Introduzione

Imparare il linguaggio C è come fare viaggio in una terra lontana e sconosciuta; è dunque sia un viaggio meraviglioso, affascinante e gratificante, ricco di sorprese e scoperte, sia un viaggio non semplice, che richiede perciò tanta pazienza e il giusto tempo.
In ogni caso, al termine del viaggio, la ricompensa ricevuta sarà elevata: si sarà appreso non un linguaggio di programmazione qualsiasi ma *il* linguaggio di programmazione per eccellenza che già antecedentemente alla sua prima standardizzazione, avvenuta nel lontano 1989 (ANSI C), aveva iniziato a entusiasmare una vasta platea di programmatori.
Come avremo modo di verificare durante tutto il percorso di apprendimento che il libro intende offrire, C è un linguaggio estremamente espressivo e sintetico che dà grande "fiducia" e libertà operativa al programmatore, il cui unico limite potrà essere dato solo dalla poca fantasia o dalla scarsa preparazione sulle regole sintattiche dei costrutti o sulla semantica delle operazioni.
Dal punto di vista più pratico, e forse meno filosofico, imparare C consente di sviluppare programmi di uso generale, davvero a 360 gradi, programmi cioè utili per qualsiasi ambito applicativo (sistemi operativi, robotica, database, networking, grafica e così via). Esso quindi si dimostra, ancora oggi, dove è presente una pletora di altri linguaggi di programmazione che promettono di essere più *easy* e più *safe*, lo strumento di eccellenza adoperato da milioni di programmatori che apre le porte del mondo della programmazione *reale*, sicuramente più *hard* ma anche più appagante.
Last but not least, C è un impressionante e imprescindibile strumento didattico usato soprattutto dalle università più giudiziose per insegnare sia i fondamenti della programmazione ma anche come è fatto, a "basso livello", un calcolatore elettronico (si pensi ai puntatori, la cui disamina non può prescindere da una spiegazione approfondita di cos'è e come è organizzata la memoria di un elaboratore).
Desideriamo, inoltre, spendere qualche parola sui principi ispiratori che hanno guidato l'autore nella scrittura del presente testo.

- Gli argomenti propri di ogni capitolo hanno un preciso e chiaro ordine. Ogni capitolo esprimerà compiutamente il relativo obiettivo didattico e non si sovrapporrà ai contenuti di altri capitoli ne li comprenderà. Per esempio, se nel Capitolo 2 si parlerà delle variabili, solo nel successivo Capitolo 3 si parlerà degli array; allo stesso modo, solo dopo aver trattato anche delle funzioni nel Capitolo 6, si parlerà dei puntatori nel Capitolo 7 e di tutte le loro *relazioni* e utilizzi con le variabili, gli array e le funzioni. Quanto detto può apparire abbastanza ovvio ma non lo è. Infatti oggi si sta

assistendo alla proliferazione di testi con argomenti scritti "a spirale", dove cioè un argomento può contenere riferimenti iniziali ad altri argomenti i quali saranno poi trattati approfonditamente solo nel capitolo di pertinenza. Questo, a parere dell'autore, laddove non strettamente necessario e comunque non legato ai costrutti del linguaggio (è evidente che per mostrare il valore di una variabile bisogna dire qualcosa di propedeutico sull'istruzione `printf`), induce a distrazioni e fa perdere inutilmente tempo nella corretta comprensione della corrente unità didattica.

- Il modo espositivo seguito è rigoroso, laddove necessario piuttosto formale, e si è prestata molta attenzione al corretto uso della terminologia propria del linguaggio. Questo non è un libro del tipo "Impariamo C in 24 ore" oppure "C For Dummies". I libri che pretendono di insegnare C in quel modo molto probabilmente sono solo uno specchio per le allodole; illudono, dando false promesse. C è un linguaggio complesso e ricco di sfumature; per insegnarlo, ci vogliono "serietà" e il giusto rigore; per impararlo, ci vogliono pazienza e disciplina.
- I listati e gli snippet di codice sono stati pensati in modo che diano una chiara indicazione pratica dei relativi argomenti teorici; sono piuttosto brevi, autoconclusivi e supportati da ulteriori commenti che danno, talune volte, ulteriori spiegazioni teoriche.

Organizzazione del libro

Il libro è organizzato nei capitoli elencati di seguito.

- Capitolo 1, "Introduzione al linguaggio C": introduciamo il lettore ad alcuni concetti propedeutici del calcolatore elettronico e dello sviluppo di un programma in C. Redigiamo anche un primo programma e mostriamo come compilarlo ed eseguirlo.
- Capitolo 2, "Variabili, costanti, letterali e tipi": parliamo delle variabili, delle costanti e dei tipi di dato fondamentali del linguaggio. Chiudiamo con una trattazione delle conversioni di tipo e di un confronto tra `typedef` e `#define`.
- Capitolo 3, "Array": analizziamo l'importante struttura dati array nella sua forma monodimensionale e multidimensionale. Parliamo altresì degli array di lunghezza variabile, degli array costanti e dell'applicazione dell'operatore `sizeof` per ottenere la dimensione di un array.
- Capitolo 4, "Operatori": mostriamo tutti gli operatori che il linguaggio mette a disposizione per manipolare i dati; dai semplici operatori aritmetici a quelli complessi *bitwise*. Chiude un'utile tabella di precedenza degli operatori.
- Capitolo 5, "Strutture di controllo": vediamo come gestire il flusso di esecuzione di un programma attraverso le istruzioni di selezione, di iterazione e di salto.
- Capitolo 6, "Funzioni": trattiamo del fondamentale costrutto di funzione. Vediamo come essa si dichiara e si definisce, cosa sono i parametri formali, come si invoca e cosa sono i parametri attuali o argomenti. Analizziamo in dettaglio l'importante istruzione `return` e cos'è la ricorsione.
- Capitolo 7, "Puntatori": parliamo in grande dettaglio dei puntatori e della loro relazione con gli array. Vediamo anche cosa sono i puntatori a puntatori, i puntatori

a funzione, i puntatori a `void` e i puntatori nulli. Infine illustriamo in che modo le keyword `const` e `restrict` influenzino un puntatore e la conversione tra puntatori.

- Capitolo 8, "Strutture, unioni ed enumerazioni": descriviamo le strutture, le unioni e le enumerazioni evidenziando le loro differenze e quali sono i comuni casi di utilizzo.
- Capitolo 9, "Dichiarazioni": approfondiamo i concetti legati alle variabili, ai blocchi, allo scope e al *linkage*. Diamo uno sguardo di insieme agli specificatori della classe di memorizzazione, ai qualificatori di tipo, agli specificatori di tipo, agli specificatori di funzione e agli specificatori di allineamento.
- Capitolo 10, "Preprocessore e attributi": parliamo di una delle peculiarità di C, ossia il suo preprocessore, indicando tutte le direttive che mette a disposizione, dalla più comune `#define` a quella più *esoterica* `#pragma`. Trattiamo anche gli attributi, una delle novità di C23, che rappresentano un meccanismo che permette di aggiungere metadati al codice per indicare al compilatore informazioni aggiuntive su una funzione, una variabile, un tipo o altra entità, che può utilizzare per ottimizzare il codice, generare avvisi, aumentare le performance e altro ancora.
- Capitolo 11, "La libreria standard": analizziamo alcune funzionalità della libreria standard del C, focalizzandoci sulla gestione delle stringhe, dell'input e dell'output formattato e della memoria.

Struttura del libro e convenzioni

Gli argomenti del libro sono organizzati in capitoli. Ogni capitolo è numerato in ordine progressivo e denominato significativamente in base al suo obiettivo didattico (per esempio, Capitolo 2, "Variabili, costanti, letterali e tipi").
I capitoli sono poi suddivisi in paragrafi di pertinenza, al cui interno possiamo avere dei blocchi di testo o di grafica, a supporto alla teoria, denominati in accordo con il seguente schema: *Tipologia NrCapitolo.NrProgressivo* (*NomeFileSorgente*$_{opt}$) *Descrizione*$_{opt}$.

- *Tipologia* può indicare: `Listato` (un listato di codice sorgente), `Snippet` (un frammento di codice sorgente), `Sintassi` (la sintassi di un costrutto C), `Shell` (un comando di shell), `Output` (l'output di un listato o di uno snippet), `Figura` (un'immagine), `Tabella` (una tabella).
- *NrCapitolo.NrProgressivo* indica il numero di capitolo e il numero progressivo della relativa tipologia di blocco di testo o di grafica.
- (*NomeFileSorgente*) indica, opzionalmente, il nome del file di codice sorgente se il blocco di testo rappresenta un listato o uno snippet.
- *Descrizione* indica, opzionalmente, una descrizione del blocco di testo o di grafica.

Per esempio, il blocco denominato `Listato 6.10 (VariableArgumentsList.c)` indica il decimo listato di codice del Capitolo 6 avente come nome del file di codice sorgente `VariableArgumentsList.c` e nessuna descrizione. Oppure il blocco `Snippet 3.12 (Snippet_3_12.c) Array Costanti` indica il dodicesimo snippet di codice del Capitolo 3 avente come nome del file di codice sorgente `Snippet_3_12.c` e come descrizione `Array Costanti`.
Per quanto attiene ai listati, abbiamo adottato la seguente convenzione: i puntini di sospensione (…) eventualmente presenti indicano che in quel punto sono state omesse

alcune parti del listato, presenti però nei relativi file `.c` allegati al libro. Gli stessi caratteri possono talvolta trovarsi anche negli output di un programma eccessivamente lungo.

NOTA

I comandi di shell devono essere scritti *così come sono*, ossia senza inserire i caratteri di fine riga che invece sono presenti nel libro per garantire un'adeguata formattazione del testo.

Codice sorgente

Tutti i listati e gli snippet di codice sono disponibili sul sito di Apogeo all'indirizzo `https://bit.ly/apo-moc` e su GitHub al seguente indirizzo: `https://github.com/thp1972/Libro_C23/tree/main/Codice`. Dopo aver aperto la pagina web indicata, possiamo scaricarli facendo clic sul pulsante *Code* e poi su *Download ZIP* (Figura P.1).
Alternativamente possiamo creare in locale (*cloning*) una copia del repository remoto del codice sorgente tramite un qualsiasi client Git.

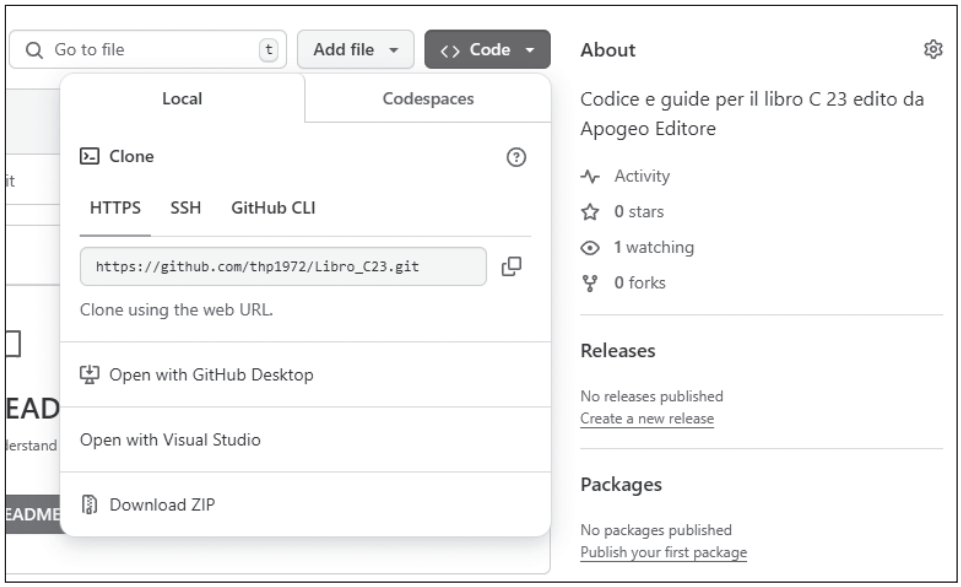

Figura P.1 Scaricare direttamente un repository tramite un archivio ZIP.

DETTAGLIO

GitHub è, principalmente, una piattaforma web *cloud-based* dove è possibile archiviare, gestire e condividere progetti software grazie all'utilizzo di un sistema di controllo delle versioni (*version control system*) distribuito basato su Git, software sviluppato dallo stesso creatore di Linux, Linus Torvalds. Un sistema di controllo della versione è, in breve, un software che consente di tenere traccia nel tempo delle modifiche dei file di un progetto (per esempio, i file di codice sorgente), ripristinarli a una determinata versione se qualcosa non

funziona più, far collaborare più persone su uno stesso progetto contemporaneamente, proteggere i file da perdite locali accidentali e così via. Infine, un *repository* è una sorta di cartella o archivio *virtuale* di un progetto che contiene file, codice sorgente, documentazione, immagini e più in generale tutti quegli asset che necessitano di un versioning.

Dopo aver ottenuto i file di codice sorgente, la struttura delle directory sarà la seguente: `Libro_C23 > Codice > Capitolo_01`, `Capitolo_02` e così via. Ogni capitolo avrà una propria cartella, al cui interno saranno presenti ulteriori sottocartelle contenenti il codice sorgent C divise per listati (`Listati`) e snippet di codice (`Snippets`).
Mi preme inoltre evidenziare i seguenti punti.

- Ho deciso di non imporre l'utilizzo di un IDE (*Integrated Development Environment*) specifico, come NetBeans, Eclipse o Visual Studio Code, preferendo invece adottare un approccio più "a basso livello" che si concretizza nell'insegnare a compilare ed eseguire codice C direttamente dalla riga di comando utilizzando i tool nativi del linguaggio. Tale scelta è motivata dalla natura *cross-platform* di C: le nozioni apprese potranno essere riutilizzate senza sforzi aggiuntivi e in modo coerente su qualsiasi sistema operativo, sia esso Windows, GNU/Linux o macOS.
- Ho scelto gli strumenti di compilazione propri della suite *open source* GCC (*GNU Compiler Collection*) disponibile per tutti i suddetti sistemi operativi.
- Ho impiegato come sistema operativo di riferimento GNU/Linux, distribuzione Fedora 41. I risultati dei comandi di compilazione o esecuzione del codice saranno infatti mostrati come output di una shell propria di questo sistema.

IMPORTANTE

All'indirizzo `https://github.com/thp1972/Libro_C23/blob/main/Guide/GCC/README.md` è presente una guida su GCC nella quale vengono trattate la sua installazione e configurazione ma anche le fasi di compilazione, linking, esecuzione e debugging del codice C.

NOTA

Esistono diverse raccolte di strumenti per compilare ed eseguire codice scritto in C. Tra le principali citiamo: *Clang*, compilatore nativo C/C++/Objective-C, sotto-progetto dell'infrastruttura LLVM; *Microsoft Visual C++*, compilatore C/C++ utilizzato specificamente per lo sviluppo di applicazioni Windows; *Intel oneAPI DPC++/C++ Compiler*, compilatore C/C++ altamente performante e ottimizzato per le piattaforme basate sui processori Intel; *Pelles C*, compilatore C/C++, con annesso IDE integrato, disponibile però solo su sistemi Windows.