unità di output il momento in cui dovrà inviare le informazioni dalla unità di memoria a certi dispositivi di output.
6. *Unità di memoria secondaria*. Questa è la sezione di “magazzino” a lungo termine e con alta capacità del computer. I programmi e i dati, che in un certo momento non sono utilizzati dalle altre unità, saranno normalmente sistemati su un dispositivo di memoria secondaria (come i dischi), finché non saranno nuovamente necessari; probabilmente ore, giorni, mesi o anche anni più tardi.

## 1.4 L’elaborazione batch, la multiprogrammazione e il timesharing

I primi computer erano in grado di eseguire soltanto un *job (processo)* o un *task (lavoro)* alla volta. Questo modo di operare dei computer è spesso chiamato *elaborazione batch (elaborazione a lotti)* monoutente. Il computer esegue un solo programma per volta, mentre i dati sono elaborati in gruppi ossia batch. In questi primi sistemi, generalmente, gli utenti sottoponevano i loro processi al centro di elaborazione in mazzi di schede perforate. Gli utenti dovevano spesso aspettare ore, o anche giorni, prima che i tabulati fossero restituiti alle loro scrivanie.

Man mano che i computer sono diventati più potenti, si è reso evidente che l’elaborazione batch monoutente utilizzava raramente in modo efficiente le risorse del computer. Si pensò quindi di fare in modo che molti processi o lavori potessero *condividere (share)* le risorse del computer per ottenerne un miglior utilizzo.. Questo modo di operare è detto

*multiprogrammazione*. La multiprogrammazione richiede l'elaborazione "simultanea" di molti processi sul computer: questo, in altre parole, condivide le proprie risorse tra i vari processi che competono per la sua attenzione. Con i primi sistemi in multiprogrammazione però, gli utenti dovevano ancora sottoporre i processi su mazzi di schede perforate e attendere ore o giorni per i risultati.

Negli anni '60, molti gruppi industriali e le università esplorarono il concetto di *timesharing* (*condivisione del tempo*). Il timesharing è un caso speciale di multiprogrammazione per mezzo del quale gli utenti accedono al computer attraverso dispositivi di input/output o *terminali*. In un tipico sistema di computer in timesharing, potrebbero esserci dozzine o anche centinaia di utenti che condividono contemporaneamente il computer. In realtà, il computer non serve tutti gli utenti in modo simultaneo. Esso esegue piuttosto una piccola porzione del processo di un utente e in seguito presta il proprio servizio all'utente successivo. Il computer fa tutto ciò così velocemente che, in un secondo, è in grado di fornire molte volte il proprio servizio a ogni utente. In questo modo gli utenti *sembrano* essere serviti simultaneamente.

## 1.5 L'elaborazione personale, distribuita e client/server

Nel 1977, la Apple Computer rese popolare il fenomeno della *elaborazione personale*. Al principio, questa era il sogno di ogni hobbista. I computer divennero abbastanza economici perché la gente potesse comprarli per il proprio uso personale o per i propri affari. Nel 1981, la IBM, il maggior fornitore di computer nel mondo, introdusse il Personal Computer IBM. L'elaborazione personale, letteralmente nel giro di una notte, diventò legittima negli affari, nell'industria e nelle organizzazioni governative.

Questi computer erano però delle unità "indipendenti": la gente eseguiva il proprio lavoro sulla propria macchina e quindi trasportava avanti e dietro dei dischi, per condividere le informazioni. Per quanto i primi personal computer non fossero abbastanza potenti da gestire molti utenti in timesharing, queste macchine potevano però essere collegate insieme in reti di computer, a volte su linee telefoniche e a volte in reti locali all'interno di una organizzazione. Ciò ha portato al fenomeno della *elaborazione distribuita*, in cui il carico di elaborazione di una organizzazione, invece di essere eseguito necessariamente in qualche installazione centrale di computer, è distribuito attraverso le reti alle varie postazioni in cui viene svolto effettivamente il lavoro. I personal computer erano sufficientemente potenti da gestire le richieste di elaborazione di utenti individuali e le fondamentali attività della comunicazione: passare avanti e indietro le informazioni in forma elettronica.

Oggi giorno, i personal computer più potenti lo sono tanto quanto le macchine da un milione di dollari di solo dieci anni fa. Le macchine desktop (da tavolo) più potenti (dette *workstation*) forniscono ai singoli utenti delle capacità enormi. L'informazione è facilmente condivisa attraverso le reti di computer, nelle quali alcune macchine, dette *file server* (*fornitori di servizi per i file*), offrono un magazzino comune di programmi e dati che può essere utilizzato da macchine *client* (*clienti*) distribuite in tutta la rete; da cui il termine *elaborazione client/server*. Il C e il C++ sono diventati i linguaggi di programmazione scelti per scrivere il software dei sistemi operativi, delle reti di computer e delle applicazioni distribuite in ambienti client/server.

## 1.6 I linguaggi macchina, assembly e di alto livello.

I programmatori scrivono le istruzioni in vari linguaggi di programmazione, alcuni direttamente comprensibili dal computer e altri che richiedono dei passi intermedi di *traduzione*. Oggigiorno si utilizzano centinaia di linguaggi per i computer. Questi possono essere suddivisi in tre categorie generali:

1. Linguaggi macchina
2. Linguaggi assembly
3. Linguaggi di alto livello

Ogni computer può comprendere direttamente soltanto il proprio *linguaggio macchina*. Il linguaggio macchina è la “lingua naturale” di un particolare computer. Esso è strettamente correlato con la progettazione dell’hardware del computer. I linguaggi macchina consistono generalmente di sequenze di numeri (riducibili in definitiva a successioni di 1 e 0) che ordinano ai computer di eseguire, una per volta, le loro operazioni più elementari. I linguaggi macchina *dipendono dalla macchina*: in altre parole, un particolare linguaggio macchina può essere utilizzato soltanto su un tipo di computer. I linguaggi macchina sono scomodi per gli esseri umani, come si può vedere dal seguente frammento di programma in linguaggio macchina, che aggiunge la paga degli straordinari a quella base e immagazzina il risultato nella paga lorda.

```
+1300042774
+1400593419
+1200274027
```

Man mano che i computer sono diventati più popolari, è divenuto evidente che la programmazione in linguaggio macchina era semplicemente troppo lenta e noiosa per la maggior parte dei programmatori. Invece di usare le sequenze di numeri che i computer potevano capire direttamente, i programmatori cominciarono a usare delle abbreviazioni simili all’inglese, per rappresentare le operazioni elementari del computer. Tali abbreviazioni formarono le basi per i *linguaggi assembly*. Furono sviluppati dei *programmi traduttori*, chiamati *assembler (assemblatori)*, per convertire in linguaggio macchina, alla velocità del computer, i programmi scritti in linguaggio assembly. Anche il seguente frammento di programma in linguaggio assembly aggiunge la paga degli straordinari a quella base e immagazzina il risultato nella paga lorda, ma in modo più chiaro del suo equivalente in linguaggio macchina:

```
LOAD    BASEPAY
ADD     OVERPAY
STORE  GROSSPAY
```

L’utilizzo dei computer aumentò rapidamente con l’avvento dei linguaggi assembly, ma questi richiedevano ancora molte istruzioni per eseguire anche il più semplice dei compiti. Per accelerare il processo di programmazione, furono sviluppati dei *linguaggi di alto livello* in cui si poteva scrivere una singola istruzione per eseguire un compito essenziale. I programmi traduttori, che convertono in linguaggio macchina il codice scritto in quello di alto livello, sono chiamati *compilatori*. I linguaggi di alto livello consentono ai programmatori di scrivere delle istruzioni, che sembrano quasi simili all’inglese di ogni giorno e contengono le notazioni matematiche utilizzate comunemente. Un programma per la busta paga scritto in un linguaggio di alto livello potrebbe contenere una istruzione come:

```
grossPay = basePay + overTimePay
```

Ovviamente dal punto di vista dei programmatori, i linguaggi di alto livello sono molto più desiderabili, in confronto ai linguaggi macchina o a quelli assembly. Il C e il C++ sono tra i più potenti e più diffusamente utilizzati linguaggi di alto livello.

## 1.7 La storia del C

Il C si è evoluto da due precedenti linguaggi: il BCPL e il B. Il BCPL fu sviluppato da Martin Richards, nel 1967, come un linguaggio per scrivere il software dei sistemi operativi e dei compilatori. Ken Thompson prese a modello il BCPL, per molte caratteristiche del suo linguaggio B, e usò B per creare le prime versioni del sistema operativo UNIX nei Bell Laboratories, nel 1970, su un computer DEC PDP-7. Il BCPL e il B erano linguaggi “non tipizzati”: ogni unità di informazione occupava una “word” (parola) nella memoria e, per esempio, l’onere di considerare un dato come un numero intero o come uno reale ricadeva sulle spalle del programmatore.

Il linguaggio C fu una evoluzione del B sviluppata da Dennis Ritchie, nei Bell Laboratories, e fu implementato originariamente su un computer DEC PDP-11, nel 1972. Il C inizialmente divenne famoso come il linguaggio in cui era stato sviluppato il sistema operativo UNIX. Oggi, sono scritti in C e/o in C++ quasi tutti i principali sistemi operativi dell’ultima generazione. Negli ultimi vent’anni, il C è diventato disponibile sulla maggior parte dei computer. Il C è indipendente dall’hardware. Con una progettazione attenta, è possibile scrivere in C dei programmi che siano *portabili* sulla maggior parte dei computer. Il C utilizza molti importanti concetti del BCPL e del B, mentre aggiunge la tipizzazione dei dati e altre potenti caratteristiche.

Dagli ultimi anni ’70, il C si è evoluto in quello che oggi è chiamato “C tradizionale”. La pubblicazione, nel 1978, del libro di Kernighan e Ritchie, *Il linguaggio di programmazione C*, ha attirato molte attenzioni su questo linguaggio. Questa pubblicazione è diventata, in campo informatico, uno dei libri di maggior successo di tutti i tempi.

La rapida espansione del C sui diversi tipi di computer (detti, a volte, *piattaforme hardware*) ha prodotto molte varianti. Queste erano simili, ma spesso incompatibili. Ciò rappresentava un problema serio per i programmatori, che avevano bisogno di sviluppare un codice che potesse girare su piattaforme diverse. Divenne allora evidente che era necessaria una versione standard del C. Nel 1983, per “fornire una definizione del linguaggio che non fosse ambigua e che fosse indipendente dalle macchine”, si creò il comitato tecnico X3J11, alle dipendenze del Comitato Nazionale Americano per gli Standard dei Computer e l’Elaborazione della Informazione (X3). Lo standard fu approvato nel 1989. Il documento si chiama ANSI/ISO 9899: 1990. Copie di questo documento possono essere richieste all’Istituto Nazionale Americano per gli Standard, il cui indirizzo è riportato nella Prefazione di questo testo. La seconda edizione del libro di Kernighan e Ritchie, pubblicata nel 1988, riflette questa versione, chiamata ANSI C, usata ancora oggi in tutto il mondo (Ke88).



### *Obiettivo portabilità 1.1*

*Dato che il C è un linguaggio indipendente dagli hardware ed è largamente diffuso, le applicazioni scritte in C possono essere eseguite, con poche o nessuna modifica, su una vasta gamma di sistemi di computer differenti.*

## 1.8 La libreria standard del C

I programmi scritti in C, come apprenderete nel Capitolo 5, consistono di moduli o pezzi chiamati *funzioni*. Voi potrete programmare tutte le funzioni di cui avrete bisogno per scrivere un programma C, ma la maggior parte dei programmatori C traggono vantaggio da una ricca collezione di funzioni già esistenti, chiamata *libreria standard del C*. Di conseguenza, ci sono in realtà due aspetti del “mondo” del C da imparare. Il primo è imparare il linguaggio C in sé, mentre il secondo è imparare a utilizzare le funzioni della libreria standard del C. Nel corso di questo libro, parleremo di molte di quelle funzioni. L'Appendice B (condensata e adattata a partire dallo stesso documento dello standard ANSI C) elenca tutte le funzioni disponibili nella libreria standard del C. Il libro di Plauger (P192) dovrebbe essere letto dai programmatori che abbiano bisogno di una profonda conoscenza delle funzioni incluse nella libreria, di come implementarle e di come usarle per scrivere un codice portabile.

In questo corso, sarete incoraggiati a utilizzare un *approccio di costruzione a blocchi* per creare i programmi. Evitate di inventare nuovamente la ruota e utilizzate i pezzi già esistenti: questa si chiama *riusabilità del software*. Quando programmerete in C, userete tipicamente i seguenti blocchi:

- Le funzioni della libreria standard
- Le funzioni create da voi stessi
- Le funzioni che sono state sviluppate da altre persone e che vi sono state messe a disposizione

Il vantaggio di creare le vostre funzioni è che saprete esattamente in che modo funzionino. Sarete in grado di esaminare il codice C. Lo svantaggio è dato dal dispendio di tempo richiesto dallo sforzo per progettare e sviluppare delle nuove funzioni.

Usare le funzioni già esistenti vi eviterà di inventare nuovamente la ruota. Nel caso delle funzioni dello standard ANSI, saprete che sono state scritte con molta cura e che i vostri programmi avranno un'alta probabilità di essere portabile, poiché state utilizzando delle funzioni che sono virtualmente disponibili su tutte le implementazioni ANSI C.



### *Obiettivo efficienza 1.1*

---

*Utilizzare le funzioni della libreria standard ANSI, invece di scrivere le vostre corrispondenti versioni, potrà migliorare l'efficienza del programma, perché quelle funzioni sono state scritte con cura per una esecuzione efficiente.*



### *Obiettivo portabilità 1.2*

---

*Utilizzare le funzioni della libreria standard ANSI, invece di scrivere le vostre corrispondenti versioni, potrà migliorare la portabilità del programma, perché quelle funzioni sono state incluse in quasi tutte le versioni del C.*

## 1.9 Gli altri linguaggi di alto livello

Sono stati sviluppati centinaia di linguaggi di alto livello, ma soltanto alcuni hanno ottenuto un largo consenso. Il *FORTRAN* (FORmula TRANslator, Traduttore di formula) fu sviluppato dalla IBM, tra il 1954 e il 1957, per essere utilizzato nelle applicazioni scientifiche e

di ingegneria che richiedono complessi calcoli matematici. Il FORTRAN è utilizzato diffusamente ancora oggi.

Il *COBOL* (COmmon Business Oriented Language, Linguaggio orientato alle comuni attività commerciali) fu sviluppato nel 1959 da un gruppo di produttori di computer e di utenti governativi e industriali. Il COBOL è utilizzato principalmente per le applicazioni commerciali che richiedono una precisa ed efficiente manipolazione di grandi quantità di dati. Ancora oggi, più della metà di tutto il software commerciale è programmato in COBOL. Più di un milione di persone sono impiegate come programmatori COBOL.

Il *Pascal* fu progettato circa nello stesso periodo del C. Esso era destinato a un uso accademico. Diremo qualcosa di più a proposito del Pascal nella prossima sezione.

## 1.10 La programmazione strutturata

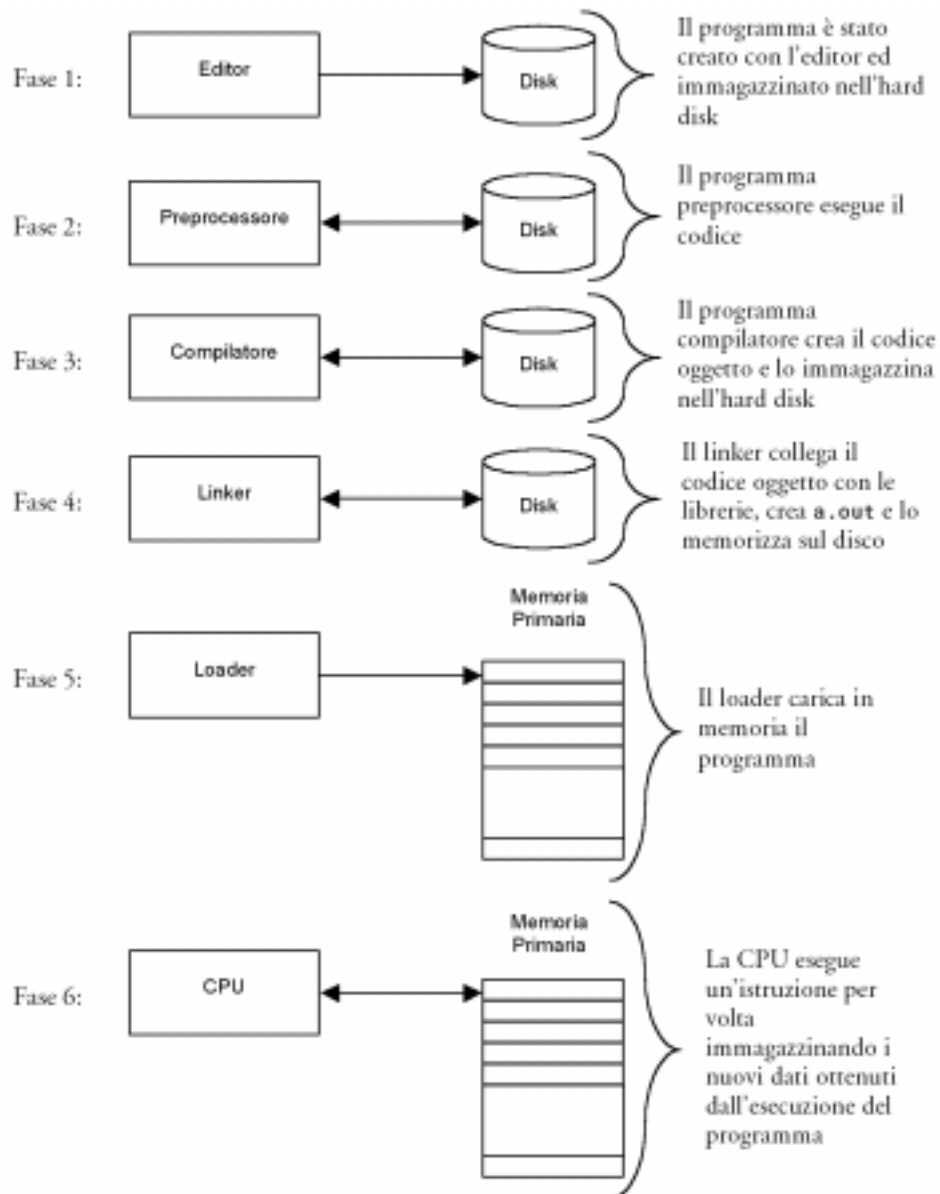
Durante gli anni '60, molti grandi progetti di sviluppo di software incontrarono delle serie difficoltà. Si era tipicamente in ritardo rispetto ai piani di sviluppo del software, i costi eccedevano notevolmente i budget e i prodotti finiti non erano affidabili. La gente cominciò a comprendere che lo sviluppo del software era una attività più complessa di quanto avesse immaginato. L'attività di ricerca degli anni '60 ebbe come risultato lo sviluppo della *programmazione strutturata*: un approccio disciplinato alla scrittura di programmi chiari, dalla correttezza dimostrabile e semplici da modificare. Il Capitolo 3 e il Capitolo 4 presenteranno una panoramica generale sui principi della programmazione strutturata. Il resto del testo affronterà lo sviluppo di programmi strutturati in C.

Uno dei risultati più tangibili prodotti da questa ricerca fu lo sviluppo, da parte del Professor Nicklaus Wirth nel 1971, del linguaggio di programmazione Pascal. Il Pascal, che prese il suo nome dal matematico e filosofo del diciassettesimo secolo Blaise Pascal, fu progettato per insegnare la programmazione strutturata negli ambienti accademici e divenne rapidamente il linguaggio di introduzione alla programmazione preferito nella maggior parte delle università. Sfortunatamente, il linguaggio mancava di molte delle strutture necessarie per renderlo utile nelle applicazioni commerciali, industriali e governative, perciò non fu accettato diffusamente in questi ambienti. La storia può ben ricordare che la sua selezione come base del linguaggio di programmazione *Ada* è stata la reale importanza del Pascal.

L'Ada fu sviluppato sotto il patrocinio del Dipartimento della Difesa degli Stati Uniti (DOD), durante gli anni '70 e all'inizio degli '80. Per produrre gli imponenti sistemi software di comando e di controllo del DOD, erano stati utilizzati centinaia di linguaggi diversi. Il DOD voleva un unico linguaggio che potesse andare incontro alle proprie necessità. Il Pascal fu scelto come base ma Ada, il linguaggio finale, risultò completamente diverso dal Pascal. Il linguaggio prese il suo nome da Lady Ada Lovelace, figlia del poeta Lord Byron. A Lady Lovelace è generalmente attribuito il merito di avere scritto, all'inizio del 1800, il primo programma per computer del mondo. Una importante caratteristica dell'Ada è il *multitasking*; questo consente ai programmatori di specificare che molte attività dovranno verificarsi in parallelo. Gli altri linguaggi di alto livello utilizzati diffusamente, di cui abbiamo discusso (inclusi il C e il C++), consentono di scrivere programmi che eseguono soltanto una attività per volta. Resta da vedere se l'Ada abbia raggiunto il suo obiettivo: ovverosia, produrre un software affidabile e ridurre sostanzialmente i costi di sviluppo e manutenzione del software.

## 1.11 Le basi dell'ambiente C

Tutti i sistemi C consistono generalmente di tre parti: l'ambiente, il linguaggio e la libreria standard del C. La discussione seguente spiegherà il tipico ambiente di sviluppo C, mostrato nella Figura 1.1.



**Figura 1.1** Un tipico ambiente C.

I programmi scritti in C passano tipicamente attraverso 6 fasi, prima di essere eseguiti (Figura 1.1). Queste sono: *editare*, *prelaborare*, *compilare*, *linkare* (*collegare*), *caricare* ed *eseguire*. In questo contesto ci concentreremo su un tipico sistema C basato su UNIX. Qualora non siate utilizzando un sistema UNIX, fate riferimento ai manuali del vostro sistema, o chiedete al vostro insegnante di eseguire queste attività nel vostro ambiente.

La prima fase consiste nella scrittura del codice (editing) in un file: questa si esegue con un *programma chiamato editor*. Il programmatore scrive un programma in C con l'editor e, se necessario, eseguirà delle correzioni. Il programma sarà quindi immagazzinato in un dispositivo di memoria secondaria, come un disco. Il nome del file del programma C dovrà terminare con l'estensione **.c**. Il **vi** e l'**emacs** sono due editor largamente utilizzati sui sistemi UNIX. I pacchetti software del C/C++, come il Borland C++ per i PC IBM e compatibili e il Symantec C++ per l'Apple Macintosh, dispongono di editor incorporati che sono agevolmente integrati nell'ambiente di programmazione. Supponiamo che il lettore sappia come editare un programma.

In seguito, il programmatore immetterà il comando di *compilazione* del programma. Il compilatore tradurrà il programma C nel codice in linguaggio macchina (detto anche *codice oggetto*). In un sistema C, prima che incominci la fase di traduzione, sarà eseguito automaticamente il programma *preprocessore*. Il preprocessore del C obbedisce a comandi speciali, chiamati *direttive del preprocessore*, con le quali si indica che sul programma dovranno essere eseguite determinate manipolazioni, prima della compilazione vera e propria. Tali manipolazioni consistono generalmente nella inclusione di altri file in quello da compilare, e nella sostituzione di simboli speciali con un testo del programma. Nei primi capitoli, saranno discusse le direttive più comuni del preprocessore; nel Capitolo 13 presenteremo una trattazione dettagliata di tutte le caratteristiche del preprocessore. Questo sarà invocato automaticamente dal compilatore, prima che il programma sia convertito in linguaggio macchina.

La quarta fase è chiamata *linking* (*collegamento*). I programmi scritti in C contengono tipicamente dei riferimenti a funzioni definite altrove, per esempio nelle librerie standard o in quelle di un gruppo di programmatori che lavorano su un particolare progetto. Di conseguenza, il codice oggetto prodotto dal compilatore conterrà tipicamente dei "buchi" dovuti a queste parti mancanti. Il *linker* collega il codice oggetto con quello delle funzioni mancanti per produrre una *immagine eseguibile* (senza pezzi mancanti). In un tipico sistema basato su UNIX, il comando per compilare e collegare i pezzi di un programma è **cc**. Per esempio, per compilare e collegare i pezzi di un programma chiamato **welcome.c** scriverete

```
cc welcome.c
```

al prompt di UNIX e premerete il tasto invio. Nel caso il programma sia stato compilato e collegato correttamente, sarà prodotto un file chiamato **a.out**. Questo sarà l'immagine eseguibile del nostro programma **welcome.c**.

La quinta fase è chiamata *caricamento*. Prima che possa essere eseguito, un programma dovrà essere caricato nella memoria. Questa operazione sarà eseguita dal *loader* (*caricatore*) che prenderà l'immagine eseguibile dal disco e la trasferirà nella memoria.

Finalmente, sotto il controllo della sua CPU, il computer eseguirà il programma, una istruzione per volta. Per caricare ed eseguire il programma in un sistema UNIX, scriveremo **a.out** al prompt di UNIX e premeremo il tasto invio.

La maggior parte dei programmi scritti in C ricevono e/o producono dei dati. Certe funzioni scritte in C prendono il proprio input (i dati in ingresso) dallo **stdin** (il *dispositivo standard per l'input*), che è assegnato normalmente alla tastiera, ma lo **stdin** potrebbe anche essere connesso a un altro dispositivo. L'output (i dati in uscita) è inviato allo **stdout** (il *dispositivo standard per l'output*), che è normalmente lo schermo del computer, ma che potrebbe anche essere connesso a un altro dispositivo. Quando affermiamo che un programma stampa un risultato, normalmente, intendiamo affermare che quello è visualizzato sullo schermo. I dati possono essere inviati ad altri dispositivi, come i dischi e le stampanti (stampa su carta). C'è anche un *dispositivo standard per l'errore* chiamato **stderr**. Il dispositivo dello **stderr** (normalmente connesso allo schermo) è utilizzato per visualizzare i messaggi di errore. È uso comune dirigere i dati dell'output regolare, ovvero lo **stdout**, verso un dispositivo diverso dallo schermo, mentre si mantiene lo **stderr** assegnato allo schermo, così che l'utente possa immediatamente essere informato degli errori.

## 1.12 Note generali sul C e su questo libro

Il C è un linguaggio difficile. A volte, i programmatori C esperti sono orgogliosi di essere capaci di creare utilizzi bizzarri, contorti e complicati del linguaggio. Questa è una cattiva abitudine di programmazione. Essa rende i programmi difficili da leggere, maggiormente soggetti a comportamenti strani e più difficili da collaudare e correggere. Questo libro è adeguato ai programmatori principianti, perciò ci siamo sforzati di scrivere programmi chiari e ben strutturati. Uno dei nostri obiettivi principali, in questo libro, è quello di perseguire la *chiarezza* dei programmi, attraverso le comprovate tecniche di programmazione strutturata e le molte correlate buone abitudini di programmazione.



### Buona abitudine 1.1

*Scrivete i vostri programmi C in una maniera semplice e diretta. Questa regola è detta a volte KIS ("keep it simple", mantienilo semplice). Non "forzate" il linguaggio, tentando delle "stranezze".*

Potreste aver sentito dire che il C è un linguaggio portabile e che i programmi scritti in C possono essere eseguiti su molti computer differenti. *La portabilità è un obiettivo sfuggente*. Il documento dello standard ANSI (An90) annovera ben 11 pagine di insidiosi problemi riguardanti la portabilità. Sono stati scritti libri interi sull'argomento della portabilità nel C (Ja89) (Ra90).



### Obiettivo portabilità 1.3

*Per quanto sia possibile scrivere programmi portabili, ci sono molti problemi tra le tante implementazioni del C e i vari computer, che rendono la portabilità un obiettivo difficile da raggiungere. Scrivere semplicemente dei programmi in C non ne garantisce la portabilità.*

Abbiamo eseguito una attenta verifica del documento dello standard ANSI C e abbiamo confrontato con quello la nostra presentazione per completezza e accuratezza. Tuttavia, il C è un linguaggio molto ricco e ci sono alcune sottigliezze nel linguaggio e certi argomenti avanzati che non abbiamo trattato. Vi suggeriamo di leggere lo stesso documento dello standard ANSI C o il manuale di riferimento di Kernighan e Ritchie (Ke88), nel caso doveste aver bisogno di ulteriori dettagli tecnici sull'ANSI C.

Noi abbiamo preferito limitare la nostra discussione all'ANSI C. Molte caratteristiche incluse nell'ANSI C non sono compatibili con le altre implementazioni del C, perciò potreste verificare che alcuni dei programmi di questo testo non funzionano sui compilatori C più datati.



#### *Buona abitudine 1.2*

---

*Leggete i manuali della versione del C che state usando. Consultate frequentemente questi manuali per essere sicuri di conoscere la ricca collezione di caratteristiche del C e di usarle in modo corretto.*



#### *Buona abitudine 1.3*

---

*Il vostro computer e il vostro compilatore sono dei buoni insegnanti. Qualora non siate sicuri di come funzioni una caratteristica del C, scrivete un programma di esempio con quella caratteristica, compilatelo ed eseguitelo e osservate quello che succede.*

## **1.13 Il Concurrent C**

Altre versioni del C sono state sviluppate per mezzo degli sforzi continui della ricerca nei Bell Laboratories. Gehani (Ge89) ha sviluppato il *Concurrent C*: un superinsieme del C che include delle primitive per specificare delle attività multiple che potranno essere eseguite in parallelo. Entro i prossimi dieci anni, con l'aumento dell'utilizzo dei *multiprocessori* (ovverosia, dei computer con più di una CPU), i linguaggi come il Concurrent C e le caratteristiche dei sistemi operativi che supportano il parallelismo nelle applicazioni dell'utente, diventeranno sempre più popolari. Per quanto riguarda questo testo, il Concurrent C è ancora principalmente un linguaggio di ricerca. I corsi e i libri di testo sui sistemi operativi (De90) includono solitamente trattazioni corpose sulla programmazione concorrente.

## **1.14 La programmazione orientata agli oggetti e il C++**

Un altro superinsieme del C, ossia il C++, fu sviluppato da Stroustrup (St86) nei Bell Laboratories. Il C++ fornisce diverse caratteristiche che migliorano il linguaggio C. Ma, cosa più importante, fornisce delle primitive per fare una *programmazione orientata agli oggetti*.

Gli *oggetti* sono essenzialmente dei *componenti* software riusabili che modellano gli elementi del mondo reale. C'è una rivoluzione in preparazione nella comunità del software. Costruire il software velocemente, correttamente e in modo economico rimane un obiettivo sfuggente e questo in un periodo in cui aumentano vertiginosamente le richieste per un software nuovo e più potente.

Gli sviluppatori di software stanno scoprendo che, utilizzando un approccio modulare e orientato agli oggetti per la progettazione e l'implementazione, si potrebbe fare in modo che i gruppi di sviluppo siano da 10 a 100 volte più produttivi di quanto fosse possibile con le tecniche di programmazione convenzionali.

Sono stati sviluppati molti linguaggi di programmazione orientati agli oggetti. Molti docenti sono convinti che la miglior strategia educativa oggi sia di imparare a fondo il C e di studiare, in seguito, il C++.

## Esercizi di autovalutazione

- 1.1 Riempite gli spazi in ognuna delle seguenti frasi:
- L'azienda che ha scatenato il fenomeno della elaborazione personale nel mondo è stata la \_\_\_\_\_.
  - Il computer che ha legittimato l'elaborazione personale nel commercio e nell'industria è stato il \_\_\_\_\_.
  - I computer elaborano i dati sotto il controllo di insiemi di istruzioni chiamati \_\_\_\_\_ per computer.
  - Le sei principali unità logiche del computer sono \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_ e \_\_\_\_\_.
  - Il \_\_\_\_\_ è un caso speciale della multiprogrammazione in cui gli utenti accedono al computer attraverso dispositivi chiamati terminali.
  - Le tre classi di linguaggi discusse in questo capitolo sono \_\_\_\_\_, \_\_\_\_\_ e \_\_\_\_\_.
  - I programmi che traducono in linguaggio macchina quelli scritti in linguaggio di alto livello sono chiamati \_\_\_\_\_.
  - Il C è largamente conosciuto come il linguaggio di sviluppo del sistema operativo \_\_\_\_\_.
  - Questo libro presenta la versione del C chiamata \_\_\_\_\_ C, che è stata standardizzata recentemente dall'Istituto Nazionale Americano per gli Standard.
  - Il linguaggio \_\_\_\_\_ fu sviluppato da Wirth per insegnare la programmazione strutturata nelle università.
  - Il Dipartimento della Difesa sviluppò il linguaggio Ada con una caratteristica detta \_\_\_\_\_, che consente ai programmatori di specificare che molte attività possono procedere in parallelo.
- 1.2 Riempite gli spazi in ognuna delle seguenti frasi riguardanti l'ambiente C.
- I programmi C sono normalmente immessi in un computer usando un programma \_\_\_\_\_.
  - In un sistema C, prima che incominci la fase di traduzione, sarà eseguito automaticamente un programma \_\_\_\_\_.
  - I due tipi più comuni di direttive del preprocessore sono \_\_\_\_\_ e \_\_\_\_\_.
  - Il programma \_\_\_\_\_ combina l'output del compilatore con le varie funzioni di libreria, per produrre una immagine eseguibile.
  - Il programma \_\_\_\_\_ trasferisce l'immagine eseguibile dal disco alla memoria.
  - Per caricare ed eseguire il programma compilato più recentemente su un sistema UNIX, digitate \_\_\_\_\_.

## Risposte agli esercizi di autovalutazione

1.1 a) Apple. b) Personal Computer della IBM. c) programmi. d) unità di input, unità di output, unità di memoria, unità aritmetica e logica (ALU), unità di elaborazione centrale (CPU), unità di memoria secondaria. e) timesharing. f) linguaggi macchina, linguaggi assembly, linguaggi di alto livello. g) compilatori. h) UNIX. i) ANSI. j) Pascal. k) multitasking.

1.2 a) editor. b) preprocessore. c) includere altri file in quello da compilare, sostituire i simboli speciali con il testo del programma. d) linker. e) loader. f) **a.out**.

## Esercizi

- 1.3 Classificate ognuno dei seguenti elementi come hardware o software:
- CPU
  - compilatore C
  - ALU
  - preprocessore C
  - unità di input
  - un programma di elaborazione di testi.
- 1.4 Perché dovrete preferire scrivere un programma in un linguaggio indipendente dalla macchina, invece che in un linguaggio dipendente dalla macchina? Perché un linguaggio dipendente dalla macchina potrebbe essere più appropriato per scrivere certi tipi di programmi?
- 1.5 I programmi traduttori, come gli assembleri e i compilatori, convertono i programmi da un linguaggio (detto sorgente) in un altro (detto oggetto). Determinate quali delle seguenti affermazioni sono vere e quali sono false:
- Un compilatore traduce in linguaggio oggetto i programmi scritti in un linguaggio di alto livello.
  - Un assembler convertite in programmi in linguaggio macchina quelli scritti in linguaggio sorgente.
  - Un compilatore convertite in programmi in linguaggio oggetto quelli scritti in linguaggio sorgente.
  - I linguaggi di alto livello sono generalmente indipendenti dalla macchina.
  - Un programma in linguaggio macchina richiede una traduzione, prima che il programma possa essere eseguito su un computer.
- 1.6 Riempite gli spazi bianchi delle seguenti frasi:
- I dispositivi attraverso i quali gli utenti accedono ai sistemi di computer in timesharing sono chiamati di solito \_\_\_\_\_.
  - Un programma per computer che converte in programmi in linguaggio macchina quelli scritti in linguaggio assembly è chiamato \_\_\_\_\_.
  - L'unità logica del computer, che riceve dall'esterno le informazioni affinché il calcolatore le possa utilizzare, si chiama \_\_\_\_\_.
  - Il processo di istruzione del computer per risolvere problemi specifici è chiamato \_\_\_\_\_.
  - Quale tipo di linguaggio per computer utilizza delle abbreviazioni simili all'inglese per le istruzioni in linguaggio macchina? \_\_\_\_\_.
  - Quali sono le sei unità logiche del computer? \_\_\_\_\_.
  - Quale unità logica del computer invia ai vari dispositivi le informazioni che sono già state elaborate dallo stesso, in modo che queste possano essere utilizzate all'esterno del computer? \_\_\_\_\_.
  - Il nome generico di un programma, che converte in linguaggio macchina i programmi scritti in un certo linguaggio per computer, è \_\_\_\_\_.
  - Quale unità logica del computer conserva le informazioni? \_\_\_\_\_.
  - Quale unità logica del computer esegue i calcoli? \_\_\_\_\_.
  - Quale unità logica del computer prende le decisioni logiche? \_\_\_\_\_.
  - L'abbreviazione comunemente utilizzata per l'unità di controllo del computer è \_\_\_\_\_.
  - Il livello del linguaggio per computer più conveniente per il programmatore, per scrivere i programmi più velocemente e facilmente, è \_\_\_\_\_.
  - Il linguaggio orientato alle comuni applicazioni commerciali usato più diffusamente oggi-giorno è il \_\_\_\_\_.
  - L'unico linguaggio che un computer può comprendere direttamente è il \_\_\_\_\_ del computer.

- p) Quale unità logica del computer coordina le attività di tutte le altre unità logiche?  
 -----
- 1.7 Determinate se ognuna delle seguenti affermazioni sia vera o falsa. Spiegate le vostre risposte.
- I linguaggi macchina dipendono generalmente dalla macchina.
  - Il timesharing consente realmente a molti utenti di usare simultaneamente un computer.
  - Come gli altri linguaggi di alto livello, il C è generalmente considerato indipendente dalla macchina.
- 1.8 Discutete il significato di ognuno dei seguenti nomi in ambiente UNIX:
- stdin**
  - stdout**
  - stderr**
- 1.9 Quale caratteristica chiave è fornita dal Concurrent C che non sia disponibile in ANSI C?
- 1.10 Perché oggi giorno c'è così tanta attenzione concentrata sulla programmazione orientata agli oggetti, in generale, e sul C++ in particolare?

## Letture consigliate

- (An90) ANSI, *American National Standard for Information Systems—Programming Language C (ANSI Document ANSI/IISO 9899: 1990)*, New York, NY: American National Standards Institute, 1990.  
 Questo è il documento che definisce l'ANSI C. Il documento è disponibile in vendita presso l'American National Standards Institute, 1430 Broadway, New York, New York 10018.
- (De90) Deitel, H. M. , *Operating Systems* (Second Edition), Reading, MA: Addison-Wesley Publishing Company, 1990.  
 È un manuale per il tradizionale corso di informatica sui sistemi operativi. I capitoli 4 e 5 presentano una approfondita discussione sulla programmazione concorrente.
- (Ge89) Gehani, N. , and W. D. Roome, *The Concurrent C Programming Language*, Summit, NJ: Silicon Press, 1989.  
 Questo è il libro di definizione del Concurrent C: un superinsieme del linguaggio C, che consente ai programmatori di specificare una esecuzione parallela di attività multiple. È anche incluso un compendio del Concurrent C++.
- (Ja89) Jaeschke, R. , *Portability and the C Language*, Indianapolis, IN: Hayden Books, 1989.  
 Questo libro discute la scrittura in C di programmi portabili. Jaeschke ha partecipato ai comitati ANSI e ISO per gli standard del C.
- (Ke88) Kernighan, B. W. , and D. M. Ritchie, *The C Programming Language* (Second Edition), Englewood Cliffs, NJ: Prentice Hall, 1988.  
 Questo libro è un classico nel settore. Il libro è ampiamente utilizzato, nei corsi di C e nei seminari per i programmatori di ruolo, e include un eccellente manuale di riferimento. Ritchie è l'autore del linguaggio C e uno dei progettisti del sistema operativo UNIX.
- (P192) Plauger, P. J., *The Standard C Library*, Englewood Cliffs, NJ: Prentice Hall, 1992.  
 Definisce e dimostra l'utilizzo delle funzioni incluse nella libreria standard del C. Plauger, nell'ambito del comitato che ha sviluppato lo standard ANSI C, è stato il capo del sottocomitato per la libreria ed è anche stato colui il quale ha convocato il comitato ISO per l'evoluzione del C.
- (Ra90) Rabinowitz, H., and C. Schaap, *Portable C*, Englewood Cliffs, NJ: Prentice Hall, 1990.  
 Questo libro è stato sviluppato per un corso sulla portabilità tenuto negli AT&T Bell Laboratories. Rabinowitz lavora presso il Laboratorio di Intelligenza Artificiale della NINEX Corporation, mentre Schaap è un dirigente della Delft Consulting Corporation.
- (Ri78) Ritchie, D. M. ; S. C. Johnson; M. E. Lesk; and B. W. Kernighan, "UNIX Time-Sharing System: The C Programming Language", *The Bell System Technical Journal*, Vol. 57, No. 6, Part 2, July—August 1978, pp. 1991-2019.  
 Questo è uno degli articoli classici di introduzione al linguaggio C. Apparve in un numero speciale della rivista *Bell System Technical Journal* dedicata al "Sistema di timesharing in UNIX".

- (Ri84) Ritchie, D. M. , “The UNIX System: The Evolution of the UNIX Time-Sharing System”, *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8, Part 2, October 1984, pp. 1577-1593.  
Un articolo classico sul sistema operativo UNIX. Questo articolo apparve in una edizione speciale della rivista *Bell System Technical Journal* completamente dedicata al “Sistema UNIX”.
- (Ro84) Rosler, L., “The UNIX System: The Evolution of C—Past and Future”, *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8, Part 2, October 1984, pp. 1685-1699.  
È un eccellente articolo, da far seguire a (Ri78), per il lettore interessato a tracciare la storia del C e le radici dello sforzo per standardizzare l’ANSI C. Apparve in una edizione speciale della rivista *Bell System Technical Journal* dedicata al “Sistema UNIX”.
- (St84) Stroustrup, B., “The UNIX System: Data Abstraction in C”, *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8, Part 2, October 1984, pp. 1701-1732.  
L’articolo classico di introduzione al C++. Comparve in una edizione speciale della rivista *Bell System Technical Journal* dedicata al “Sistema UNIX”.
- (St91) Stroustrup, B. *The C++ Programming Language* (Second Edition), Reading, MA: Addison-Wesley Series in Computer Science, 1991.  
Questo libro è il riferimento per la definizione del C++: un superinsieme del C, che apporta vari miglioramenti al C, specialmente le caratteristiche per la programmazione orientata agli oggetti. Stroustrup ha sviluppato il C++ negli AT&T Bell Laboratories.
- (To89) Tondo, C. L., and S. E. Gimpel, *The C Answer Book*, Englewood Cliffs, NJ: Prentice Hall, 1989.  
Questo libro unico fornisce le risposte agli esercizi nel Kernighan e Ritchie (Ke88). Gli autori mostrano uno stile di programmazione esemplare, mentre forniscono spiegazioni sui loro approcci alla soluzione dei problemi e sulle scelte di progettazione. Tondo lavora presso la IBM Corporation e la Nova University in Ft. Lauderdale, Florida. Gimpel è un consulente.

