
Prefazione

In questo volume completiamo l'esplorazione del linguaggio C++ che abbiamo iniziato in *C++ Fondamenti di programmazione*. I due testi fanno parte di un percorso didattico unitario, come testimoniano i frequenti riferimenti incrociati (soprattutto negli esercizi di fine capitolo); abbiamo fatto il possibile, comunque, per renderli utilizzabili anche separatamente.

Il volume *Fondamenti* si concentrava sugli aspetti di base della programmazione in C++. Questo volume introduce tecniche e metodologie più avanzate.

Il testo segue lo standard ANSI del C++; tenete presente che molte funzionalità previste nell'ANSI non sono implementate nelle versioni precedenti C++. Per avere informazioni più dettagliate sul linguaggio vi conviene consultare il manuale di riferimento del vostro sistema o procurarvi una copia del documento ANSI/ISO 9899: 1990, "American National Standard for Information Systems-Programming Language C", dell'American National Standards Institute, 11 West 42nd Street, New York, New York 10036.

La metodologia di insegnamento

Questo libro contiene un'ampia varietà di esempi, esercizi e progetti che prendono spunto da molte situazioni e che offrono agli studenti la possibilità di risolvere problemi reali. Il libro prende in esame i principi della progettazione del software, insistendo sull'importanza della chiarezza dei programmi ed evitando l'uso di terminologia complessa a favore di esempi chiari e diretti, il cui codice sia stato collaudato sulle piattaforme C++ più diffuse.

L'apprendimento attraverso il codice

Il libro presenta una grande quantità di esempi basati sul codice. Ogni argomento viene presentato nell'ambito di un programma C++ completo e funzionante, seguito sempre da una o più finestre che mostrano l'output del programma in questione. Viene quindi usato il linguaggio per insegnare il linguaggio, e la lettura di questi programmi offre un'esperienza molto simile alla loro esecuzione reale su di un computer.

L'accesso al World Wide Web

Tutto il codice presente nel libro si trova anche in Internet, nel booksite abbinato a questo libro, all'indirizzo <http://www.apogeeonline.com/education/booksite>. Il nostro consiglio è quello di scaricare tutto il codice, eseguendo poi ogni singolo programma via via che appare nel testo. Potete anche modificare il codice degli esempi e vedere cosa succede, imparando così a programmare programmando. Tutto questo materiale è protetto da diritti d'autore, quindi utilizzatelo liberamente per studiare il C++, ma non pubblicatene alcuna parte senza l'esplicito permesso da parte degli autori e della casa editrice.

Obiettivi

Ogni capitolo inizia con la presentazione degli *Obiettivi*. Gli studenti possono così sapere in anticipo ciò che andranno ad apprendere e, alla fine della lettura del capitolo, potranno verificare se hanno raggiunto o meno questi obiettivi.

Il codice e gli esempi

Le funzionalità del C++ vengono presentate nell'ambito di programmi completi e funzionanti. Ogni programma è seguito dalle immagini degli output che vengono prodotti, così che gli studenti possano assicurarsi della correttezza dei risultati. I programmi presentati vanno da poche linee di codice a esempi composti da varie centinaia di righe. Gli studenti dovrebbero scaricare tutto il codice dal sito Web <http://www.apogonline.com/education/booksite>, eseguendo poi ogni programma via via che questo viene presentato all'interno del testo.

Figure e immagini

Il libro offre un'ampia varietà di grafici, immagini e output di programmi. Nelle sezioni dedicate alle strutture di controllo, per esempio, appaiono vari diagrammi di flusso molto utili. [Nota: I diagrammi di flusso non vengono presentati come uno strumento di sviluppo, ma ricorriamo a una breve presentazione basata su questi diagrammi per specificare le singole operazioni di ogni struttura di controllo C++.]

Consigli e suggerimenti

Il libro offre molti suggerimenti riguardanti la programmazione, per aiutare gli studenti a concentrarsi sugli aspetti più importanti dello sviluppo di programmi. Questi consigli vengono forniti attraverso le sezioni chiamate *Buona abitudine*, *Errore tipico*, *Collaudo e messa a punto*, *Obiettivo efficienza*, *Obiettivo portabilità*, *Ingegneria del software*.



Buona abitudine

Il concetto più importante per chi inizia a programmare è la chiarezza, e all'interno delle sezioni *Buona abitudine* vengono presentate delle tecniche per la scrittura di programmi chiari, comprensibili e più facilmente gestibili.



Errore tipico

Tutti gli studenti che affrontano per la prima volta un nuovo linguaggio tendono a commettere frequentemente gli stessi errori. Le sezioni *Errore tipico* aiutano gli studenti a evitare di commettere i più comuni.



Collaudo e messa a punto

Queste sezioni forniscono consigli circa le attività di test e debugging dei programmi C++, anche a livello preventivo.



Obiettivo efficienza

In base alla nostra esperienza, insegnare agli studenti come scrivere programmi chiari e comprensibili deve costituire l'obiettivo più importante di qualsiasi corso di programmazione. Gli studenti, però, vogliono normalmente imparare a scrivere programmi che vengano eseguiti in modo veloce, che utilizzino poca memoria, che richiedano una minima quantità di comandi e che offrano prestazioni eccellenti. Le sezioni *Obiettivo efficienza* offrono suggerimenti su come migliorare le prestazioni dei propri programmi.



Obiettivo portabilità

Alcuni programmatori pensano che, implementando un'applicazione C++, questa sia immediatamente portabile su tutte le piattaforme; sfortunatamente, non è sempre così. Le sezioni *Obiettivo portabilità* aiutano gli studenti a scrivere codice realmente portabile, fornendo inoltre informazioni su come il C++ sia in grado di raggiungere questo elevato livello di portabilità.



Ingegneria del software

Il C++ è un linguaggio molto efficace nell'ambito della progettazione del software, e le sezioni *Ingegneria del software* prendono in esame gli aspetti architetturali e progettuali che influiscono sulla realizzazione di sistemi software, specialmente nel caso di sistemi su vasta scala. Molte di queste informazioni saranno utili agli studenti nei corsi più avanzati, oltre che nel mondo del lavoro.

Esercizi di autovalutazione

Il libro propone molti esercizi corredati di risposte, così che gli studenti possano prepararsi alle esercitazioni vere e proprie. Gli studenti dovrebbero essere incoraggiati a svolgere tutti questi esercizi di autovalutazione.

Esercizi

Ogni capitolo si conclude con un insieme di esercizi di vario tipo, che permettono agli insegnanti di adattare le proprie lezioni alle esigenze particolari di ogni classe. Gli esercizi verificano l'apprendimento dei concetti e dei termini più importanti, chiedono agli studenti di scrivere singole istruzioni, piccole porzioni di funzioni, funzioni e programmi completi fino ad arrivare alla costruzione di interi progetti.

Indice analitico

Alla fine del libro è possibile trovare un indice analitico completo, molto utile sia a chi legge questo libro per la prima volta che ai programmatori che lo usano come riferimento.

Panoramica del libro

Capitolo 1 – I template. Questo capitolo parla di una caratteristica piuttosto recente di un linguaggio sempre in evoluzione come il C++. I template di funzione sono stati già introdotti brevemente nel Capitolo 3 del volume Fondamenti, questo capitolo ne presenta altri esempi. I template di classe consentono invece di cogliere l'essenza di un tipo di dato astratto o ADT (per esempio pila, array, o coda) e di creare, con una minima aggiunta di codice, versioni di tale ADT per i tipi specifici necessari (ad esempio coda di **int**, coda di **float**, coda di **string** e così via). Per questo motivo le istanze dei template di classe sono anche dette *tipi parametrizzati*. Il capitolo mostra l'uso dei parametri "di tipo" e "non di tipo" e illustra l'interazione tra i template e altri concetti del linguaggio come l'ereditarietà, le funzioni **friend**, e i membri **static**. Gli esercizi chiedono allo studente di scrivere diversi template di funzione e di classe, e di impiegarli in programmi completi. La trattazione di questo argomento viene ripresa nel Capitolo 9 su container, iteratori e algoritmi: la libreria standard dei template (STL).

Capitolo 2 – La gestione delle eccezioni. Questo capitolo parla di uno dei miglioramenti più recenti apportati al linguaggio. La gestione delle eccezioni consente di scrivere programmi più robusti, più capaci di gestire le condizioni di errore e più adatti alle aree di applicazione critiche di un'azienda. Il capitolo discute quando è appropriato utilizzare le eccezioni, introduce i concetti di base della gestione delle eccezioni con i blocchi **try**, le istruzioni **throw** e i blocchi **catch**, indica come e quando rilanciare un'eccezione, spiega come scrivere una specifica d'eccezione ed elaborare eccezioni inattese ed infine discute il legame importante tra eccezioni, costruttori, distruttori ed ereditarietà. A fine capitolo c'è una sezione di 43 esercizi su programmi che illustrano la diversità e le potenzialità delle eccezioni in C++.

Capitolo 3 – L'elaborazione dei file. Questo capitolo discute le tecniche di elaborazione dei file di testo ad accesso sequenziale e casuale. All'inizio c'è un'introduzione alla gerarchia dei dati bit/byte/campi/record/file. Viene quindi presentata una panoramica dei file e degli stream del C++. I file ad accesso sequenziale sono discussi con una serie di tre programmi che mostrano come aprirli e chiuderli, come memorizzarvi i dati sequenzialmente e come leggerli sequenzialmente. I file ad accesso casuale sono discussi tramite una serie di quattro programmi che mostrano come creare un file sequenziale ad accesso casuale e come leggervi e scrivervi dati tramite l'accesso casuale. Il quarto programma combina le due tecniche di accesso, sequenziale e casuale, in un programma completo di gestione delle transazioni. Molti partecipanti ai seminari che teniamo nelle aziende ci hanno confidato che grazie a questo capitolo sono stati in grado di creare programmi di elaborazione dei file efficaci ed immediatamente utili nelle loro organizzazioni. Gli esercizi chiedono allo studente di implementare diversi programmi di creazione ed elaborazione di file ad accesso sia sequenziale che casuale.

Capitolo 4 – Le strutture dati. Questo capitolo discute le tecniche per creare e incapsulare strutture dati dinamiche. Il capitolo inizia con una discussione su classi ricorsive e allocazione dinamica della memoria, per proseguire con una discussione sul modo in cui si creano e mantengono diverse strutture dati dinamiche, tra cui le liste concatenate, le code, le pile e gli alberi. Per ciascun tipo di struttura dati presentiamo un programma completo e funzionante, mostrandone anche qualche output di esempio. Questo capitolo è un valido strumento per migliorare la propria padronanza dei puntatori, in quanto include numerosi

esempi sulla risoluzione del riferimento singola e doppia, un concetto particolarmente difficile. Un problema che si presenta con i puntatori è che gli studenti non riescono facilmente a visualizzare le strutture dati ed il modo in cui sono collegati tra loro i nodi. Per questo motivo abbiamo incluso alcune illustrazioni che mostrano i collegamenti e la sequenza in cui sono creati. L'esempio sugli alberi binari rappresenta una pietra miliare nello studio di puntatori e strutture dati dinamiche: in esso si crea un albero binario, si eliminano i valori duplicati e si introduce l'attraversamento dell'albero con visita ordinata, simmetrica e posticipata. Gli studenti provano un genuino senso di soddisfazione quando studiano ed implementano questo esempio ed apprezzano in particolare la visita ordinata, perché visualizza appunto il valore dei nodi in ordine. Il capitolo include numerosi esercizi, tra cui segnaliamo senz'altro quello intitolato "Costruite il vostro compilatore". Altri esercizi chiedono allo studente di sviluppare un programma di conversione dalla notazione infissa a quella postfissa ed un altro programma di valutazione di un'espressione postfissa, quindi modifichiamo l'algoritmo di valutazione della notazione postfissa per generare codice in linguaggio macchina. Il compilatore pone questo codice in un file (utilizzando le tecniche del Capitolo 1 del volume Fondamenti) e gli studenti lo eseguono sul simulatore software che hanno sviluppato nell'esercizio del Capitolo 5 del volume Fondamenti. I 35 esercizi includono la simulazione di una coda al supermercato, la ricerca ricorsiva in una lista, la visualizzazione ricorsiva degli elementi di una lista in ordine inverso, l'eliminazione di un nodo da un albero binario, l'attraversamento di un albero binario per livelli, la visualizzazione di diversi alberi binari, la scrittura di una porzione di un compilatore ottimizzato, la scrittura di un interprete, l'inserimento/eliminazione in qualsiasi punto di una lista concatenata, l'implementazione di liste e code senza puntatori in coda, l'analisi delle prestazioni della ricerca e dell'ordinamento in un albero binario ed infine l'implementazione di una classe di liste indicizzate. Questo capitolo è stato rivisto ma non migliorato in modo sostanziale, perché conteneva già una buona trattazione sulle strutture dati più importanti. Dopo averlo studiato, il lettore è pronto per lo studio dei container, degli iteratori e degli algoritmi della STL, che illustriamo nel Capitolo 9. I container della STL sono strutture dati preimpacchettate in forma di template che sono utilizzabili nella stragrande maggioranza di applicazioni. La STL rappresenta un grande passo avanti nella concezione del riutilizzo del software.

Capitolo 5 - Bit, caratteri, stringhe e strutture. Questo capitolo presenta diverse caratteristiche importanti. Le potenti funzionalità del C++ di manipolazione dei bit consentono di scrivere programmi che sfruttano le macchine a un livello più vicino all'hardware. Ciò consente di elaborare stringhe di bit, di impostare bit individuali su 1 o 0 e di memorizzare le informazioni in maniera più compatta. Queste funzionalità si trovano generalmente soltanto in linguaggi a basso livello come i linguaggi assembly ma sono di una certa utilità ai sistemisti che scrivono i sistemi operativi o il software di rete. Come ricorderete, nel Capitolo 5 del volume Fondamenti abbiamo introdotto la manipolazione delle stringhe in stile C (**char ***), presentando alcune delle funzioni più utilizzate. In questo capitolo continuiamo la presentazione dei caratteri e delle stringhe in stile C, presentando varie funzioni di manipolazione dei caratteri della libreria **cctype** per verificare se un carattere è una cifra, un carattere alfabetico, un carattere alfanumerico, una cifra esadecimale, una lettera minuscola, una lettera maiuscola e così via. Presentiamo poi le restanti funzioni di manipolazione di stringhe delle altre librerie correlate, come al solito ognuna nel contesto di un programma in C++ completo e funzionante. Le **struct** sono simili ai record del Pascal e di altri linguaggi in quanto aggregazioni di dati di tipi diversi. Le **struct** sono utilizzate nel

Capitolo 3 per creare file di informazioni: esse sono utilizzate congiuntamente ai puntatori e all'allocazione dinamica della memoria nel Capitolo 4 per formare strutture dati dinamiche come le liste concatenate, le code, le pile e gli alberi. Segnaliamo in questo capitolo la simulazione del mescolamento e della distribuzione di carte da gioco nella sua versione rivista e ottimizzata. I docenti dovrebbero cogliere l'occasione a questo punto per porre l'accento sulla qualità degli algoritmi. I 36 esercizi chiedono allo studente di cimentarsi con la maggior parte delle funzionalità discusse nel capitolo. Segnaliamo l'esercizio che verte sullo sviluppo di un programma di correzione ortografica automatica. I primi cinque capitoli del volume Fondamenti e i capitoli dal 5 al 7 di questo volume illustrano in realtà la porzione di C++ ereditata dal C. In particolare questo capitolo illustra in maniera più approfondita le stringhe in stile C (**char ***) a beneficio dei programmatori che lavorano spesso su codice C ereditato. Il Capitolo 8, lo ripetiamo, discute la classe **string** e la manipolazione delle stringhe viste come oggetti a tutti gli effetti.

Capitolo 6 – Il preprocessore. Questo capitolo contiene una discussione dettagliata delle direttive al preprocessore e alcune informazioni aggiuntive sulla direttiva **#include**, che consente di inserire una copia di un file specificato al posto della direttiva stessa prima della fase di compilazione, e sulla direttiva **#define**, che consente di creare costanti simboliche e macro. Il capitolo spiega poi la compilazione condizionale, che consente di controllare l'esecuzione delle direttive al preprocessore e la compilazione del codice del programma. Si discutono quindi l'operatore **#**, che converte il suo operando in una stringa, l'operatore **##**, che concatena due token, e le varie costanti simboliche predefinite del preprocessore (**__LINE__**, **__FILE__**, **__DATE__**, **__TIME__** e **__STDC__**). Infine si discute la macro **assert** contenuta nel file di intestazione **assert.h**: **assert** è di enorme valore durante le fasi di verifica e di debugging di un programma. Abbiamo utilizzato **assert** in diversi esempi, ma preferiamo in generale la gestione delle eccezioni, che abbiamo introdotto nel Capitolo 2.

Capitolo 7 – Il codice C ereditato. Questo capitolo presenta diversi argomenti, alcuni dei quali non si trovano normalmente nei corsi introduttivi di programmazione. Qui infatti mostriamo come reindirizzare l'input di un programma in modo che provenga da un file, reindirizzare l'output di un programma su un file, reindirizzare l'output di un programma in modo che divenga l'input di un altro programma (piping), effettuare l'appendung dell'output di un programma su un file esistente, sviluppare funzioni ad un numero variabile di argomenti, passare gli argomenti della linea di comando alla funzione **main** per utilizzarli nel programma, compilare programmi formati da componenti che si trovano su più di un file sorgente, registrare una funzione con **atexit** perché venga eseguita automaticamente al termine del programma, terminare l'esecuzione di un programma con la funzione **exit**, utilizzare i qualificatori di tipo **const** e **volatile**, specificare il tipo di una costante numerica che utilizza suffissi interi o a virgola mobile, utilizzare la libreria di gestione dei segnali per intercettare eventi inattesi, creare e utilizzare array dinamici con **calloc** e **realloc**, utilizzare le unioni per risparmiare memoria ed infine utilizzare le specifiche di linking per poter effettuare il linking del programma con codice C ereditato. Come suggerisce lo stesso titolo, questo capitolo è stato concepito espressamente per i programmatori C++ che hanno a che fare con codice C ereditato.

Capitolo 8 – La classe string e l'I/O su stream tramite stringhe. Questo capitolo parla della capacità del C++ di effettuare l'input da stringhe in memoria e l'output su stringhe in memoria, capacità detta anche *formattazione interna* o *elaborazione degli stream su stringhe*.

La classe **string** è un componente essenziale della Libreria standard. Noi abbiamo posto questo capitolo verso la fine del testo, mentre molti docenti inseriscono la discussione delle “stringhe come oggetti a tutti gli effetti” molto prima nei loro corsi. Abbiamo posto la trattazione delle stringhe in stile C nel Capitolo 5 del volume Fondamenti e in parti successive per diverse ragioni. La prima è che pensiamo che rafforzi la comprensione dei puntatori. La seconda è che prevediamo che nei prossimi anni i programmatori C++ dovranno saper leggere e modificare l’enorme quantità di codice C ereditato accumulatosi nell’ultimo quarto di secolo, e questo codice tratta le stringhe come puntatori, così come gran parte del codice C++ scritto negli anni passati. Nel Capitolo 8 parliamo dell’assegnamento, del concatenamento e del confronto di stringhe. Mostriamo come determinare varie caratteristiche di una **string** quali dimensione, capacità e se è vuota o meno. Mostriamo poi come ridimensionare una **string**, illustriamo le varie funzioni di ricerca, che consentono di ricercare una sottostringa in una **string** (effettuando la ricerca anche al contrario), come trovare la prima o l’ultima occorrenza di un carattere contenuto in una **string**, ed infine come trovare la prima o l’ultima occorrenza di un carattere non incluso in una **string**. Mostriamo infine come sostituire, eliminare e inserire caratteri in una **string** e come convertire un oggetto **string** in una stringa in stile C (**char ***).

Capitolo 9 – La libreria standard dei template (STL). Secondo noi questo capitolo è il fiore all’occhiello dell’intero volume. Ribadiamo qui che questo non è un libro sulla STL e che non troverete una discussione su di essa prima di questo capitolo. Il Capitolo 8 menziona brevemente gli iteratori, ma specifica che la loro trattazione si trova in questo capitolo. Grazie a questo capitolo il nostro corso discute adesso i quattro modelli di programmazione: procedurale, basata sugli oggetti, orientata agli oggetti e generica (con la STL). La sfida dell’insegnamento della programmazione orientata agli oggetti diverrà sempre più appassionante con il diffondersi delle librerie di classi e template, e noi pensiamo che nei prossimi decenni la crescita dei componenti riutilizzabili sarà esponenziale. I futuri corsi base di informatica dovranno presentare il linguaggio oggetto del corso, indicare come creare classi utili, fornire una panoramica delle più importanti librerie di classi esistenti e mostrare come riutilizzare questi componenti. I corsi di informatica superiori e, in realtà, tutti i corsi correlati in qualche modo all’informatica (il che al giorno d’oggi significa praticamente corsi su qualsiasi argomento) riguarderanno non solo il loro corpo di conoscenza, ma anche la discussione su come applicare le librerie di classi a quell’area di conoscenza. Al giorno d’oggi ci sono molti sforzi per supportare il riutilizzo del software su diverse piattaforme, e ciò significa che alla fine non importerà in che linguaggio sono scritte le classi, sarà comunque possibile riutilizzarle da molti linguaggi diversi.

Capitolo 10 – Le nuove funzionalità del C++ previste dallo Standard ANSI/ISO. Questo capitolo illustra diverse aggiunte apportate al linguaggio dallo Standard ANSI/ISO. Discutiamo il tipo di dato **bool** e i suoi valori **false** e **true**, che sono una rappresentazione più naturale rispetto a valori nulli e non nulli, benché questi ultimi si possano ancora utilizzare. Discutiamo poi i quattro nuovi operatori di cast: **static_cast**, **const_cast**, **reinterpret_cast** e **dynamic_cast**. Essi forniscono un meccanismo più sicuro di cast rispetto a quello in stile C. Mostriamo quindi gli spazi dei nomi, una caratteristica particolarmente importante per gli sviluppatori che lavorano a sistemi di ragguardevoli dimensioni, specialmente se utilizzano diverse librerie di classi. Gli spazi dei nomi consentono di evitare le collisioni dei nomi che in precedenza rimanevano occulte e inspiegabili. Prendiamo poi in considerazione le informazioni di tipo a run-time (RTTI), che consentono ai

programmi di controllare il tipo di un oggetto, cosa che non potevano fare in precedenza a meno che il programmatore non avesse incluso esplicitamente un codice di controllo dei tipi (non fra le migliori abitudini di programmazione). Discutiamo quindi l'utilizzo degli operatori **typeid** e **dynamic_cast** e le nuove parole chiave operatore, che sono utili ai programmatori che non amano gli operatori criptici, anche se il loro utilizzo primario si indirizza agli ambienti e ai mercati internazionali, in cui non tutti i caratteri potrebbero essere disponibili su determinate tastiere locali. Parliamo in seguito della parola chiave **explicit** che evita una chiamata automatica del compilatore ai costruttori di conversione nei casi in cui ciò non sarebbe desiderabile: i costruttori di conversione **explicit** possono essere invocati unicamente con la sintassi del costruttore, non tramite conversioni implicite. Introduciamo la parola chiave **mutable**, che consente la modifica di un membro di un oggetto **const**: in precedenza questa operazione si poteva effettuare eludendo il carattere **const** di tutto l'oggetto, operazione che naturalmente non è sicura. Infine parliamo di alcune caratteristiche che non sono nuove, ma che abbiamo scelto di non includere nella parte principale del corso perché sono relativamente complesse, come gli operatori di puntamento al membro **.*** e **->*** e l'utilizzo di classi base virtuali nell'ereditarietà multipla.